Belarusian National Technical University
Faculty «International institute of distance education»
Department «Information systems and technologies»

ELECTRONIC INSTRUCTIONAL MATERIALS AND COURSE REQUIREMENTS

COMPUTER SCIENCE

for specialty:
1-53 01 01 «Automation of technological processes and production»

Contributors:
Antsypov Nikita A., lecturer
Kazakevich Victor A., P.H.D, docent
Radkevich Andrey S., senior lecturer
Stepanov Vladimir Y., lecturer
Hvitko Evgeni A., lecturer

BNTU Minsk 2020

List of materials

Texts of lectures, teaching materials for practical classes, materials for knowledge control, elements of the syllabus of the discipline.

Explanatory note

The purpose of the electronic instructional materials and course requirements by the discipline «Computer science» (EIMCR) is to develop theoretical systemic and practical knowledge in different fields of Computer science.

Features of structuring and submission of educational material:

EIMCR includes the following sections: theoretical, practical, knowledge control, auxiliary.

The theoretical section presents lecture material in accordance with the main sections and topics of the syllabus.

The practical section of the EIMCR contains materials for conducting practical classes aimed to develop modern computational thinking, basic skills in computing and making decisions in the field of the fundamentals of computer theory and many computer science fields.

The knowledge control section of the EIMCR contains: guidelines for the implementation of the control work aimed at developing the skills of independent work on the course under study, developing the skills of selecting, analyzing and writing out the necessary material, as well as the correct execution of the tasks; list of questions for the credit by the discipline.

The auxiliary section of the EIMCR contains the following elements of the syllabus: explanatory note; thematic lectures plan; tables of distribution of classroom hours by topics and informational and methodological part.

EIMCR contains active links to quickly find the necessary material.

CONTENT

PART 1. SYLLABUS

Module name	Computer Science
Contact hours	Lectures – 17; Laboratory experiences – 51; Independent learning – 42
Credits	Credits-3
Class Times	Lectures – 2hrs per session – twice a week (4 hours per week)
Recommended reading	**Required text:** - Introduction to Computers (Shelly Cashman Series) by Gary B. Shelly, Steven M. Freund and Misty E. Vermaat. - Microsoft Office: Introductory (Shelly Cashman Series(r) Office) by Gary B. Shelly, Misty E. Vermaat. **Additional reading:** https://drive.google.com/drive/folders/1ubTisMWVYUaKE3gFBT9HjHgyo0b2pNlz?usp=sharing
Learning resources	https://drive.google.com/drive/folders/1oJloe0YGapyNb2Sidd9kXLCpsBPxrmHi?usp=sharing
Module Description	This is an introductory module in information processing and fundamental computer concepts. The module is intended for individuals with no previous computing experience or competence. It includes the history of computers, information about using computers today, the basic components of computers and computer terminology and laboratory experiences using computer software. This module aims to encourage the development of computational thinking, that is thinking about what can be computed and how by the use of abstraction and decomposition. It includes consideration of the data required. Learning computational thinking involves learning to program, by writing computer code, because this is the means by which computational thinking is expressed. A continuation of the course is the discipline of Information Technology.
Learning Outcomes	After learning of Computer Science course students will be able to: - explain the many concepts in Information representation; - understand and explain how the hardware works in basic level; - understand the Processor fundamentals; - exposed by the System software; - work with communication and Internet technologies;

	- use the Microsoft Office suite of applications; - explain the many concepts in security, privacy and data integrity; - understand and explain why ethics and ownership are so important today; - explain the many concepts in databases and data modelling. The module also includes the study of the foundational principles and practices of computation and computational thinking and their application in the design and development of computer systems, the learning of fundamental concepts of computer programing and using a visual development environment.
Teaching methodology	Lectures, practical on MS Office / Application packages in the lab supplemented by group activities. PowerPoint presentations and Google drive storage to supplement lectures.
Institution Outcomes	- to develop computational thinking; - to develop an understanding of the main principles of solving problems using computers; - to develop an understanding that every computer system is made up of subsystems, which in turn consist of further subsystems; - to develop an understanding of the component parts of computer systems and how they interrelate, including software, data, hardware, communications, etc.; - to acquire skills of work with specialized software and office software; - to acquire the skills necessary to apply this understanding to develop computer-based solutions to problems; - to enable students to succeed in a university learning environment through acquisition of appropriate learning and knowledge seeking skills; - to improved communication skills for the students to become effective learners in their chosen higher education institution; - to become more prepared to compete in the world marketplace.

Delivery schedule

Section	Topics	Lecture/ examination hours
1.	*Introduction to Computer Science* Computer science is the subject that studies what computers can do. Information theory. An algorithm. Cryptography. Designing computers. Computer architecture. Programming language. The operating system.	1

2.	*Brief History of Computer* Explains the importance of computer fluency. Turing machine. Computability Theory. Optimisation problems. Transistors.	1
3.	*Information and data. Representation of information* Boolean Logic. Representing Numbers and Letters. Binary representation.	1
4.	*Arithmetic and Logic Unit. The CPU and Von Neumann Architecture.* Most of the computers of today's world run on the von Neumann architecture. Input comes in stores in a memory; the input becomes the command and ready to process. The processor fetches the command from memory. There are two parts in the process. First CU control unit, second ALU arithmetic logic unit.	1
5.	*Intro to Algorithms and Data Structures* Algorithm is the specific steps used to complete the computation. Some algorithms are better than others even if they produce equal results. Generally, the fewer steps it takes to compute, the better it is, though sometimes we care about other factors, like how much memory it uses. The "Big O" computational complexity. Array. Matrix. Struct. Queues and stacks.	1
6.	*Software Engineering* To build huge programs programmers use a set of tools and practices. Breaking big programs into smaller functions allows many people to work simultaneously. They don't have to worry about the whole thing, just the function they're working on. Object Oriented Programming. Functional units. Application Programming Interface. Integrated Development Environments. Quality Assurance testing.	1
7.	*Operating system. System software and application software.* A computer's operating system is one of the most important "parts" of the computer. Almost every type of computer – including mobile telephones,	1

	video game systems, E-book readers, etc. Windows, Linux, Mac OS X. A program or group of programs designed for end users. Application software can be divided into two general classes: systems software and applications software.	
8.	*Computer Memory, Processing, and Storage.* A motherboard - The biggest component of the machine (It's a place where you stuck everything on). A Processor: Where all the calculations are made. A RAM: A Temporary memory storage unit for the machine. A HDD: Hard Drive, save files. Memory and storage are two terms that are regularly used to mean the same thing in computer technology. The reason for this is that some devices designed as memory also act as storage devices.	1
9.	*Input and Output devices. Files and File Systems. Compression.* Considers the devices which are used to input the data and the programs in the computer are known as "Input Devices". Input device can read data and convert them to a form that a computer can use. Output Device can produce the final product of machine processing into a form usable by humans. There are many types of files, like text files, music files, photos and videos. File format. NTFS. FAT32. exFAT. Fragmentation. Defragmentation. ZIP. RAR.	1
10.	*Linux File System and Linux Command Line* Linux File System is a layer which is under the operating system that handles the positioning of your data on the storage, without it; the system cannot knows which file starts from where and ends where. Ext, Ext2, Ext3, Ext4. Linux File System Directories. Command line is one of the many strengths of Linux based systems. Basics Commands: ls, pwd, cd, mv, rm, mkdir.	1
11.	*Computers Networks & The Internet* Data communications refers to the transmission of	1

	digital data between two or more computers and a computer network or data network is a telecommunications network that allows computers to exchange data. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet.	
12.	*Cybersecurity and Cryptography* The crime that involves and uses computer devices and Internet, is known as cybercrime. Cybercrime can be committed against an individual or a group; it can also be committed against government and private organizations. Cryptography is a method of protecting information and communications through the use of codes so that only those for whom the information is intended can read and process it.	1
13.	*Machine Learning & Artificial Intelligence* Machine Learning is the learning in which machine can learn by its own without being explicitly programmed. It is an application of AI that provide system the ability to automatically learn and improve from experience. Here we can generate a program by integrating input and output of that program.	1
14.	*Text processors. Office suite of applications. Microsoft Word. Formatting, Objects & Tools.* Microsoft Office is a family of client software, server software, and services developed by Microsoft. Microsoft Word allows writing with confidence, knowing intelligent technology can help with spelling, grammar and even stylistic writing suggestions.	1
15.	Introduction to Microsoft Excel, Features and tools, Managing, Worksheets and Formatting. *It can do calculations and graphics. For example, it can make charts and other pictures from data tables. It also has a macro programming language called Visual Basic for Applications (VBA).* *Formulas, Functions and Graphs / Charts.* *Inside the Microsoft Excel.*	1

16.	Introduction to Microsoft PowerPoint and Features. *In PowerPoint, as in most other presentation software, text, graphics, movies, and other objects are positioned on individual pages or "slides".* *Microsoft PowerPoint is a part of Microsoft Office.* *Graphics Object, tools and Charts, Custom Animations.* *Inside the Microsoft PowerPoint.*	1
17.	Databases. Introduction to Microsoft Access and Features. *Access is an easy-to-use tool for creating business applications, from templates or from scratch. With its design tools, Access can help to create highly functional applications in a minimal amount of time.*	1
Final Exam.		1,5

Academic policies

Attendance classwork/ assessments/ examinations	Absences will adversely affect your final grade. Please note that participation points cannot be earned if you are not present in the classroom. If you miss a class, you are responsible for finding out from someone else in class what you missed and for making up any work you missed. Late work may not be accepted or may lose points, at the discretion of your lecturer. NOTE: Excused absences are possible from doctor / hospital (if you are ill) or from Administration of IIDE BNTU. If a student misses any of the Examinations due to a valid reason accepted by the Institute, then he / she need to obtain the written authorisation from the IIDE Administration to complete the same as a make-up exam.
Cheating and Plagiarism	Plagiarism and cheating are serious academic offenses. Plagiarism means «knowingly copying another's work or ideas and calling them one's own or not giving proper credit or citation». This includes copying sections or entire papers from printed or electronic sources as well as handing in papers, homework or assignments written by others for you, or by students for other classes, or purchasing academic papers. It also includes any attempted copying from fellow classmates inside or outside of the classroom, or allowing another student to copy or use your work. Plagiarism and cheating are dishonest and will not be tolerated.

	Students caught plagiarising or cheating may be subject to sanctions. If you ever have any questions about this issue, please consult with IIDE Administration or talk to your lecturer for more information.
Exams & Written Assignments	Students are always responsible for turning in any assigned work ON TIME, and missed exams will not be allowed to be made up unless individual arrangements are made with the lecturer. The lecturer has the right to assign a zero grade to any assignment that is not turned in by the deadline.
Course Withdrawal	If you are thinking of withdrawing from class, you should first explain why you want to withdraw with your lecturer. Perhaps there is a way to complete the course with a good grade. Withdrawal forms are available from the IIDE Administration.
Mobile Phones	Students are **not allowed** to use mobile phones during tests or exam inside the classroom. Students are advised to leave the mobile phones at home or TURN it OFF during teachings.
Laptop computers	Students are allowed to use Laptops in the class with the prior approval from the respective lecturer teaching the module.
Calculators	Allowed.

PART 2. THEORETICAL BASICS.

Contributors of EIMCR are not pretended on any copyrights of following materials.

Lecture 1 Introduction to Computer Science

We built computers to expand our brains. Originally scientists build computers to solve arithmetic, but they turned out to be incredibly useful for many other things as well: running the entire internet, lifelike graphics, artificial brains or simulating the Universe, but amazingly all of it boils down to just flipping zeros and ones. Computers have become smaller and more powerful at an incredible rate. There is more computing power in your cell phone then there was in the entire world in the mid 60s. And the entire Apollo moon landing could have been run on a couple of Nintendos.

Computer science is the subject that studies what computers can do. It is diverse and overlapping field but I'm going to split it into three parts. The fundamental theory of computer science, computer engineering, and Applications. We'll start with the father of theoretical computer science: Alan Turing, who formalized the concept of a Turing machine which is a simple description of a general purpose computer. People came up with other designs for computing machines but they are all equivalent to a Turing machine which makes it the foundation of computer science. A Turing machine contains several parts: An infinitely long tape that is split into cells containing symbols. There is also a head that can read and write symbols to the tape, a state register that stores the state of the head and a list of possible instructions. In today's computers the tape is like the working memory or RAM, the head is the central processing unit and the list of instructions is held in the computer's memory. Even though a Turing machine is a simple set of rules, it's incredibly powerful, and this is essentially what all computers do nowadays. Although our computers obviously have a few more parts like permanent storage and all the other components. Every problem that is computable by a Turing machine is computable using Lambda calculus which is the basis of research in programming languages.

Computability Theory attempts to classify what is and isn't computable. There are some problems that due to their very nature, can never be solved by a computer, a famous example is the halting problem where you try to predict whether a program will stop running or carry on forever. There are programs where this is impossible to answer by a computer or a human. Many problems are theoretically solvable but in practice take too much memory or more steps than lifetime of the Universe to solve, and computational complexity attempts to categorize these problems according to how they scale. There are many different classes of complexity and many classes of problem that fall into each type. There are a lot of real world problems that fall into these impossible to solve categories, but fortunately computer scientists have a bunch

of sneaky tricks where you can fudge things and get pretty good answers but you'll never know if they are the best answer.

An algorithm is a set of instructions independent of the hardware or programming language designed to solve a particular problem. It is kind of like a recipe of how to build a program and a lot of work is put into developing algorithms to get the best out of computers. Different algorithms can get to the same final result, like sorting a random set of numbers into order, but some algorithms are much more efficient than others, this is studied in O(n) complexity.

Information theory studies the properties of information and how it can be measured, stored and communicated. One application of this is how well you can compress data, making it take up less memory while preserving all or most of the information but there are lots of other applications. Related to information theory is coding theory.

And Cryptography is obviously very important for keeping information sent over the internet secret. There are many different encryption schemes which scramble data and usually rely on some very complex mathematical problem to keep the information locked up. These are the main branches of theoretical computer science although there are many, more I don't have time to go into Logic, Graph Theory, Computational Geometry, Automata Theory, Quantum Computation, Parallel Programming, Formal Methods and Data structures, but let's move on to computer engineering.

Designing computers is difficult because they have to do so many different things. Designers need to try and make sure they are capable of solving many different kinds of problem as optimally as possible. Every single task that is run on the computer goes through the core of the computer: the CPU. When you are doing lots of different things at the same time, the CPU needs to switch back and forth between these jobs to make sure everything gets done in a reasonable time. This is controlled by a scheduler, which chooses what to do when and tries to get through the tasks in the most efficient way, which can be very difficult problem. Multiprocessing helps speed things up because the CPU has several cores that can execute multiple jobs in parallel. But this makes the job of the scheduler even more complex.

Computer architecture is how a processor is designed to perform tasks and different architectures are good at different things. CPUs are general purpose, GPUs are optimized for graphics and FPGAs can be programmed to be very fast at a very narrow range of task. On top of the raw hardware there are many layers of software, written by programmers using many different programming languages.

A programming language is how humans tell a computer what to do and they vary greatly depending on the job at hand from low level languages like assembly through to high level languages like python or JavaScript for coding websites and apps. In general, the closer a language is to the hardware, the more difficult it is for humans to use. At all stages of this hierarchy the code that programmers write needs to be turned into raw CPU instructions and this is done by one or several programs called compilers. Designing programming languages and compilers is a big deal, because they are the tool that software engineers use to make everything and so they

need to be as easy to use as possible but also be versatile enough to allow the programmers to build their crazy ideas.

The operating system is the most important piece of software on the computer as it is what we interact with and it controls how all of the other programs are run on the hardware, and engineering a good operating system is a huge challenge. This brings us to software engineering: writing bundles of instructions telling the computer what to do. Building good software is an art form because you have to translate your creative ideas into logical instructions in a specific language, make it as efficient as possible to run and as free of errors as you can. So there are many best practices and design philosophies that people follow.

Some other important areas are getting many computers to communicate and work together to solve problems. Storing and retrieving large amounts of data. Determining how well computer systems are performing at specific tasks, and creating highly detailed and realistic graphics. Now we get to a really cool part of computer science, getting computers to solve real world problems. These technologies underlie a lot of the programs, apps and websites we use. When you are going on vacation and you want to get the best trip for the money you are solving an optimization problem.

Optimization problems appear everywhere and finding the best path or most efficient combination of parts can save businesses millions of dollars. This is related to Boolean Satisfiability where you attempt to work out if a logic formula can be satisfied or not. This was the first problem proved to be NP-complete and so widely considered to be impossible to solve, but amazing development of new SAT solvers means that huge SAT problems are solved routinely today especially in artificial intelligence. Computers extend our brains multiply our cognitive abilities. The forefront of computer science research is developing computer systems that can think for themselves: Artificial Intelligence.

There are many avenues that AI research takes, the most prominent of which is machine learning which aims to develop algorithms and techniques to enable computers to learn from large amounts of data and then use what they’ve learned to do something useful like make decisions or classify things. And there are many different types of machine learning. Closely related are fields like computer vision, trying to make computers able to see objects in images like we do, which uses image processing techniques.

Natural language processing aims to get computers to understand and communicate using human language, or to process large amounts of data in the form of words for analysis. This commonly uses another field called knowledge representation where data is organized according to their relationships, like words with similar meanings are clustered together. Machine learning algorithms have improved because of the large amount of data we give them. Big data looks at how to manage and analyze large amounts of data to get value from it. And will get even more data from the Internet of Things, adding data collection and communications to everyday objects.

Hacking is not a traditional academic discipline but definitely worth mentioning. Trying to find weaknesses in computer systems, and take advantage of them without being noticed. Computational Science uses computers to help answer scientific questions from fundamental physics to neuroscience, and often makes use of Super Computing which throws the weight of the world's most powerful computers at very large problems, often in the area of Simulation. Then there is Human Computer Interaction which looks at how to design computer systems to be easy and intuitive to use.

Virtual reality, Augmented Reality and Telepresence enhancing or replacing our experience of reality. And finally Robotics which gives computers a physical embodiment, from a Roomba to trying to make intelligent human like machines. So that is my Map of Computer Science, a field that is still developing as fast as it ever has despite that fact that the underlying hardware is hitting some hard limits as we struggle to miniaturize transistors anymore, so lots of people are working on other kinds of computers to try and overcome this problem. Computers have had an absolutely huge impact on human society and so it is going to be interesting to see where this technology goes in the next one hundred years. Who knows, perhaps one day, we'll all be computers.

Brief History of Computer

So, computers really have allowed us to do some pretty amazing things - think global telecommunications, international commerce, global transportation, breakthroughs in medicine, distributed education, online shopping, online dating, just the Internet in general.

Computers are allowing us to explore our own world and other worlds, and of course some seemingly mundane things like permitting us to spy on our pets from work or communicate with our friends in a nearly indecipherable stream of emoji! But don't call computers magical.

First of all, we are going to look are the history of computers… even before we had electricity. We're going retrace the design decisions that have given us our present-day components.

We are living in a time likely to be remembered as the Electronic Age. With billions of transistors in just your smartphones, computers can seem pretty complicated, but really, they're just simple machines that perform complex actions through many layers of abstraction.

Since the start of civilization itself, humans have had an increasing need for special devices to help manage laborious tasks, and as the scale of society continued to grow, these computational devices began to play a crucial role in amplifying our mental abilities.

The earliest recognized device for computing was the abacus, invented in Mesopotamia around 2500 BCE. It's essentially a hand operated calculator, that helps add and subtract many numbers. It also stores the current state of the computation, much like your hard drive does today. The abacus was created because, the scale of society had become greater than what a single person could keep and manipulate in their mind. There might be thousands of people in a village or tens of thousands of

cattle. There are many variants of the abacus, but let's look at a really basic version with each row representing a different power of ten. So each bead on the bottom row represents a single unit, in the next row they represent 10, the row above 100, and so on. Let's say we have 3 heads of cattle represented by 3 beads on the bottom row on the right side. If we were to buy 4 more cattle we would just slide 4 more beads to the right for a total of 7. But if we were to add 5 more after the first 3 we would run out of beads, so we would slide everything back to the left, slide one bead on the second row to the right, representing ten, and then add the final 2 beads on the bottom row for a total of 12.

This is particularly useful with large numbers. So if we were to add 1,251 we would just add 1 to the bottom row, 5 to the second row, 2 to the third row, and 1 to the fourth row - we don't have to add in our head and the abacus stores the total for us.

Over the next 4000 years, humans developed all sorts of clever computing devices, like the astrolabe, which enabled ships to calculate their latitude at sea. Or the slide rule, for assisting with multiplication and division. And there are literally hundreds of types of clocks created that could be used to calculate sunrise, tides, positions of celestial bodies, and even just the time.

Each one of these devices made something that was previously laborious to calculate much faster, easier, and often more accurate — it lowered the barrier to entry, and at the same time, amplified our mental abilities.

However, none of these devices were called "computers". The earliest documented use of the word "computer" is from 1613, in a book by Richard Braithwaite. And it wasn't a machine at all - it was a job title. In those days, computer was a person who did calculations, sometimes with the help of machines, but often not. This job title persisted until the late 1800s, when the meaning of computer started shifting to refer to devices.

Notable among these devices was the Step Reckoner, built by German polymath Gottfried Leibniz in 1694. Leibniz said "... it is beneath the dignity of excellent men to waste their time in calculation when any peasant could do the work just as accurately with the aid of a machine."

It worked kind of like the odometer in your car, which is really just a machine for adding up the number of miles your car has driven. The device had a series of gears that turned; each gear had ten teeth, to represent the digits from 0 to 9. Whenever a gear bypassed nine, it rotated back to 0 and advanced the adjacent gear by one tooth. Kind of like when hitting 10 on that basic abacus. This worked in reverse when doing subtraction, too. With some clever mechanical tricks, the Step Reckoner was also able to multiply and divide numbers. Multiplications and divisions are really just many additions and subtractions.

The Step Reckoner was able to do this in an automated way, and was the first machine that could do all four of these operations. And this design was so successful it was used for the next three centuries of calculator design. Unfortunately, even with mechanical calculators, most real world problems required many steps of computation before an answer was determined. It could take hours or days to

generate a single result. Also, these hand-crafted machines were expensive, and not accessible to most of the population. So, before 20th century, most people experienced computing through pre-computed tables assembled by those amazing “human computers” we talked about.

Speed and accuracy is particularly important on the battlefield, and so militaries were among the first to apply computing to complex problems. A particularly difficult problem is accurately firing artillery shells, which by the 1800s could travel well over a kilometer (or a bit more than half a mile).

Add to this varying wind conditions, temperature, and atmospheric pressure, and even hitting something as large as a ship was difficult. Range Tables were created that allowed gunners to look up environmental conditions and the distance they wanted to fire, and the table would tell them the angle to set the canon. These Range Tables worked so well, they were used well into World War Two. The problem was, if you changed the design of the cannon or of the shell, a whole new table had to be computed, which was massively time consuming and inevitably led to errors.

Charles Babbage acknowledged this problem in 1822 in a paper to the Royal Astronomical Society entitled: “Note on the application of machinery to the computation of astronomical and mathematical tables". Let’s go to the thought bubble. Charles Babbage proposed a new mechanical device called the Difference Engine, a much more complex machine that could approximate polynomials. Polynomials describe the relationship between several variables - like range and air pressure, or amount of pizza Andrew eats and happiness. Polynomials could also be used to approximate logarithmic and trigonometric functions, which are a real hassle to calculate by hand.

Babbage started construction in 1823, and over the next two decades, tried to fabricate and assemble the 25,000 components, collectively weighing around 15 tons. Unfortunately, the project was ultimately abandoned. But, in 1991, historians finished constructing a Difference Engine based on Babbage's drawings and writings - and it worked! But more importantly, during construction of the Difference Engine, Babbage imagined an even more complex machine - the Analytical Engine. Unlike the Difference Engine, Step Reckoner and all other computational devices before it - the Analytical Engine was a “general purpose computer”. It could be used for many things, not just one particular computation; it could be given data and run operations in sequence; it had memory and even a primitive printer. Like the Difference Engine, it was ahead of its time, and was never fully constructed. However, the idea of an “automatic computer” – one that could guide itself through a series of operations automatically, was a huge deal, and would foreshadow computer programs.

English mathematician Ada Lovelace wrote hypothetical programs for the Analytical Engine, saying, “A new, a vast, and a powerful language is developed for the future use of analysis.” For her work, Ada is often considered the world’s first programmer. The Analytical Engine would inspire, arguably, the first generation of computer scientists, who incorporated many of Babbage’s ideas in their machines. This is why Babbage is often considered the "father of computing".

So by the end of the 19th century, computing devices were used for special purpose tasks in the sciences and engineering, but rarely seen in business, government or domestic life. However, the US government faced a serious problem for its 1890 census that demanded the kind of efficiency that only computers could provide.

The US Constitution requires that a census be conducted every ten years, for the purposes of distributing federal funds, representation in congress, and good stuff like that. And by 1880, the US population was booming, mostly due to immigration. That census took seven years to manually compile and by the time it was completed, it was already out of date – and it was predicted that the 1890 census would take 13 years to compute. That's a little problematic when it's required every decade!

The Census bureau turned to Herman Hollerith, who had built a tabulating machine. His machine was "electro-mechanical" – it used traditional mechanical systems for keeping count, like Leibniz's Step Reckoner – but coupled them with electrically-powered components. Hollerith's machine used punch cards which were paper cards with a grid of locations that can be punched out to represent data.

For example, there was a series of holes for marital status. If you were married, you would punch out the married spot, then when the card was inserted into Hollerith's machine, little metal pins would come down over the card – if a spot was punched out, the pin would pass through the hole in the paper and into a little vial of mercury, which completed the circuit. This now completed circuit powered an electric motor, which turned a gear to add one, in this case, to the "married" total.

Hollerith's machine was roughly 10x faster than manual tabulations, and the Census was completed in just two and a half years - saving the census office millions of dollars. Businesses began recognizing the value of computing, and saw its potential to boost profits by improving labor- and data-intensive tasks, like accounting, insurance appraisals, and inventory management. To meet this demand, Hollerith founded The Tabulating Machine Company, which later merged with other machine makers in 1924 to become The International Business Machines Corporation or IBM - which you've probably heard of.

These electro-mechanical "business machines" were a huge success, transforming commerce and government, and by the mid-1900s, the explosion in world population and the rise of globalized trade demanded even faster and more flexible tools for processing data, setting the stage for digital computers, which we'll talk about next.

One of the largest electro-mechanical computers built was the Harvard Mark I, completed in 1944 by IBM for the Allies during World War 2. It contained 765,000 components, three million connections, and five hundred miles of wire. To keep its internal mechanics synchronized, it used a 50-foot shaft running right through the machine driven by a five horsepower motor. One of the earliest uses for this technology was running simulations for the Manhattan Project. The brains of these huge electro-mechanical beasts were relays: electrically-controlled mechanical switches. In a relay, there is a control wire that determines whether a circuit is opened or closed. The control wire connects to a coil of wire inside the relay. When current

flows through the coil, an electromagnetic field is created, which in turn, attracts a metal arm inside the relay, snapping it shut and completing the circuit. You can think of a relay like a water faucet. The control wire is like the faucet handle. Open the faucet, and water flows through the pipe. Close the faucet, and the flow of water stops.

Relays are doing the same thing, just with electrons instead of water. The controlled circuit can then connect to other circuits, or to something like a motor, which might increment a count on a gear, like in Hollerith's tabulating machine we talked about last episode. Unfortunately, the mechanical arm inside of a relay *has mass*, and therefore can't move instantly between opened and closed states. A good relay in the 1940's might be able to flick back and forth fifty times in a second. That might seem pretty fast, but it's not fast enough to be useful at solving large, complex problems.

The Harvard Mark I could do 3 additions or subtractions per second; multiplications took 6 seconds, and divisions took 15. And more complex operations, like a trigonometric function, could take over a minute. In addition to slow switching speed, another limitation was wear and tear. Anything mechanical that moves will wear over time. Some things break entirely, and other things start getting sticky, slow, and just plain unreliable. And as the number of relays increases, the probability of a failure increases too. The Harvard Mark I had roughly 3500 relays. Even if you assume a relay has an operational life of 10 years, this would mean you'd have to replace, on average, one faulty relay every day! That's a big problem when you are in the middle of running some important, multi-day calculation.

And that's not all engineers had to contend with. These huge, dark, and warm machines also attracted insects. In September 1947, operators on the Harvard Mark II pulled a dead moth from a malfunctioning relay. Grace Hopper who we'll talk more about in a later noted, "From then on, when anything went wrong with a computer, we said it had bugs in it." And that's where we get the term computer bug.

It was clear that a faster, more reliable alternative to electro-mechanical relays was needed if computing was going to advance further, and fortunately that alternative already existed!

In 1904, English physicist John Ambrose Fleming developed a new electrical component called a thermionic valve, which housed two electrodes inside an airtight glass bulb - this was the first vacuum tube. One of the electrodes could be heated, which would cause it to emit electrons – a process called thermionic emission. The other electrode could then attract these electrons to create the flow of our electric faucet, but only if it was positively charged - if it had a negative or neutral charge, the electrons would no longer be attracted across the vacuum so no current would flow. An electronic component that permits the one-way flow of current is called a diode, but what was really needed was a switch to help turn this flow on and off. Luckily, shortly after, in 1906, American inventor Lee de Forest added a third "control" electrode that sits between the two electrodes in Fleming's design. By applying a positive charge to the control electrode, it would permit the flow of electrons as before. But if the control electrode was given a negative charge, it would prevent the

flow of electrons. So by manipulating the control wire, one could open or close the circuit. It's pretty much the same thing as a relay - but importantly, vacuum tubes have no moving parts. This meant there was less wear, and more importantly, they could switch thousands of times per second. These triode vacuum tubes would become the basis of radio, long distance telephone, and many other electronic devices for nearly a half century.

The first large-scale use of vacuum tubes for computing was the Colossus Mk 1 designed by engineer Tommy Flowers and completed in December of 1943. The Colossus was installed at Bletchley Park, in the UK, and helped to decrypt Nazi communications. Colossus is regarded as the first programmable, electronic computer.

By the 1950's, even vacuum-tube-based computing was reaching its limits. To reduce cost and size, as well as improve reliability and speed, a radical new electronic switch would be needed. In 1947, Bell Laboratory scientists John Bardeen, Walter Brattain, and William Shockley invented the transistor, and with it, a whole new era of computing was born! The physics behind transistors is pretty complex, relying on quantum mechanics, so we're going to stick to the basics. A transistor is just like a relay or vacuum tube - it's a switch that can be opened or closed by applying electrical power via a control wire. Typically, transistors have two electrodes separated by a material that sometimes can conduct electricity, and other times resist it – a semiconductor. In this case, the control wire attaches to a "gate" electrode. By changing the electrical charge of the gate, the conductivity of the semiconducting material can be manipulated, allowing current to flow or be stopped – like the water faucet analogy we discussed earlier.

Even the very first transistor at Bell Labs showed tremendous promise – it could switch between on and off states 10,000 times per second. Further, unlike vacuum tubes made of glass and with carefully suspended, fragile components, transistors were solid material known as a solid state component. Almost immediately, transistors could be made smaller than the smallest possible relays or vacuum tubes. This led to dramatically smaller and cheaper computers, like the IBM 608, released in 1957 – the first fully transistor-powered, commercially-available computer.

Lecture 2 Boolean Logic

Last lecture, we talked about how computers evolved from electromechanical devices, that often had decimal representations of numbers – like those represented by teeth on a gear – to electronic computers with transistors that can turn the flow of electricity on or off. And fortunately, even with just two states of electricity, we can represent important information. We call this representation Binary -- which literally means "of two states". You might think two states isn't a lot to work with, and you'd be right! But, it's exactly what you need for representing the values "true" and "false". In computers, an "on" state, when electricity is flowing, represents true. The off state, no electricity flowing, represents false. We can also write binary as 1's and

0’s instead of true’s and false’s – they are just different expressions of the same signal – but we’ll talk more about that in the next part of the lecture.

One of the reasons why computers use binary is that an entire branch of mathematics already existed that dealt exclusively with true and false values. And it had figured out all of the necessary rules and operations for manipulating them. It's called Boolean Algebra!

In “regular” algebra -- the type you probably learned in high school -- the values of variables are numbers, and operations on those numbers are things like addition and multiplication. But in Boolean Algebra, the values of variables are true and false, and the operations are logical. There are three fundamental operations in Boolean Algebra: a NOT, an AND, and an OR operation. And these operations turn out to be really useful so we’re going to look at them individually.

A NOT takes a single boolean value, either true or false, and negates it. It flips true to false, and false to true. We can write out a little logic table that shows the original value under Input, and the outcome after applying the operation under Output.

The AND Boolean operation takes two inputs, but still has a single output. In this case the output is only true if both inputs are true.

Another useful boolean operation in computation is OR -- where only one input has to be true for the output to be true.

The last boolean operation is called an Exclusive OR - or XOR for short. XOR is like a regular OR, but with one difference: if both inputs are true, the XOR is false. The only time an XOR is true is when one input is true and the other input is false.

Representing Numbers and Letters with Binary.

Now we’re going to talk about how computers store and represent numerical data. So, as I mentioned earlier, a single binary value can be used to represent a number. Instead of true and false, we can call these two states 1 and 0 which is actually incredibly useful. And if we want to represent larger things we just need to add more binary digits. This works exactly the same way as the decimal numbers that we’re all familiar with. With decimal numbers there are "only" 10 possible values a single digit can be; 0 through 9, and to get numbers larger than 9 we just start adding more digits to the front. We can do the same with binary.

For example, let’s take the number two hundred and sixty three. What does this number actually represent? Well, it means we’ve got 2 one-hundreds, 6 tens, and 3 ones. If you add those all together, we’ve got 263. Notice how each column has a different multiplier. In this case, it’s 100, 10, and 1. Each multiplier is ten times larger than the one to the right. That's because each column has ten possible digits to work with, 0 through 9, after which you have to carry one to the next column. For this reason, it’s called base-ten notation, also called decimal since deci means ten.

AND Binary works exactly the same way, it’s just base-two. That’s because there are only two possible digits in binary – 1 and 0. This means that each multiplier has to be two times larger than the column to its right. Instead of hundreds, tens, and ones, we now have fours, twos and ones. Take for example the binary number: 101. This means we have 1 four, 0 twos, and 1 one. Add those all together and we’ve got

the number 5 in base ten. But to represent larger numbers, binary needs a lot more digits. Take this number in binary 10110111. We can convert it to decimal in the same way. We have 1 times 128, 0 x 64, 1 x 32, 1 x 16, 0 x 8, 1 x 4, 1 x 2, and 1 x 1. Which all adds up to 183. Math with binary numbers isn't hard either.

Take for example decimal addition of 183 plus 19. First we add 3 + 9, that's 12, so we put 2 as the sum and carry 1 to the ten's column. Now we add 8 plus 1 plus the 1 we carried, thats 10, so the sum is 0 carry 1. Finally, we add 1 plus the 1 we carried, which equals 2. So the total sum is 202. Here's the same sum but in binary. Just as before, we start with the ones column. Adding 1+1 results in 2, even in binary. But, there is no symbol "2" so we use 10 and put 0 as our sum and carry the 1. Just like in our decimal example. 1 plus 1, plus the 1 carried, equals 3 or 11 in binary, so we put the sum as 1 and we carry 1 again, and so on. We end up with 11001010, which is the same as the number 202 in base ten.

Each of these binary digits, 1 or 0, is called a "bit". So in these last few examples, we were using 8-bit numbers with their lowest value of zero and highest value is 255, which requires all 8 bits to be set to 1. Thats 256 different values, or 2 to the 8th power. You might have heard of 8-bit computers, or 8-bit graphics or audio. These were computers that did most of their operations in chunks of 8 bits. But 256 different values isn't a lot to work with, so it meant things like 8-bit games were limited to 256 different colors for their graphics. And 8-bits is such a common size in computing, it has a special word: a byte. A byte is 8 bits.

You've heard of kilobytes, megabytes, gigabytes and so on. These prefixes denote different scales of data. Just like one kilogram is a thousand grams, 1 kilobyte is a thousand bytes…. or really 8000 bits. Mega is a million bytes (MB), and giga is a billion bytes (GB). Today you have a hard drive that has 1 terabyte (TB) of storage. That's 8 trillion ones and zeros. But that's not always true. In binary, a kilobyte has two to the power of 10 bytes, or 1024. 1000 is also right when talking about kilobytes, but we should acknowledge it isn't the only correct definition.

You've probably also heard the term 32-bit or 64-bit computers – you're almost certainly using one right now. What this means is that they operate in chunks of 32 or 64 bits. That's a lot of bits! The largest number you can represent with 32 bits is just under 4.3 billion. Which is thirty-two 1's in binary. Of course, not everything is a positive number. So we need a way to represent positive and negative numbers. Most computers use the first bit for the sign: 1 for negative, 0 for positive numbers, and then use the remaining 31 bits for the number itself. That gives us a range of roughly plus or minus two billion. While this is a pretty big range of numbers, it's not enough for many tasks. This is why 64-bit numbers are useful. The largest value a 64-bit number can represent is around 9.2 quintillion! Most importantly, as we'll discuss in a later lecture, computers must label locations in their memory, known as addresses, in order to store and retrieve values. As computer memory has grown to gigabytes and terabytes – that's trillions of bytes – it was necessary to have 64-bit memory addresses as well.

In addition to negative and positive numbers, computers must deal with numbers that are not whole numbers, like 12.7 and 3.14. These are called "floating point"

numbers, because the decimal point can float around in the middle of number. Several methods have been developed to represent floating point numbers. The most common of which is the IEEE 754 standard. This standard stores decimal values sort of like scientific notation. For example, 625.9 can be written as 0.6259 x 10^3 (0.6259 times 10 to the power of 3). There are two important numbers here: the .6259 is called the significand. And 3 is the exponent.

In a 32-bit floating point number, the first bit is used for the sign of the number – positive or negative. The next 8 bits are used to store the exponent and the remaining 23 bits are used to store the significand. Ok, we've talked a lot about numbers, but your name is probably composed of letters, so it's really useful for computers to also have a way to represent text. However, rather than have a special form of storage for letters, computers simply use numbers to represent letters.

Enter ASCII, the American Standard Code for Information Interchange. Invented in 1963, ASCII was a 7-bit code, enough to store 128 different values. With this expanded range, it could encode capital letters, lowercase letters, digits 0 through 9, and symbols like the @ sign and punctuation marks. For example, a lowercase 'a' is represented by the number 97, while a capital 'A' is 65. A colon is 58 and a closed parenthesis is 41. ASCII even had a selection of special command codes, such as a newline character to tell the computer where to wrap a line to the next row. Because ASCII was such an early standard, it became widely used, and critically, allowed different computers built by different companies to exchange data. Fortunately, there are 8 bits in a byte, not 7, and it soon became popular to use codes 128 through 255, previously unused, for "national" characters. In the US, those extra numbers were largely used to encode additional symbols, like mathematical notation, graphical elements, and common accented characters. On the other hand, while the Latin characters were used universally, Russian computers used the extra codes to encode Cyrillic characters, and Greek computers, Greek letters, and so on. And national character codes worked pretty well for most countries. The problem was, if you opened an email written in Latvian on a Turkish computer, the result was completely incomprehensible. And things totally broke with the rise of computing in Asia, as languages like Chinese and Japanese have thousands of characters. There was no way to encode all those characters in 8-bits! In response, each country invented multi-byte encoding schemes, all of which were mutually incompatible. And so it was born – Unicode – one format to rule them all. Devised in 1992 to finally do away with all of the different international schemes it replaced them with one universal encoding scheme. The most common version of Unicode uses 16 bits with space for over a million codes - enough for every single character from every language ever used – more than 120,000 of them in over 100 types of script plus space for mathematical symbols and even graphical characters like Emoji. And in the same way that ASCII defines a scheme for encoding letters as binary numbers, other file formats – like MP3s or GIFs – use binary numbers to encode sounds or colors of a pixel in our photos, movies, and music. Most importantly, under the hood it all comes down to long sequences of bits. Text messages, video, music, every webpage on the internet,

and even your computer's operating system, are nothing but long sequences of 1s and 0s.

How Computers Calculate – the ALU.

Representing and storing numbers is an important function of a computer, but the real goal is computation, or manipulating numbers in a structured and purposeful way, like adding two numbers together. These operations are handled by a computer's Arithmetic and Logic Unit, but most people call it by its street name: the ALU. The ALU is the mathematical brain of a computer. When you understand an ALU's design and function, you'll understand a fundamental part of modern computers. First though, look at this beauty. This is perhaps the most famous ALU ever, the Intel 74181. When it was released in 1970, it was the first complete ALU that fit entirely inside of a single chip. Which was a huge engineering feat at the time.

An ALU is really two units in one -- there's an arithmetic unit and a logic unit. Let's start with the arithmetic unit, which is responsible for handling all numerical operations in a computer, like addition and subtraction. It also does a bunch of other simple things like add one to a number, which is called an increment operation, but we'll talk about those later. Today, we're going to focus on the operations that underlies almost everything else a computer does - adding two numbers together.

So we will use a high-level of abstraction and build our components out of logic gates, in this case: AND, OR, NOT and XOR gates. The simplest adding circuit that we can build takes two binary digits, and adds them together. So we have two inputs, A and B, and one output, which is the sum of those two digits.

Just to clarify: A, B and the output are all single bits. There are only four possible input combinations. The first three are: 0 + 0 = 0 1 + 0 = 1 0 + 1 = 1 Remember that in binary, 1 is the same as true, and 0 is the same as false. So this set of inputs exactly matches the boolean logic of an XOR gate, and we can use it as our 1-bit adder. But the fourth input combination, 1 + 1, is a special case. 1 + 1 is 2 (obviously) but there's no 2 digit in binary, so as we talked about last, the result is 0 and the 1 is carried to the next column. So the sum is really 10 in binary. Now, the output of our XOR gate is partially correct - 1 plus 1, outputs 0. But, we need an extra output wire for that carry bit. The carry bit is only "true" when the inputs are 1 AND 1, because that's the only time when the result (two) is bigger than 1 bit can store… and conveniently we have a gate for that! An AND gate, which is only true when both inputs are true, so we'll add that to our circuit too. And that's it. This circuit is called a half adder. It's not that complicated - just two logic gates - but let's abstract away even this level of detail and encapsulate our newly minted half adder as its own component, with two inputs - bits A and B - and two outputs, the sum and the carry bits.

Anyway, If you want to add more than 1 + 1 we're going to need a "Full Adder." A full adder is a bit more complicated – it takes three bits as inputs: A, B and C. So the maximum possible input is 1 + 1 + 1, which equals 1 carry out 1, so we still only need two output wires: sum and carry. We can build a full adder using half adders. To do this, we use a half adder to add A plus B just like before – but then feed that result and input C into a second half adder. Lastly, we need a OR gate to check if

either one of the carry bits was true. Armed with our new components, we can now build a circuit that takes two, 8-bit numbers – Let’s call them A and B – and adds them together. Let’s start with the very first bit of A and B, which we’ll call A0 and B0. At this point, there is no carry bit to deal with, because this is our first addition. So we can use our half adder to add these two bits together. The output is sum0. Now we want to add A1 and B1 together. It's possible there was a carry from the previous addition of A0 and B0, so this time we need to use a full adder that also inputs the carry bit. We output this result as sum1. Then, we take any carry from this full adder, and run it into the next full adder that handles A2 and B2. And we just keep doing this in a big chain until all 8 bits have been added. Notice how the carry bits ripple forward to each subsequent adder. For this reason, this is called an 8-bit ripple carry adder. Notice how our last full adder has a carry out. If there is a carry into the 9th bit, it means the sum of the two numbers is too large to fit into 8-bits. This is called an overflow. In general, an overflow occurs when the result of an addition is too large to be represented by the number of bits you are using. This can usually cause errors and unexpected behavior. Famously, the original PacMan arcade game used 8 bits to keep track of what level you were on. This meant that if you made it past level 255 – the largest number storablein 8 bits – to level 256, the ALU overflowed. So if we want to avoid overflows, we can extend our circuit with more full adders, allowing us to add 16 or 32 bit numbers.

The ALU’s arithmetic unit also has circuits for other math operations and in general these 8 operations are always supported. And like our adder, these other operations are built from individual logic gates. You may have noticed that there are no multiply and divide operations. That's because simple ALUs don’t have a circuit for this, and instead just perform a series of additions. Let’s say you want to multiply 12 by 5. That’s the same thing as adding 12 to itself 5 times.

Ok, let’s move on to the other half of the ALU: the Logic Unit. Instead of arithmetic operations, the Logic Unit performs logical operations, like AND, OR and NOT, which we’ve talked about previously. It also performs simple numerical tests, like checking if a number is negative. For example, here’s a circuit that tests if the output of the ALU is zero. It does this using a bunch of OR gates to see if any of the bits are 1. Even if one single bit is 1, we know the number can’t be zero and then we use a final NOT gate to flip this input so the output is 1 only if the input number is 0.

The CPU and Von Neumann Architecture.

Now we're looking at the CPU and Von Neumann Architecture, just a real introduction to it from quite a basic level. So every object in the world has its own architecture. Like human has its body architecture, car has its own architecture, even building and offices has its own architecture. Architecture defines how function works. The same way computer has its own architecture on which computer runs. There are two types of architecture. First von Neumann architecture, second Harvard architecture. Architecture also known as computer model. Let's discuss von Neumann architecture in detail. Von Neumann architecture was designed and developed by the great mathematician and scientist John von Neumann in the year 1945. Most of the computer of today's world run on the von Neumann architecture whether mainframe

or personal computer. Let's see the architecture. Input comes in stores in a memory, the input becomes the command and ready to process. The processor fetch the command from memory. There is two part in the process. First CU control unit, second ALU arithmetic logic unit. The role of the CU is to manage the commands and input/output. ALU is work to perform mathematics functions. After processing complete, the control unit stores the result in memory and display the output. This is the working of von Neumann architecture. Let's discuss components. One input. Input comes through input device like keyboard, mouse etc. Two memory. The data temporary store in the memory called RAM. Three process. Process has two parts. one control unit. The control unit works like traffic cop, it control and manage the process and input/output. The control unit fetch and store data to and from memory. Two arithmetic logic unit. The role of the ALU is to perform mathematics calculation like addition subtraction multiplication/division etc. Four output. The output display through the monitor printer etc. We will learn the working of hardware later in detail. let's understand von Neumann architecture with example. We enter 20 plus 20 through keyboard. The data is stored in the memory called RAM and ready to process. The control unit fetch the command and checks witch operation is to be para form. If its arithmetic operation the CU command to the ALU for operation. ALU you perform the addition and send back the result to the CU. Cu store the result in memory and display it through output device.

Intro to Algorithms and Data Structures.

Algorithm is the specific steps used to complete the computation. Some algorithms are better than others even if they produce equal results. Generally, the fewer steps it takes to compute, the better it is, though sometimes we care about other factors, like how much memory it uses. One of the most storied algorithmic problems in all of computer science is sorting… as in sorting names or sorting numbers. Computers sort all the time. You might think “sorting isn’t so tough… how many algorithms can there possibly be?” The answer is: a lot. Computer Scientists have spent decades inventing algorithms for sorting, with cool names like Bubble Sort and Spaghetti Sort. Let’s try sorting! Imagine we have a set of airfare prices to Minsk. We’ll talk about how data like this is represented in memory later, but for now, a series of items like this is called an array. Let’s take a look at these numbers to help see how we might sort this programmatically. We’ll start with a simple algorithm. First, let’s scan down the array to find the smallest number. Starting at the top with 307. It’s the only number we’ve seen, so it’s also the smallest. The next is 239, that’s smaller than 307, so it becomes our new smallest number. Next is 214, our new smallest number. 250 is not, neither is 384, 299, 223 or 312. So we’ve finished scanning all numbers, and 214 is the smallest. To put this into ascending order, we swap 214 with the number in the top location. Great. We sorted one number! Now we repeat the same procedure, but instead of starting at the top, we can start one spot below. First we see 239, which we save as our new smallest number. Scanning the rest of the array, we find 223 is the next smallest, so we swap this with the number in the second spot. Now we repeat again, starting from the third number down. This time, we swap 239 with 307. This process continues until we get to the very last

number, and the array is sorted! The process we just walked through is one way – or one algorithm – for sorting an array. It's called Selection Sort -- and it's pretty basic.

Here's the pseudo-code. This function can be used to sort 8, 80, or 80 million numbers - and once you've written the function, you can use it over and over again. With this sort algorithm, we loop through each position in the array, from top to bottom, and then for each of those positions, we have to loop through the array to find the smallest number to swap. You can see this in the code, where one FOR loop is nested inside of another FOR loop. This means, very roughly, that if we want to sort N items, we have to loop N times, inside of which, we loop N times, for a grand total of roughly N times N loops... Or N squared. This relationship of input size to the number of steps the algorithm takes to run characterizes the complexity of the Selection Sort algorithm. It gives you an approximation of how fast, or slow, an algorithm is going to be. Computer Scientists write this order of growth in something known as "big O notation".

N squared is not particularly efficient. Our example array had n = 8 items, and 8 squared is 64. If we increase the size of our array from 8 items to 80, the running time is now 80 squared, which is 6,400. So although our array only grew by 10 times - from 8 to 80 – the running time increased by 100 times – from 64 to 6,400! This effect magnifies as the array gets larger. That's a big problem for a company like Google, which has to sort arrays with millions or billions of entries. So, you might ask is there a more efficient sorting algorithm?

Let's go back to our old, unsorted array and try a different algorithm, merge sort. The first thing merge sort does is check if the size of the array is greater than 1. If it is, it splits the array into two halves. Since our array is size 8, it gets split into two arrays of size 4. These are still bigger than size 1, so they get split again, into arrays of size 2, and finally they split into 8 arrays with 1 item in each. Now we are ready to merge, which is how "merge sort" gets its name. Starting with the first two arrays, we read the first – and only – value in them, in this case, 307 and 239. 239 is smaller, so we take that value first. The only number left is 307, so we put that value second. We've successfully merged two arrays. We now repeat this process for the remaining pairs, putting them each in sorted order. Then the merge process repeats. Again, we take the first two arrays, and we compare the first numbers in them. This time its 239 and 214. 214 is lowest, so we take that number first. Now we look again at the first two numbers in both arrays: 239 and 250. 239 is lower, so we take that number next. Now we look at the next two numbers: 307 and 250. 250 is lower, so we take that. Finally, we're left with just 307, so that gets added last. In every case, we start with two arrays, each individually sorted, and merge them into a larger sorted array. We repeat the exact same merging process for the two remaining arrays of size two. Now we have two sorted arrays of size 4. Just as before, we merge, comparing the first two numbers in each array, and taking the lowest. We repeat this until all the numbers are merged, and then our array is fully sorted again!

The "Big O" computational complexity of merge sort is N times the Log of N. The N comes from the number of times we need to compare and merge items, which is directly proportional to the number of items in the array. The Log N comes from

the number of merge steps. In our example, we broke our array of 8 items into 4, then 2, and finally 1. That's 3 splits. Splitting in half repeatedly like this has a logarithmic relationship with the number of item! Log base 2 of 8 equals 3 splits. If we double the size of our array to 16 – that's twice as many items to sort – it only increases the number of split steps by 1 since log base 2 of 16 equals 4. Even if we increase the size of the array more than a thousand times, from 8 items to 8000 items, the number of split steps stays pretty low. Log base 2 of 8000 is roughly 13. That's more, but not much more than 3 -- about four times larger – and yet we're sorting a lot more numbers. For this reason, merge sort is much more efficient than selection sort.

Now I want to move on to one of classic algorithmic problems: graph search! A graph is a network of nodes connected by lines. You can think of it like a map, with cities and roads connecting them. Routes between these cities take different amounts of time. We can label each line with what is called a cost or weight. In this case, it's weeks of travel. Now let's say we want to find the fastest route for an army at Highgarden to reach the castle at Winterfell. The simplest approach would just be to try every single path exhaustively and calculate the total cost of each. That's a brute force approach. We could have used a brute force approach in sorting, by systematically trying every permutation of the array to check if it's sorted. This would have an N factorial complexity - that is the number of nodes, times one less, times one less than that, and so on until 1. Which is way worse than even N squared. But, we can be way more clever! The classic algorithmic solution to this graph problem was invented by one of the greatest minds in computer science practice and theory, Edsger Dijkstra, so it's appropriately named Dijkstra's algorithm. We start in Highgarden with a cost of 0, which we mark inside the node. For now, we mark all other cities with question marks - we don't know the cost of getting to them yet. Dijkstra's algorithm always starts with the node with lowest cost. In this case, it only knows about one node, Highgarden, so it starts there. It follows all paths from that node to all connecting nodes that are one step away, and records the cost to get to each of them. That completes one round of the algorithm. We haven't encountered Winterfell yet, so we loop and run Dijkstra's algorithm again. With Highgarden already checked, the next lowest cost node is King's Landing. Just as before, we follow every unvisited line to any connecting cities. The line to The Trident has a cost of 5. However, we want to keep a running cost from Highgarden, so the total cost of getting to The Trident is 8 plus 5, which is 13 weeks. Now we follow the offroad path to Riverrun, which has a high cost of 25, for a total of 33. But we can see inside of Riverrun that we've already found a path with a lower cost of just 10. So we disregard our new path, and stick with the previous, better path. We've now explored every line from King's Landing and didn't find Winterfell, so we move on. The next lowest cost node is Riverrun, at 10 weeks. First we check the path to The Trident, which has a total cost of 10 plus 2, or 12. That's slightly better than the previous path we found, which had a cost of 13, so we update the path and cost to The Trident. There is also a line from Riverrun to Pyke with a cost of 3. 10 plus 3 is 13, which beats the previous cost of 14, and so we update Pyke's path and cost as well. That's all paths from Riverrun checked... so... you guessed it, Dijkstra's

algorithm loops again. The node with the next lowest cost is The Trident and the only line from The Trident that we haven't checked is a path to Winterfell! It has a cost of 10, plus we need to add in the cost of 12 it takes to get to The Trident, for a grand total cost of 22. We check our last path, from Pyke to Winterfell, which sums to 31. Now we know the lowest total cost, and also the fastest route for the army to get there, which avoids King's Landing!

Dijkstra's original algorithm, conceived in 1956, had a complexity of the number of nodes in the graph squared O(n2). And squared, as we already discussed, is never great, because it means the algorithm can't scale to big problems. Fortunately, Dijkstra's algorithm was improved a few years later to take the number of nodes in the graph, times the log of the number of nodes, PLUS the number of lines. O(n log n + 1).

Although this looks more complicated, it's actually quite a bit faster. Plugging in our example graph, with 6 cities and 9 lines, proves it. Our algorithm drops from 36 loops to around 14. Algorithms are everywhere and the modern world would not be possible without them.

Lecture 3 Data Structures

Let's start our lecture with such interesting and important thing as data structures. Maybe you already know one basic data structure as Arrays, also called lists or Vectors in some languages. These are a series of values stored in memory. So instead of just a single value being saved into a variable, like 'j equals 5', we can define a whole series of numbers, and save that into an array variable. To be able to find a particular value in this array, we have to specify an index. Almost all programing languages start arrays at index 0, and use a square bracket syntax to denote array access. So, for example, if we want to add the values in the first and third spots of our array 'j', and save that into a variable 'a', we would write a line of code like this.

How an array is stored in memory is pretty simple. For example, let's say that the compiler chose to store ours at memory location 1,000. The array contains 7 numbers, and these are stored one after another in memory, as seen here. So when we write "j index of 0", the computer goes to memory location 1,000, with an offset of 0, and we get the value 5. If we wanted to retrieve "j index of 5", our program goes to memory location 1000, plus an offset of 5, which in this case, holds a value of 4. It's easy to confuse the fifth number in the array with the number at index 5. They are not the same. Remember, the number at index 5 is the 6th number in the array because the first number is at index 0.

It's easy to confuse the fifth number in the array with the number at index 5. They are not the same. Remember, the number at index 5 is the 6th number in the array because the first number is at index 0. Arrays are extremely versatile data structures, used all the time, and so there are many functions that can handle them to do useful things. For example, pretty much every programming language comes with

a built-in sort function, where you just pass in your array, and it comes back sorted. So there's no need to write that algorithm from scratch. Very closely related are Strings, which are just arrays of characters, like letters, numbers, punctuation and other written symbols. Most often, to save a string into memory, you just put it in quotes, like so.

Although it doesn't look like an array, it is. Behind the scenes, the memory looks like this. Note that the string ends with a zero in memory. It's not the character zero, but the binary value 0. This is called the null character, and denotes the end of the string in memory. This is important because if I call a function like "print quote", which writes the string to the screen, it prints out each character in turn starting at the first memory location, but it needs to know when to stop! The zero tells string functions when to stop. Because computers work with text so often, there are many functions that specifically handle strings. For example, many programming languages have a string concatenation function, or "strcat", which takes in two strings, and copies the second one to the end of the first. We can use arrays for making one dimensional lists, but sometimes you want to manipulate data that is two dimensional, like a grid of numbers in a spreadsheet, or the pixels on your computer screen. For this, we need a Matrix.

You can think of a Matrix as an array of arrays! So a 3 by 3 matrix is really 2 an array of size 3, with each index storing an array of size 3. We can initialize a matrix like so. In memory, this is packed together in order like this. To access a value, you need to specify two indexes, like "J index of 2, then index of 1" - this tells the computer you're looking for the item in subarray 2 at position 1. And this would give us the value 12. The cool thing about matrices is we're not limited to 3 by 3 -- we can make them any size we want -- and we can also make them any number of dimensions we want. For example, we can create a five dimensional matrix and access it like this. So far, we've been storing individual numbers or letters into our arrays or matrices. But often it's useful to store a block of related variables together.

Like, you might want to store a bank account number along with its balance. Groups of variables like these can be bundled together into a Struct. Now we can create variables that aren't just single numbers, but are compound data structures, able to store several pieces of data at once. We can even make arrays of structs that we define, which are automatically bundled together in memory. If we access, for example, J index of 0, we get back the whole struct stored there, and we can pull the specific account number and balance data we want. This array of structs, like any other array, gets created at a fixed size that can't be enlarged to add more items. Also, arrays must be stored in order in memory, making it hard to add a new item to the middle. But, the struct data structure can be used for building more complicated data structures that avoid these restrictions.

Let's take a look at this struct that's called a "node". It stores a variable, like a number, and also a pointer. A pointer is a special variable that points, hence the name, to a location in memory. Using this struct, we can create a linked list, which is a flexible data structure that can store many nodes. It does this by having each node point to the next node in the list. Let's imagine we have three node structs saved in

memory, at locations 1000, 1002 and 1008. They might be spaced apart, because they were created at different times, and other data can sit between them. So, you see that the first node contains the value 7, and the location 1008 in its “next” pointer. This means that the next node in the linked list is located at memory location 1008. Looking down the linked list, to the next node, we see it stores the value 112 and points to another node at location 1002. If we follow that, we find a node that contains the value 14 and points back to the first node at location 1000.

So this linked list happened to be circular, but it could also have been terminated by using a next pointer value of 0 -- the null value -- which would indicate we’ve reached the end of the list. When programmers use linked lists, they rarely look at the memory values stored in the next pointers. Instead, they can use an abstraction of a linked list, that looks like this, which is much easier to conceptualize. Unlike an array, whose size has to be pre-defined, linked lists can be dynamically extended or shortened. For example, we can allocate a new node in memory, and insert it into this list, just by changing the next pointers. Linked Lists can also easily be re-ordered, trimmed, split, reversed, and so on. Owing to this flexibility, many more-complex data structures are built on top of linked lists The most famous and universal are queues and stacks.

A queue – like the line at your post office – goes in order of arrival. The person who has been waiting the longest, gets served first. This behavior is called First-In First-Out, or FIFO. Imagine we have a pointer, named “post office queue”, that points to the first node in our linked list. Once we’re done serving Hank, we can read Hank’s next pointer, and update our “post office queue” pointer to the next person in the line. We’ve successfully dequeued Hank -- he’s gone, done, finished. If we want to enqueue someone, that is, add them to the line, we have to traverse down the linked list until we hit the end, and then change that next pointer to point to the new person.

With just a small change, we can use linked lists as stacks, which are LIFO… Last-In First-Out. You can think of this like a stack of pancakes... as you make them, you add them to the top of stack. And when you want to eat one, you take them from the top of the stack. Instead of enqueueing and dequeuing, data is pushed onto the stack and popped from the stacks.

If we update our node struct to contain not just one, but two pointers, we can build trees, another data structure that’s used in many algorithms. Again, programmers rarely look at the values of these pointers, and instead conceptualize trees like this: The top most node is called the root. And any nodes that hang from other nodes are called children nodes. As you might expect, nodes above children are called parent nodes. And finally, any nodes that have no children -- where the tree ends -- are called Leaf Nodes. In our example, nodes can have up to two children, and for that reason, this particular data structure is called a binary tree. But you could just as easily have trees with three, four or any number of children by modifying the data structure accordingly. You can even have tree nodes that use linked lists to store all the nodes they point to. An important property of trees – both in real life and in data structures – is that there’s a one-way path from roots to leaves.

Software Engineering.

So I think you know a lot about sorting and often code to sort a list of numbers might only be ten lines long, which is easy enough for a single programmer to write. Plus, it's short enough that you don't need any special tools – you could do it in Notepad. But, a sorting algorithm isn't a program; it's likely only a small part of a much larger program. For example, Microsoft Office has roughly 40 millions lines of code. That's way too big for any one person to figure out and write!

To build huge programs like this, programmers use a set of tools and practices. Breaking big programs into smaller functions allows many people to work simultaneously. They don't have to worry about the whole thing, just the function they're working on. So, if you're tasked with writing a sort algorithm, you only need to make sure it sorts properly and efficiently. However, even packing code up into functions isn't enough. Microsoft Office probably contains hundreds of thousands of them. That's better than dealing with 40 million lines of code, but it's still way too many "things" for one person or team to manage. The solution is to package functions into hierarchies, pulling related code together into "objects".

For example, car's software might have several functions related to cruise control, like setting speed, nudging speed up or down, and stopping cruise control altogether. Since they're all related, we can wrap them up into a unified cruise control object. But, we don't have to stop there, cruise control is just one part of the engine's software. There might also be sets of functions that control spark plug ignition, fuel pumps, and the radiator. So we can create a "parent" Engine Object that contains all of these "children" objects. In addition to children *objects*, the engine itself might have its *own* functions. You want to be able to stop and start it, for example. It'll also have its own variables, like how many miles the car has traveled. In general, objects can contain other objects, functions and variables. And of course, the engine is just one part of a Car Object. There's also the transmission, wheels, doors, windows, and so on. Now, as a programmer, if I want to set the cruise control, I navigate down the object hierarchy, from the outermost objects to more and more deeply nested ones. Eventually, I reach the function I want to trigger: "Car, then engine, then cruise control, then set cruise speed to 55".

Programming languages often use something equivalent to the syntax shown here. The idea of packing up functional units into nested objects is called Object Oriented Programming. This is very similar to what we've done all series long: hide complexity by encapsulating low-level details in higher-order components.

Breaking up a big program, like a car's software, into functional units is perfect for teams. One team might be responsible for the cruise control system, and a single programmer on that team tackles a handful of functions. This is similar to how big, physical things are built, like skyscrapers. You'll have electricians running wires, plumbers fitting pipes, welders welding, painters painting, and hundreds of other people teeming all over the hull.

They work together on different parts simultaneously, leveraging their different skills. Until one day, you've got a whole working building! But, returning to our cruise control example… its code is going to have to make use of functions in other

parts of the engine's software, to keep the car at a constant speed. That code isn't part of the cruise control team's responsibility. It's another team's code. Because the cruise control team didn't write that, they're going to need good documentation about what each function in the code does, and a well-defined Application Programming Interface -- or API for short.

You can think of an API as the way that collaborating programmers interact across various parts of the code. For example, in the IgnitionControl object, there might be functions to set the RPM of the engine, check the spark plug voltage, as well as fire the individual spark plugs. Being able to set the motor's RPM is really useful, the cruise control team is going to need to call that function. But, they don't know much about how the ignition system works. It's not a good idea to let them call functions that fire the individual spark plugs. Or the engine might explode! Maybe. The API allows the right people access to the right functions and data. Object Oriented Programming languages do this by letting you specify whether functions are public or private. If a function is marked as "private", it means only functions inside that object can call it. So, in this example, only other functions inside of IgnitionControl, like the setRPM function, can fire the sparkplugs. On the other hand, because the setRPM function is marked as public, other objects can call it, like cruise control. This ability to hide complexity, and selectively reveal it, is the essence of Object Oriented Programming, and it's a powerful and popular way to tackle building large and complex programs. Pretty much every piece of software on your computer, or game running on your console, was built using an Object Oriented Programming Language, like C++, C# or Objective-C. Other popular "OO" languages you may have heard of are Python and Java.

It's important to remember that code, before being compiled, is just text. As I mentioned earlier, you could write code in Notepad or any old word processor. Some people do. But generally, today's software developers use special-purpose applications for writing programs, ones that integrate many useful tools for writing, organizing, compiling and testing code. Because they put everything you need in one place, they're called Integrated Development Environments, or IDEs for short. All IDEs provide a text editor for writing code, often with useful features like automatic color-coding to improve readability. Many even check for syntax errors as you type, like spell check for code. Big programs contain lots of individual source files, so IDEs allow programmers to organize and efficiently navigate everything. Also built right into the IDE is the ability to compile and run code. And if your program crashes, because it's still a work in progress, the IDE can take you back to the line of code where it happened, and often provide additional information to help you track down and fix the bug, which is a process called debugging. This is important because most programmers spend 70 to 80% of their time testing and debugging, not writing new code. Good tools, contained in IDEs, can go a long way when it comes to helping programmers prevent and find errors.

In addition to coding and debugging, another important part of a programmer's job is documenting their code. This can be done in standalone files called "read-me's" which tell other programmers to read that help file before diving in. It can also

happen right in the code itself with comments. These are specially-marked statements that the program knows to ignore when the code is compiled. They exist only to help programmers figure out what's what in the source code. Good documentation helps programmers when they revisit code they haven't seen for a while, but it's also crucial for programmers who are totally new to it. Documentation also promotes code reuse. So, instead of having programmers constantly write the same things over and over, they can track down someone else's code that does what they need. Then, thanks to documentation, they can put it to work in their program, without ever having to read through the code. "Read the docs" as they say.

In addition to IDEs, another important piece of software that helps big teams work collaboratively on big coding projects is called Source Control, also known as version control or revision control. Most often, at a big software company like Apple or Microsoft, code for projects is stored on centralized servers, called a code repository. When a programmer wants to work on a piece of code, they can check it out, sort of like checking out a book out from a library. Often, this can be done right in an IDE. Then, they can edit this code all they want on their personal computer, adding new features and testing if they work. When the programmer is confident their changes are working and there are no loose ends, they can check the code back into the repository, known as committing code, for everyone else to use. While a piece of code is checked out, and presumably getting updated or modified, other programmers leave it alone. This prevents weird conflicts and duplicated work. In this way, hundreds of programmers can be simultaneously checking in and out pieces of code, iteratively building up huge systems. Critically, you don't want someone committing buggy code, because other people and teams may rely on it. Their code could crash, creating confusion and lost time. The master version of the code, stored on the server, should always compile without errors and run with minimal bugs. But sometimes bugs creep in. Fortunately, source control software keeps track of all changes, and if a bug is found, the whole code, or just a piece, can be rolled back to an earlier, stable version. It also keeps track of who made each change, so coworkers can send nasty, I mean, helpful and encouraging emails to the offending person. Debugging goes hand in hand with writing code, and it's most often done by an individual or small team.

The big picture version of debugging is Quality Assurance testing, or QA. This is where a team rigorously tests out a piece of software, attempting to create unforeseen conditions that might trip it up. Basically, they elicit bugs. Getting all the wrinkles out is a huge effort, but vital in making sure the software works as intended for as many users in as many situations as imaginable before it ships. You've probably heard of beta software This is a version of software that's mostly complete, but not 100% fully tested. Companies will sometimes release beta versions to the public to help them identify issues, it's essentially like getting a free QA team. What you don't hear about as much is the version that comes before the beta: the alpha version. This is usually so rough and buggy, it's only tested internally. So, that's the tip of the iceberg in terms of the tools, tricks and techniques that allow software engineers to construct the huge pieces of software that we know and love today, like YouTube, Grand Theft Auto 5, and Powerpoint.

Operating Systems

The operating system allows us to control the hardware of our computer. So our hard drives, and our memory, and our CPU you can all communicate back and forth to each other using this operating system. The operating system is also where all of your applications will run. So anything that you're doing in a spreadsheet, in a word processing document, or in a browser is all going to execute inside of your operating system. This operating system is also a way for us as humans to be able to interoperate with this technology. The operating system may be at a command line where you type everything in using your keyboard. Or it may be a graphical user interface where you use icons and a mouse to be able to use this technology. There's some common features that you'll find in any operating system that you happen to use. One of these features is file management. You're able to store a document or a spreadsheet onto the storage capabilities of that operating system. And then you can delete that file, rename that file, or manage where that data happens to go. An operating system is also going to provide a way to run applications. This means not only is it going to display that application on the screen, there's going to be a lot of interaction between that application and the memory that you're using. Or there may be files that are spelled out to disk using swap file management. And then swap back in as you're using other parts of that application. As you're using this application, you're storing information on hard drives. Perhaps transferring information to a USB. There's going to be a keyboard and mouse that you're using. And all of this hardware and software works together by using the interactions of that operating system. And there needs to be some way to manage the overall operation of the OS. The operating system will usually include programs or utilities that allow you as the end user to be able to monitor and manage any aspect of that OS.

One of the most popular operating systems is Microsoft Windows. Not only are there the traditional desktop operating systems for Windows such as Windows 7 and Windows 10, they're also server versions of Microsoft Windows that are used to manage very large scale applications in the data center. The advantages of Microsoft Windows is that it has a huge support from the industry. Most applications are automatically written to run in Microsoft Windows. You also have many different options for the type of Windows that you're using. There's versions of Windows that are designed to be use at home and other versions of Windows that are designed to be used in the data center for large scale applications. Unfortunately, having such a popular operating system with such a large number of installed systems means that you're also a very big Target for security exploitation. The bad guys know that if they can find one way into Windows, then they're able to take advantage of many millions of desktops around the world. Another challenge with Windows is that it has to support a wide variety of hardware. So many different monitors, many different video cards, different storage devices, and different types of printers all have to interoperate and work properly in this single operating system. Here's a screenshot of Windows 10. You can see the menu along the left side. You've got a search bar at the bottom, and other icons that you can choose here to quickly access the applications that you need.

Another popular operating system is Apple's macOS. This is primarily designed as a desktop operating system designed to run on Apple hardware. Apple's macOS is designed for ease of use. And because you're running on Apple's hardware, you know that the operating system and the hardware of that computer will be extremely compatible with each other. This also means you have less support in the industry for applications running on that operating system than something like Microsoft Windows. And with Microsoft Windows, you can choose the components you'd like so that you can build a computer to fit any particular price point. With macOS the only hardware options you have are those available from Apple. So you may find that your initial hardware cost is a little bit more than if you were to build the system yourself. And here's a macOS desktop. And you can see there are a lot of similarities between Mac OS and Windows. There's icons at the bottom that can be used to start applications. You can see the storage devices are listed as icons on the desktop. And in many ways, the similarities are the same between Windows, macOS, and the Linux operating systems.

Another popular operating system is the free operating system of Linux. With Linux, there is no single Linux operating system. Instead there is a combination of features put together into what the Linux community calls distributions. One of the obvious advantages of Linux is the cost. There is no cost associated for using or running any applications on a Linux operating system. Linux also works on a wide variety of hardware so you can build your own computer and run the Linux operating system on top of it. And the user community for Linux is very large and worldwide. So if you have any questions about the Linux operating system, there will always be someone willing to help. Because Linux is an operating system that relies on this user community though, there are some limitations to the amount of hardware that can be supported. You may find that there is limited driver support for hardware that is not mainstream. Especially for customized or proprietary hardware that might be inside of laptop computers. You might also find that support options for Linux are also limited. There's no single Linux company to go to for support. So you have to make sure you have knowledgeable people available to help you with any problems that you might have with the Linux operating system. Here's the desktop of an Ubuntu Linux distribution. And as you can see, it has very similar functionality to Microsoft Windows or macOS. You have the standard icons on the screen that allow you to run applications. Or a search bar at the top so that you can find your documents or other applications all with a few keystrokes.

One of the distinguishing characteristics of an operating system is the type of processor that is supported by that OS. There are usually two options to choose from. A 32-bit processor or a 64-bit processor. One of the differences between these two is the amount of information that can be processed or stored by that processor. For example, a 32-bit processor can store values that are 2 to the 32nd power, which means you get about four billion different values. If you've ever seen the amount of available memory that you can put in a 32-bit operating system, you know that it's about 4 gigabytes of memory. And so you can see that's exactly where that four gigabyte value comes from. A 64-bit processor can store information that is 2 to the

64th power, which is much larger than our 4 gigabit value with a 32-bit operating system. In fact, in a 64-bit operating system, it's about 17 billion gigabytes of information that can be accessed, which is very different than the 4 gigabytes available in a 32-bit operating system. This means that theoretically, your operating system running on a 64-bit processor could allow you to install 17 billion gigabytes of memory. In reality though, the operating system probably has a much smaller maximum value. Although certainly much larger than the capabilities you would have with a 32-bit operating system. Another important consideration when you're using 32-bit versus 64-bit operating systems are the type of drivers that you're using in the operating system itself. The driver is special software that allows your operating system to be able to communicate with the hardware of your computer. If you're running a 32-bit operating system, you have to use 32-bit drivers. If you're using a 64-bit operating system, you have to use drivers that are specifically written for that 64-bit OS. You cannot use 32-bit drivers on a 64-bit operating system, or vise versa. You'll sometimes see the 32-bit architecture abbreviated as x86. If you see 64-bit architectures abbreviated, it's often abbreviated as x64. Another important consideration between a 32-bit operating system and a 64-bit operating system are the applications that you'll run on that OS. You'll find that there are some applications that have been written for 32-bit operating systems. And other applications that have been written for 64-bit operating systems. If you're running a 32-bit operating system, you will not be able to run 64-bit applications on it. But if you're running a 64-bit operating system, you can run either 32-bit apps or 64-bit apps. For example, if you're running a 64-bit version of Windows, you'll find that any installed 32-bit applications will be stored in the Program Files x86 directory. And any 64-bit applications will be installed in the Program Files directory.

Our mobile computing devices are getting smaller and smaller all the time. But we want the same functionality on our tablet devices as we have on our desktop systems that we're using at work. That's why you'll find that there are tablet devices these days that run a full blown version of Windows 10 on these mobile tablet devices. These tablets are made by many different manufacturers. But they tend to have the same type of hardware with touch screens detachable keyboards and perhaps a stylus that can be used in the Windows 10 operating system. Early versions of these tablets used a type of operating system that wasn't a full blown version of Windows 10. In fact, it was called Windows Mobile. Windows Mobile is no longer in development. And after December 2019, it will no longer be supported by Microsoft.

On the handset and mobile phone side, one of the operating systems you'll commonly see is the Android operating system from Google. This is actually the Open Handset Alliance. And it runs an open source Linux version of an operating system on these small mobile devices. There are many different manufacturers that make Google Android compatible systems so you're able to find a mobile phone with exactly the features you want. If you're an application developer, you can create apps for the Android operating system inside of Windows, macOS, and Linux by using the Android Software Developers Kit. If you're an end user and you want to install these

apps, you can get them from Google Play or from third party sites such as the Amazon App Store.

Apple phones use their own operating system called iOS. This is available on the Apple iPhone and the Apple iPad. It's based on Unix, and it's an operating system that is closed source. You don't have access to any of the source code used in iOS. You'll also only find iOS on Apple products. You're not able to buy hardware from a third party and run iOS on it. You can only run iOS on Apple hardware. If you're a software developer and you want to create apps for an Apple device, then you'll want to use macOS as your operating system. And use the iOS Software Developers Kit. To make that software available to the end user, you submit it to Apple. And they must approve your software to be able to have that available for end users. Once they approve it, it's added to the Apple App Store. And users are able to download and use it on their iOS device.

Google also has their own operating system. This is Chrome OS. And it's based on the Linux kernel. This operating system focuses around the Chrome web browser. Most of the apps that you'll use on Chrome OS will all run from inside of that browser. This relatively simple approach to applications means that many different manufacturers can now create hardware that will run Chrome OS. And since the operating system is less demanding, you also will find that the hardware is less expensive. Because many of the apps and Chrome OS run in this browser, there's a strong reliance on the cloud. So you have to make sure you have a good internet connection to run many of the apps available in Chrome OS. With so many different operating systems to choose from, it's important to know where the limitations are with each particular manufacturer's OS. For example, you have to think about how long a particular operating system will be supported. Different manufacturers have different philosophies around end of life. Some manufacturers like to support an operating system for a very long time. Others support it for a shorter period. And expect that you're going to upgrade to new hardware when that software hits end of life. Manufacturers also have a different philosophy on how the operating system itself will be upgraded. With Microsoft Windows, macOS, and Linux you may be presented with an upgrade option. And then you get to choose. Although even those options are slowly migrating into operating systems that will always keep themselves upgraded such as the philosophy found with Chrome OS. Even though these are very different operating systems there are certain types of data that can be shared between the different OS. For example, it's very common for movies and music to use a common format that you can watch or listen to regardless of what operating system you may be using. Where you start to run into problems though is with the applications themselves. If an application is designed for Windows, it usually is only going to run in Windows. In some cases you're able to take the data files that you've saved from that application and move the data file to a different application on a different operating system. And that may be a big reason why some web based applications have such popularity since they'll run in any web browser regardless of the underlying operating system.

Lecture 4 Computer Memory, Processing, and Storage

This lecture is about basic computer organization, including storage and processing. We will look at the basic organizational scheme for computers and go into some detail about each of the major components. Modern computers are quite complex machines, but they can be understood with a few fundamentals and a lot of layering. Layers are essential to any interaction with computers. While using a computer doesn't necessarily require that you know what is going on under the hood, if you peel back the layers and look at what makes a computer work, you are empowered to make better decisions about how to use your computer, from making choices about what to download and what to avoid, to deciding on a particular memory kit for your computer. Let's start with some basic ideas and work up to a fuller understanding.

Every computer has a clock, which is a piece of hardware (an oscillator) that sends pulses of electrical current to all components of the computer at the same time. This clock is measured in Hertz, or pulses per second. The time between two such pulses in a computer clock is a clock cycle. This gives us a unit, with which to measure how quickly or slowly things happen within a computer's hardware. Usually, we are concerned with instructions, or operations to be performed at the hardware level. "Add 10 to 5" is an instruction, for example, and this instruction can be completed within a certain number of clock cycles, depending on the particular computer. Generally speaking, the higher the frequency of oscillations, the more quickly instructions can be carried out, and therefore, the faster the computer as a whole. However, there are other factors that contribute to increased or decreased computational speed.

Now that we have a mechanism for moving current throughout the hardware components of a computer, we need a few more things. Most modern computers follow the same organizational structure, called the von Neumann Model, invented by John W. Mauchly and J. Presper Eckert, and not by John von Neumann, as is commonly believed. The von Neumann model consists of a processor, registers, an arithmetic logic unit, a control unit, input/output capability, and storage media for both short-term memory and long-term storage. We will look at each of these systems in this lesson.

A computer's CPU, or central processing unit, is the real brains of the operation. It is the hardware component that is responsible for fetching data, carrying out instructions, and delegating tasks to other hardware components. Inside a CPU are several components that make it work: registers, an arithmetic logic unit, or ALU, and a control unit.

Registers are memory locations within the CPU that the CPU can access most readily. Because registers are housed within the CPU, access time is exceptionally low. Access time increases as components are farther from the CPU. Registers can only store so much data, however, as they are usually quite small in capacity. An average CPU register holds no more than 64 bits and there are no more than 64

registers in total. Registers are therefore usually set aside for special purposes. Low-level assembly languages can access theses registers directly, using an identifier like “r0” for register 0 or “r1” for register 1. Registers, like other types of memory we’ll see in a moment, are volatile. They can only hold information, so long as the power is turned on and current is flowing. The moment the power is switched off, the information inside the registers is lost.

Also inside the CPU is the Arithmetic Logic Unit, or ALU. The ALU performs operations on bits using transistors, gates, and small memory stores. Without going into the details of how this works, the ALU is given an input of commands and data and returns an output of data that is the result of executing the commands on the input data. The control unit in the CPU is the device that tells other components what to do and how to respond to data. The control unit can, for example, instruct data from a register to be processed in the ALU and then returned to another register.

A machine cycle, also known as a von Neumann execution cycle or a fetch-decode-execute cycle, outlines the steps a computer’s hardware follows to carry out instructions and run programs using those instructions. The cycle is a 4-step process, begun by fetching instructions from memory. The instructions are decoded in the control unit as commands to the ALU, then sent to the ALU for execution of the commands. The results are then stored in memory. This simple process happens over and over again within a computer and is what makes the computer run. Because computers can move through the steps of this process billions of times per second, computational power is achieved. The fetch-decode-execute cycle model does a good job of explaining what a computer is doing when it is computing.

Understanding this cycle requires some abstraction – it requires that you ignore the specific details of how the computer fetches, decodes, and executes - but can still explain what the computer is doing. Abstraction is everywhere in computers and actually helps us understand why modern computers are so user-friendly. To browse the internet, for example, the computer user needn’t worry about the software the runs the browser, the way the browser’s program is converted into lower-level languages and eventually converted into digital logic in the hardware. At each level of abstraction, the details at lower levels are hidden and irrelevant.

More formally, there are seven levels of abstraction that computer scientists generally agree on. At the very top are users, who deal with executable programs. Moving down the levels of abstraction, there are high-level languages (Python is one example of a high-level language that we’ll look at in other lessons), assembly languages, which deal with assembly code, system software (the programs that run your operating system and other system-level programs), instruction set architecture (the methods and rules that software and hardware use to communicate with each other, microcode, which is essentially the hardware implementation of low-level machine code, and digital logic, which includes the logic gates and circuits used at the lowest level to move, manipulate, and store bits.

The interface between software and hardware becomes quite important when input and output devices come into play. Software needs to be built to handle key presses on a keyboard or clicks on a mouse, for example, which are registered in

hardware as electrical signals. Displays such as monitors are likewise hardware devices that are supported by software. An input/output system to handle these and other devices is crucial to the function of a computer if it is to have any human users. With an understanding of how the CPU delegates and carries out instructions, let's turn our attention to memory and storage.

Memory solves the problem of holding onto data, so that it can be used for computations. Luckily, at the user level, you don't need to worry too much about memory. Programmers do have to worry a bit more about memory, however. When somebody mentions memory in the context of computers, RAM is usually what is meant. RAM is primary or main memory and stands for random access memory. The name refers to the ability for cells within RAM to be accessed randomly using a system of addresses. RAM holds a certain number of memory cells of a given size. Each cell is given a unique address. Address systems generally start the lowest memory address at zero and move upward – 1, 2, 3, and so on. RAM uses transistors paired with capacitors, which lose charge over time. RAM is therefore volatile, just like registers in the CPU. The bits in this type of RAM, called dynamic RAM or DRAM, need to be constantly erased and rewritten. Static RAM, or SRAM, on the other hand, does not need constant refreshing, but uses more transistors per bit stored. There are sub-types of each of these, but in general, DRAM is good for desktop computers, while SRAM is better suited for embedded systems, like the much smaller computers that run calculators or toys.

All RAM is good only for short-term storage, as any stored information is lost when the power goes out. Luckily, RAM makes up for this limitation in speed – it's not quite as fast as the registers in the CPU, but faster than a hard drive. All the information needed for a program that is currently running is ideally all located in RAM, though this is generally not the case in practice, as RAM can't hold unlimited data. Often, whenever a computer is processing data, it moves needed bits to RAM, but actually moves more data than the program initially requires. This is because of the locality of reference principle. It assumes that neighboring memory addresses are likely to be used in sequence and within a short period of time. Pre-loading data expected to be used results in saved processing time.

Let's take a step back. The systems we've just seen are the CPU, with its registers, control unit, and ALU, input/output devices, and primary memory. These systems are all connected via busses. In modern systems, there are three busses that connect them, each with a specific purpose: one for data, one for addresses, and one for control signals.

What's left for us to discuss is a long-term storage medium, a hard drive. Hard drives, or hard disk drives, are long-term, non-volatile, or persistent, storage media, that is, they store programs and data for long-term use and do not lose the information when the power is turned off. The access time for data on a hard drive is called seek time and is much slower than the access time for primary memory and registers. Hard disk drives use stacks of spinning disks called platters, each with billions of bits stored by magnetism. A bit is either a 0 or 1, depending on whether or not it is magnetized. A read/write head scans over the platters and converts bits to

electrical signals to write bits to the platters and converts magnetized states into bits to read from the platters.

Solid state drives, on the other hand, do not have any mechanical parts. Instead, they use integrated circuits and transistors designed in such a way, that they do not require constant refreshing like RAM does. Now, we've look at just the basics of computer organization, but you should have a fuller picture of how a computer works from this lesson. You should be able to recognize that parts of a computer do not work as individual units, but are really connected in meaningful ways to produce a dynamic machine.

Files & File Systems.

You've no doubt encountered many types of files, like text files, music files, photos and videos. Now, we're going to talk about how files work, and how computers keep them all organized with File Systems. It's perfectly legal for a file to contain arbitrary, unformatted data, but it's most useful and practical if the data inside the file is organized somehow. This is called a file format. You can invent your own, and programmers do that from time to time, but it's usually best and easiest to use an existing standard, like JPEG and MP3.

Let's look at some simple file formats. The most straightforward are T-X-T files, which contain text. Like all computer files, this is just a huge list of numbers, stored as binary. If we look at the raw values of a T-X-T file in storage, it would look something like this: We can view this as decimal numbers instead of binary, but that still doesn't help us read the text. The key to interpreting this data is knowing that T-X-T files use ASCII, a character encoding standard. So, in ASCII, our first value, 72, maps to the capital letter H. And in this way, we decode the whole file.

Let's look at a more complicated example: a WAVE File – also called a WAV – which stores audio. Before we can correctly read the data, we need to know some information, like the bit rate and whether it's a single track or stereo. Data, about data, is called meta data. This metadata is stored at the front of the file, ahead of any actual data, in what's known as a Header. Here's what the first 44 bytes of a WAV file looks like. Some parts are always the same, like where it spells out W-A-V-E. Other parts contain numbers that change depending on the data contained within. The audio data comes right behind the metadata, and it's stored as a long list of numbers. These values represent the amplitude of sound captured many times per second.

As an example, let's look at a waveform of me saying: "hello!" Now that we've captured some sound, let's zoom into a little snippet. A digital microphone, like the one in your computer or smartphone, samples the sound pressure thousands of times. Each sample can be represented as a number. Larger numbers mean higher sound pressure, what's called amplitude. And these numbers are exactly what gets stored in a WAVE file! Thousands of amplitudes for every single second of audio! When it's time to play this file, an audio program needs to actuate the computer's speakers such that the original waveform is emitted.

So, now that you're getting the hang of file formats, let's talk about bitmaps or B-M-Ps, which store pictures. On a computer, PICtures are made up of little tiny square ELements called pixels. Each pixel is a combination of three colors: red, green

and blue. These are called additive primary colors, and they can be mixed together to create any other color on our electronic displays. Now, just like WAV files, BMPs start with metadata, including key values like image width, image height, and color depth.

As an example, let's say the metadata specified an image 4 pixels wide, by 4 pixels tall, with a 24-bit color depth - that's 8-bits for red, 8-bits for green, and 8-bits for blue. As a reminder, 8 bits is the same as one byte. The smallest number a byte can store is 0, and the largest is 255. Our image data is going to look something like this: Let's look at the color of our first pixel. It has 255 for its red value, 255 for green and 255 for blue. This equates to full intensity red, full intensity green and full intensity blue. These colors blend together on your computer monitor to become white. So our first pixel is white! The next pixel has a Red-Green-Blue, or RGB value of 255, 255, 0. That's the color yellow! The pixel after that has a RGB value of 0,0,0 - that's zero intensity everything, which is black. And the next one is yellow. Because the metadata specified this was a 4 by 4 image, we know that we've reached the end of our first row of pixels. So, we need to drop down a row. The next RGB value is 255,255,0 – yellow again. Okay, let's go ahead and read all the pixels in our 4x4 image. Obviously this is a simple example of a small image, but we could just as easily store this image in a BMP.

I want to emphasize again that it doesn't matter if it's a text file, WAV, BMP, or fancier formats we don't have time to discuss, like ZIPs and PPTs. Under the hood, they're all the same: long lists of numbers, stored as binary, on a storage device. File formats are the key to reading and understanding the data inside. Now that you understand files a little better, let's move on to how computers go about storing them.

Even though the underlying storage medium might be a strip of tape, a drum, a disk, or integrated circuits... hardware and software abstractions let us think of storage as a long line of little buckets that store values. In the early days, when computers only performed one computation like calculating artillery range tables – the entire storage operated like one big file. Data started at the beginning of storage, and then filled it up in order as output was produced, up to the storage capacity. However, as computational power and storage capacity improved, it became possible, and useful, to store more than one file at a time. The simplest option is to store files back-to-back. This can work... but how does the computer know where files begin and end?

Storage devices have no notion of files – they're just a mechanism for storing lots of bits. So, for this to work, we need to have a special file that records where other ones are located. This goes by many names, but a good general term is Directory File. Most often, it's kept right at the front of storage, so we always know where to access it. Location zero! Inside the Directory File are the names of all the other files in storage. In our example, they each have a name, followed by a period, and end with what's called a File Extension, like "BMP" or "WAV". Those further assist programs in identifying file types. The Directory File also stores metadata about these files, like when they were created and last modified, who the owner is,

and if it can be read, written or both. But most importantly, the directory file contains where these files begin in storage, and how long they are.

If we want to add a file, remove a file, change a filename, or similar, we have to update the information in the Directory File. It's like the Table of Contents in a book, if you make a chapter shorter, or move it somewhere else, you have to update the table of contents, otherwise the page numbers won't match! The Directory File, and the maintenance of it, is an example of a very basic File System, the part of an Operating System that manages and keep track of stored files. This particular example is a called a Flat File System, because they're all stored at one level.

Of course, packing files together, back-to-back, is a bit of a problem, because if we want to add some data to let's say "todo.txt", there's no room to do it without overwriting part of "carrie.bmp". So modern File Systems do two things. First, they store files in blocks. This leaves a little extra space for changes, called slack space. It also means that all file data is aligned to a common size, which simplifies management. In a scheme like this, our Directory File needs to keep track of what block each one is stored in.

The second thing File Systems do, is allow files to be broken up into chunks and stored across many blocks. So let's say we open "todo.txt", and we add a few more items then the file becomes too big to be saved in its one block. We don't want to overwrite the neighboring one, so instead, the File System allocates an unused block, which can accommodate extra data. With a File System scheme like this, the Directory File needs to store not just one block per file, but rather a list of blocks per file. In this way, we can have files of variable sizes that can be easily expanded and shrunk, simply by allocating and deallocating blocks.

Now let's say we want to delete "carrie.bmp". To do that, we can simply remove the entry from the Directory File. This, in turn, causes one block to become free. Note that we didn't actually erase the file's data in storage, we just deleted the record of it. At some point, that block will be overwritten with new data, but until then, it just sits there. This is one way that computer forensic teams can "recover" data from computers even though people think it has been deleted.

Ok, let's say we add even more items to our todo list, which causes the File System to allocate yet another block to the file, in this case, recycling the block freed from carrie.bmp. Now our "todo.txt" is stored across 3 blocks, spaced apart, and also out of order. Files getting broken up across storage like this is called fragmentation. It's the inevitable byproduct of files being created, deleted and modified. For many storage technologies, this is bad news. On magnetic tape, reading todo.txt into memory would require seeking to block 1, then fast forwarding to block 5, and then rewinding to block 3 – that's a lot of back and forth! In real world File Systems, large files might be stored across hundreds of blocks, and you don't want to have to wait five minutes for your files to open. The answer is defragmentation!

That might sound like technobabble, but the process is really simple, and once upon a time it was really fun to watch! The computer copies around data so that files have blocks located together in storage and in the right order. After we've defragged, we can read our todo file, now located in blocks 1 through 3, in a single, quick read

pass. So far, we've only been talking about Flat File Systems, where they're all stored in one directory. This worked ok when computers only had a little bit of storage, and you might only have a dozen or so files. But as storage capacity [kəˈpasitē] exploded so did the number of files on computers. Very quickly, it became impractical to store all files together at one level.

Just like documents in the real world, it's handy to store related files together in folders. Then we can put connected folders into folders, and so on. This is a Hierarchical [ˌhī(ə)ˈrärkikəl] File System, and its what your computer uses. There are a variety of ways to implement this, but let's stick with the File System example we've been using to convey the main idea. The biggest change is that our Directory File needs to be able to point not just to files, but also other directories.

To keep track of what's a file and what's a directory, we need some extra metadata. This Directory File is the top-most one, known as the Root Directory. All other files and folders lie beneath this directory along various file paths. We can see inside of our "Root" Directory File that we have 3 files and 2 subdirectories: music and photos. If we want to see what's stored in our music directory, we have to go to that block and read the Directory File located there; the format is the same as our root directory. There's a lot of great songs in there! In addition to being able to create hierarchies of unlimited depth, this method also allows us to easily move around files.

So, if we wanted to move "theme.wav" from our root directory to the music directory, we don't have to re-arrange any blocks of data. We can simply modify the two Directory Files, removing an entry from one and adding it to another. Importantly, the theme.wav file stays in block 5. So that's a quick overview of the key principles of File Systems. They provide yet another way to move up a new level of abstraction. File systems allow us to hide the raw bits stored on magnetic tape, spinning disks and the like, and they let us think of data as neatly organized and easily accessible files.

Compression

We have discussed some basic file formats, like text, wave, and bitmap. While these formats are perfectly fine and still used today, their simplicity also means they're not very efficient. Ideally, we want files to be as small as possible, so we can store lots of them without filling up our hard drives, and also transmit them more quickly. The answer is compression, which literally squeezes data into a smaller size. To do this, we have to encode data using fewer bits than the original representation.

This image is 4 pixels by 4 pixels. As we discussed, image data is typically stored as a list of pixel values. To know where rows end, image files have metadata, which defines properties like dimensions. But, to keep it simple today, we're not going to worry about it. Each pixel's color is a combination of three additive primary colors: red, green and blue. We store each of those values in one byte, giving us a range of 0 to 255 for each color. If you mix full intensity red, green and blue - that's 255 for all three values – you get the color white. If you mix full intensity red and green, but no blue (it's 0), you get yellow.

We have 16 pixels in our image, and each of those needs 3 bytes of color data. That means this image's data will consume 48 bytes of storage. But, we can compress

the data and pack it into a smaller number of bytes than 48! One way to compress data is to reduce repeated or redundant information. The most straightforward way to do this is called Run-Length Encoding. This takes advantage of the fact that there are often runs of identical values in files.

For example, in our image, there are 7 yellow pixels in a row. Instead of encoding redundant data: yellow pixel, yellow pixel, yellow pixel, and so on, we can just say “there’s 7 yellow pixels in a row” by inserting an extra byte that specifies the length of the run, like so: And then we can eliminate the redundant data behind it. To ensure that computers don’t get confused with which bytes are run lengths and which bytes represent color, we have to be consistent in how we apply this scheme. So, we need to preface all pixels with their run-length. In some cases, this actually adds data, but on the whole, we’ve dramatically reduced the number of bytes we need to encode this image. We’re now at 24 bytes, down from 48. That’s 50% smaller! A huge saving!

Also note that we haven’t lost any data. We can easily expand this back to the original form without any degradation. A compression technique that has this characteristic is called lossless compression, because we don’t lose anything. The decompressed data is identical to the original before compression, bit for bit. Let's take a look at another type of lossless compression, where blocks of data are replaced by more compact representations. This is sort of like “don’t forget to be awesome” being replaced by DFTBA. To do this, we need a dictionary that stores the mapping from codes to data. Lets see how this works for our example. We can view our image as not just a string of individual pixels, but as little blocks of data. For simplicity, we’re going to use pixel pairs, which are 6 bytes long, but blocks can be any size.

In our example, there are only four pairings: White-yellow, black-yellow, yellow-yellow and white-white. Those are the data blocks in our dictionary we want to generate compact codes for. What’s interesting, is that these blocks occur at different frequencies. There are 4 yellow-yellow pairs, 2 white-yellow pairs, and 1 each of black-yellow and white-white. Because yellow-yellow is the most common block, we want that to be substituted for the most compact representation. On the other hand, black-yellow and white-white, can be substituted for something longer because those blocks are infrequent. One method for generating efficient codes is building a Huffman Tree, invented by David Huffman while he was a student at MIT in the 1950s. His algorithm goes like this. First, you layout all the possible blocks and their frequencies. At every round, you select the two with the lowest frequencies. Here, that’s Black-Yellow and White-White, each with a frequency of 1. You combine these into a little tree... ...which have a combined frequency of 2, so we record that. And now one step of the algorithm done. Now we repeat the process. This time we have three things to choose from. Just like before, we select the two with the lowest frequency, put them into a little tree, and record the new total frequency of all the sub items. Ok, we’re almost done. This time it’s easy to select the two items with the lowest frequency because there are only two things left to pick. We combine these into a tree, and now we’re done! Our tree looks like this, and it has a very cool property: it’s arranged by frequency, with less common items lower

down. So, now we have a tree, but you may be wondering how this gets us to a dictionary. Well, we use our frequency-sorted tree to generate the codes we need by labeling each branch with a 0 or a 1, like so: With this, we can write out our code dictionary. Yellow-yellow is encoded as just a single 0. White-yellow is encoded as 1 0 (“one zero”) Black-Yellow is 1 1 0 and finally white-white is 1 1 1. The really cool thing about these codewords is that there’s no way to have conflicting codes, because each path down the tree is unique. This means our codes are prefix-free, that is no code starts with another complete code. Now, let’s return to our image data and compress it! Our first pixel pair, white-yellow, is substituted for the bits “1 0”. The next pair is black-yellow, which is substituted for “1 1 0”. Next is yellow-yellow with the incredibly compact substitution of just “0”. And this process repeats for the rest of the image: So instead of 48 bytes of image data ...this process has encoded it into 14 bits – NOT BYTES -- BITS!! That’s less than 2 bytes of data! But, this data is meaningless unless we also save our code dictionary. So, we’ll need to append it to the front of the image data, like this. Now, including the dictionary, our image data is 30 bytes long. That’s still a significant improvement over 48 bytes.

The two approaches we discussed, removing redundancies and using more compact representations, are often combined, and underlie almost all lossless compressed file formats, like GIF, PNG, PDF and ZIP files. Both run-length encoding and dictionary coders are lossless compression techniques. No information is lost; when you decompress, you get the original file. That’s really important for many types of files. Like, it’d be very odd if I zipped up a word document to send to you, and when you decompressed it on your computer, the text was different.

Lecture 5 Linux File System

Most people who have used Linux have seen the root directory but not everybody understands what the directories are used for. To a Windows user opening the file manager looks very much like opening the home folder in Windows and all looks very familiar. You've got your documents your downloads your pictures your videos same thing that is until they explore up the tree looking for the C Drive where's Program Files where's the directory that Linux is installed in -how do you find anything let me explain

I'll take a quick minute here for new Linux users coming from Windows. Windows and Linux evolved in very different ways once upon a time there was a thing called ms-dos the disk operating system. it was command-line only but you could still run programs games and WordPerfect but you didn't need Windows. Windows was added to PCs and you can install it on top of DOS you would start up your computer and type in win to start Windows. it used letters to assign drives with A and B being removable disks since early pcs only had floppy drives with the addition of hard drives the letter C became the letter for your internal disk additional discs were given the next available letter. you could install things in Doss wherever you wanted to. windows installed itself in his own directory called funny enough

Windows. later Microsoft changed how it booted by evolving their kernel to be less and less dependent on DOS and eventually allowed Windows to boot directly without dos at all. Microsoft's file directory structure kind of stayed the same. now Linux is different and so is its file structure. it also doesn't install applications like Windows does. starting with Windows 95 Microsoft created the Program Files directory which was the default installation directory for most applications .for the most part Linux follows UNIX traditions which is why uses the forward slash instead of the back slash like Windows. Linux also cares about capitalization. so you can have things like this file file file file file file. as you can see while they're all named file they all use different capitalization. so Linux will allow this because they're technically not named exactly the same. Mac users who have explored their hard drives might find Linux a little more familiar. this is because Mac's also evolved from a UNIX ancestor more specifically BSD. so let's have a look at the route and go over how all this work this layout for the most part is outlined in the filesystem hierarchy standard or FHS which defines the structure and layout and is maintained by the Linux Foundation. I want a note here that not all distributions follow this. some do their own special thing also several ways of structuring the folders has changed over the years but most of what follows still applies in most cases. so let's go end-to-end starting with bin in being short for binaries. these are the most basic binaries which is another word for programs or applications. things like LS to list your directory. cat to display the output of a file and other basic functions are stored here. skipping ahead a little bit I also want to point out s bin. these are system binaries that a system administrator would use and that a standard user wouldn't have access to without permission. both of these folders contain the files that need to be accessible when running in single user mode as opposed to the usual multi-user mode. single user mode is a special mode that boots you in as a root user to allow you to do system repairs and upgrades or testing. networking is usually disabled in this mode because of security issues. when you install a program in Linux it's typically not placed in these folders. next is boot this is a folder you don't want to play around in. it contains everything your OS needs to boot in other words your boot loaders live here. next we have cd-rom which I'm going to skip because it's not in all distros and it's more of a legacy mounting point for your cd-rom. so let's move on to dev. this is where your devices live. Linux again following UNIX has a standard where it was decided everything is a file. here you'll find your hardware a disk for example would be dev slash SDA here and a partition on that disk would be for example dev SD a1 SD a2 and so on you can also find everything else here from your webcam to your keyboard. this is typically an area that applications and drivers will access and is rarely something a user should be dabbling in. so going back to root the next folder is etc'. the name of this folder has been argued as standing for etc edit to configure as well as others. but it has been confirmed by Dennis Ritchie creator of Linux that it did indeed mean etc. this folder is where all your configurations are stored .however when I'm talking about configurations I'm talking about for things that are system-wide such as apt. in this folder for example you would find the list of all your sources what repos your system connects to as well as its various settings. so if you're looking for something that is a

system-wide application and not a per user setting. for example Libre Office would have settings in each user's folder and it wouldn't be system-wide because each user can have different settings and this brings me to the next folder which is home. however I'm going to save this for the end because there's some things I want to discuss about it so we'll come back to it later home folder. I'm going to save this for the end because there's some things I want to discuss about it. So we'll come back to it later.

Next are the Lib folders. This includes Lib Lib 32 and lib 64. These are where the libraries are stored. Libraries are files that applications can use to perform various functions. They're required by the binaries in bin and sbin for example. Moving on we have media and MNT or mount. These directories are where you would find your other mounted drives. It can be a floppy disk, USB stick, external hard drive, network drive or even a second hard drive. So if you're looking for that a B or D Drive this is where you want to be looking now.

This media folder wasn't always around. It was typically just MNT and that's where you mounted your storage devices. Nowadays most distros automatically mount devices for you in the media directory. So your USB stick that you inserted would be in media - user name - device name. So why are there two directories? Well if you're mounting things manually use the MNT directory and leave the media directory to the OS to manage. Most distros and file managers such as Nautilus for example what I'm using here and dolphin and PC man FM will have something on the side here. For example, in Nautilus I can click other locations and here I can access my other devices. If I had a USB stick plugged in right now it would show up here and I could simply click on it and access it

Next down the line is opt. This is the optional folder which is usually where manually installed software from vendors resides. Though some software packages found in the repo can also find their way here. VirtualBox guest additions is one example. So here for example is a VPN software that I installed and the drivers for my brother printer slash scanner. This is also where you can install software you've created yourself. This folder is where I place all the applications I've written first I on Linux.

Next we have proc. Proc is where you'll find pseudo files that contain information about system processes and resources. For example, every process will have a directory here which contains all kinds of information on that process. an example I can show you here if I open the system monitor I can see Dasia due monitors process ID or hid is two three four four. so if I navigate to proc two three four four which is the pit for the Dasia dupe monitor I can see all kinds of pseudo files here. This is much like dev where they're not actually files on the system. this is the kernel translating other information to appear as files. So for example here I can open the status file and it'll show me all kinds of information on that process. There's tons more in here but this isn't something you want to play in if you are a developer if you're writing applications this is very handy. Here you can also find information like for the CPU for example. This will give you all kinds of information on the CPU and you can also do up time which will print out your uptime for your system.

Next is root. Root is the root users home folder. unlike a user's home folder, it does not contain the typical directories inside and it does not reside in the home directory. You can store files here if you wish but you need root permissions to access it. The location of this directory also ensures that root always has access to its home folder in case you have the regular users home directory stored on another drive which you cannot access. Next is run. This one's fairly new and different distros use it in slightly different ways. It's a temp FS file system which means it runs in RAM. This also means that everything in it is gone when the system's rebooted or shut down. It's used for processes that start early in the boot procedure to store runtime information that they use to function. We've already covered sbin. So next down the line is snap. This is a folder where snap packages are stored and are mainly used by Ubuntu. Snap packages are completely self-contained applications that run differently than regular packages and applications. This will be covered in a future on its own since it'll take more time to explain. SRV this is the service directory where service data is stored. It'll probably be empty for you but if you run a server such as a web server or FTP server you would store the files that will be accessed by external users here. This allows for better security since it's at the root of the drive and it also allows you to easily mount this folder from another hard drive.

Next down the line is sys. The system folder has been around a long time. It's a way to interact with the kernel. One older example is writing to a file using VGA switcheroo and change settings on graphic cards in a hybrid system. This directory is similar to the run directory and it's not physically written to the disk. It's created every time the system boots up. So you wouldn't store anything here and nothing gets installed here.

TMP is of course a temp or temporary directory. This is where files are temporarily stored by applications that could be used during a session. One example is if you're writing a document in a word processor it will regularly save a temporary copy of what you're writing here. So that if the application crashes it can look here to see if there's a recent saved copy that you can recover. This folder is usually emptied when you reboot the system. On occasion you might find some files or directory that remain and could be stuck there because the system can't delete them. This normally isn't a big deal unless there's hundreds of files or the files are taking a lot of disk space. In which case you might want to log in as the root user in single user mode navigate to this folder and manually delete them.

Moving on we have the USR folder. This is the user application space where applications will be installed that are used by the user. As opposed to the bin directory is used by the system and system administrator to perform maintenance. It's also known as the UNIX system resource and any applications installed here are considered non-essential for basic system operation. Installed applications will reside in one of several places here such as user bin user sbin or local bin local sbin with their required library stored in local user, local Lib or user Lib. Most programs that are installed from source code will end up in the local folders. Many larger programs will install themselves into user share. Any installed source code such as the kernel source and header files will go into the SRC directory. This directory seems like a

confusing mess at first and while the directory structure and what goes where is laid out in the FHS I mentioned earlier. You'll still have to sometimes look in other places to find things. Someone making a certain application might not adhere to the standard and could just do what they want. Also some distros may treat these folders differently as well. Going back to root we have next var. Var is the variable directory. It contains files and directories that are expected to grow in size. For example, var crash holds information about processes that have crashed. Var log contains log files for both the system and many different applications which will constantly grow in size as you use the system. You'll also find other things in here like databases for mail and temporary storage for printer queues also known as the spool.

And finally we will come back to the home folder. When you enter the home folder you'll see that each user has its own folder inside of it. The home folder is where you store your personal files and documents. Like I said each user has their own home folder and each user can only access their own unless they use admin permissions. some users mount the home folder on a different drive or different partition which allows you to reinstall your system and preserve your files. The home folder also contains many different directories which store your application settings. A hidden directory is simply one that starts with a period. Linux hides these by default. You can view them in the file manager by selecting show hidden files or by pressing ctrl+H. This is of course using Nautilus in gnome and some file managers might be different. PC man FM is also press ctrl H to view hidden files. If you're in the terminal and you list files, it'll only show you what is not hidden unless you specify dash A for all and now you can see all your hidden files. These hidden directory store things like cache. Some applications like a browser used to store temporary files. Other applications might store thumbnails or information that will be used over and over repeatedly. Then you have folders like config and local which store individual application settings. Genie for example can be found in config. So here's the geany folder. Any settings that I change in the geany options are saved here. If I go back to the home folder you can see that some applications store their settings straight into the home folder like GIMP for example. These hidden folders are also where your desktop settings are saved. Whether you use open box KDE gnome unity. They all save their settings here such as what wallpaper you use what theme you use and so on. You can even place your icons and themes in these folders. so that you can have a custom look and easily save them for reuse. These hidden folders are important if you want to back up your files and your settings.

If you don't customize your system or you don't care to then you can simply backup all the folders, you see here. If you want to save all your settings, then you might want to include all the hidden files as well. So if you reinstall your system you simply log in and all your theming will already be done just like you left it. You will have to reinstall your applications but once you install them the settings you set for them will already be in place and the applications will run just like they did before. So as you can see Linux is kind of similar to Mac but very different from Windows. Although it seems like a mess it's actually a more efficient way of doing things and allows much more sharing of common resources between packages. When it comes

to adding and removing software your distro will have a package manager that will handle all this for you. Package manager tracks where everything is going so that when you remove your package it takes all those files with it. Ok, let's repeat. Everything begins in your system from this directory. All your folders, hard drives, USB drivers, everything is located in this root folder. You cannot go above this directory. Also, the root directory is designated by the slash sign. The concept of the root directory may be difficult to understand for Windows users who are used to see something like disk C, disk D and disk E. In Linux, every disk is a represented as a folder that is mounted under this root directory. This Linux Directory Structure may look like a mess, but believe me when you learn it, you will realize how much sense it makes.

The /bin folder contains programs that are essential for the system to boot and run. So, if you destroy this folder, your system won't boot and run. These programs are stored in the binary format. In other words, they are not in text format. You cannot open and read the content of these programs. The advantage of such format is that a computer can read and execute these programs very fast.

It is easy to guess from the name. This folder is needed to boot your system. It contains the Linux kernel, initial RAM disk image for drives need at boot time, and the bootloader. I also would like to point out that within this boot folder, you can find the grub folder that contains grub configuration files. The boot folder also contains the Linux kernel. Here, I need to introduce another important concept of Linux – everything is a file.

The /dev folder contains files for all devices your Linux is able to recognize. If you have some Linux experience, you may recall that when you mount a hard drive, you use a name such as /dev/sda1. The sda is the name of a first hard drive recognized by your Linux kernel and it is located in the dev folder. When the disk is mounted, you see it as a folder in that mounting point. You can also find here usb devices, cpu etc.

The /etc folder comprises all system-wide configuration files and some shell scripts that are executed during the system boot. All files here are text files, so they are human readable. If you ever did any system-wide configuration, you probably edited some files here. For example, there is /etc/fstab file that contains a table of storage devices and their mounting points.

The home directory contains a home folder for each regular user on your Linux system. For example, you can see here the folder ALU, which is my home folder. And one more folder that belongs to another user, whom I named User2. So, the home folder of every user is named by its username. You have as many folders as many users you have on your system. These users' folders are where users store their private data such as documents, videos, picture, music etc. When you open your file manager or your terminal by default you are located in you user's home folder.

You already know the /bin directory that contains programs, this /lib folder contains libraries required by those programs from the /bin folder. A library is a set of functions that are shared between programs. Thus, this /lib folder is also essential for your system to work correctly.

You will have this directory if you use the ext4 file system. Most of the modern Linux distros use ext4, so most likely you have this folder. This is a file system specific folder that is used for data recovery in case of file corruption. Unless something bad has happened, this folder should be empty on your system. This /lost+found folder is produced on every separate partition. So, if your /home folder is on a separate partition, you should have this /lost+found folder in your home directory too.

This folder is used for automatic mounting of removable media such as USB drives, CD-ROM etc. For example, if your system is configured for automatic mounting, when you insert a USB drive it will be mounted to this folder.

The /mnt folder is similar to the /media folder, it is also used to mount devices, but usually, it is used for manual mounting. You, of course, can manually mount your devices to /media, but to keep some order in your system it is better to separate these two mounting points.

This folder is not essential for your system to work. Usually, it is used to install commercial programs on your system. For example, my Dropbox installation is located in this folder.

This is a virtual file-system maintained by the Linux kernel. Usually, you do not touch anything in this folder. It is needed only for the kernel to run different processes.

This is the home directory of your root user. Don’t mix it with the / root directory. The / directory is the parental directory for the whole system, whereas this /root directory is the same as your user home directory but it is for the root account. If you log in as a root, you will be located in this directory by default. This is a folder for private data and account specific setting of your root account.

The /run is a recently introduced folder that is actually a temporary file-system. It is used to store temporary files very early in system boot before the other temporary folders become available.

Similar to /bin this folder contains binaries for essential system tasks but they are meant to be run by the super user, in other words, the administrator of the system.

This directory contains service files installed on your system. For example, if you installed a web-served on your Linux system, it will be located in this folder.

This is just a place where programs store temporary files on your system. This directory is usually cleaned on reboot.

This is probably the largest folder after your home folder. It contains all programs used by a regular user. I would like to stop little more on sub-directories of this /usr folder. /usr/bin contains the programs installed by your Linux distribution. There are usually thousands of programs here. The libraries for this /usr/bin executables are located in the /usr/lib folder. The /usr/local doesn't have any programs by default, but if you compile and install a program system-wide it will be placed here. The most useful folder is /usr/share.

It contains all the shared data used by the programs from /usr/bin. All default configuration files, themes, icons, wallpapers, sound files are stored here, one more

folder I would like to mention here is the /usr/share/doc folder, where you can find the documentation files for programs installed on your system.

The /var contains files that are of variable content, so their content is not static and it constantly changes. For example, this is where the log files are stored. If you don't know, a log file is a file that records all events happening in your system while it is running. These log files often help to find out if something is not working correctly in your system.

Linux Command Line.

First of all I would like to give you some motivation on how the command line can be useful. I have recently started my Instagram account, and I need to copy all my thumbnails into a separate folder and adjust them for Instagram. But the problem is a project for every video has its own folder. And at that moment I have more than 50 folders. So, if I did not know how to use the command line, I would have to enter each of these folders manually and copy every thumbnail individually. Luckily, I know how to achieve the same result with one simple command. I opened the terminal, and type the command to copy then I specify the path to my YouTube folder and I use this star sign, which means to enter every directory and I use the star again to copy every file with the XCF extension. XCF is the format for GIMP files and I created all my thumbnails with GIMP. I run this command. There are some errors because some files are duplicated. It's ok. Now, if I open the thumbnail folder, I can see all my thumbnails. I also copied some files which are not thumbnails, but it is not a problem to delete them. Now, I can prepare these thumbnails for the Instagram upload.

Our camera writes video in MOV format. The files are usually huge. Even a few minutes video, can be more than one gigabyte. Many of these videos are just documentary, so I prefer to convert them to mp4 and reduce the amount of space they use. I run this command. it loops through all the MOV files and converts them to mp4. I believe you can do this conversion with graphical programs too, but it would take many mouse clicks. In the Linux Command Line, it is just this one line. Now as you can see every file has been converted. So, I can remove the MOV files. I can also use the terminal to do that in the fastest way possible.

Similarly to the video conversion you have just seen, I can resize many images by running this single line of code. Every file will be reduced in size by 50% It can be very handy if you need to send many images by email, for example. This is the original file and this is the compressed file. The difference in file size is quite significant.

There is also a simple way to rename many files. For example, I have many TXT files here and let assume I need to rename them to CSV format. I just run this command. and it will change the extension from txt to csv for all TXT files. Now I have only CSV files.

You can also do such simple thing as check the bitrate of a song. Bitrate defines the quality of a music file. You just run this command And here is the bitrate of a song. You can, of course, apply this command to as many files as you want at once. And you will get the information for all files almost instantly.

This is a more advanced level and many users will never have to deal with this. But I just want to show you the possibilities the command line. The command line becomes irreplaceable when you need to work with really large files. By large, I mean many Gb. For example, this text file is 24 Gb and it is a gzip compressed file. So, if I uncompressed it, it would be more than 100 Gb. There is no way you can open such a file in your text editor. But in the command line, you can open it. Moreover, you don't need to uncompress it. You need to run this command to uncompress it on the fly and then open it with less command. Now we can scroll through this file. Well, it will take ages to scroll the whole file, but if you need to look at the content of the file, this is the way to go. There are many other tools you can use to work with such big files, less is just one of them.

I have quite a few files here. They are not very large, but not very small. Each of them contains almost 30 000 lines. And these are the top ten lines of one of them. Imagine a situation, when I needed to replace this name Carubv with another one in these files. Technically you can open them in a text editor or LibreOffice Calc, and use replace all. But it will take quite a lot of time. In the command line, you just run this command And replace the old name with the new one. Now, let's check the top ten lines of any file again. All the strings have been replaced. You can see the new row names. And let's also check the last ten lines of the file, to make sure everything has been replaced Everything is fine here too.

The previous examples probably were not useful for everyone, but if you run Linux sooner or later you will have to use the command line to maintain your system. The command line also gives you more power in managing your system. For example, the updates in the graphical programs will often be under some generic names. But this is how they look in the terminal. It is less user-friendly, but this way you will be able to actually learn your Linux system. Besides, sometimes graphical update manager may fail, and you will have to update through the terminal. Knowing at least some basics of the Linux Command Line will help you a lot in this process.

You will also need some command Line skills if something happened to your system and you cannot boot into the graphical interface. The Command Line is the only interface you have. Your actions will depend on what you have done before you lost the graphical interface, but it is always the easiest to restore your system from a backup as I showed in one of the previous videos.

Finally, you can install this hollywood program, start it, make you terminal full screen by pressing F11 and let your friends watch you and think that you are doing some hacking.

Navigating in the Linux Command Line.

In this part of lecture, you will learn how to navigate in your Linux system using the terminal. In particular, you will learn how to print the current working directory, how to change your directory and how to lease the content of your directory. Let's get started.

To show your how the navigation works I will be using both the terminal and the file manager. so when you open the terminal as well as the file manager you are always located in your home user directory. so if you type PWD which means print

working directory you will see that we are located in home our user name. you can see the same in the file manager. so if you click here you can see the current working directory is home and my user name. so it may be easier to see it here in the tree structure. so these are the all folders of the Linux system and I'm located in the directory home with the username ALU and there is another folder with another user name here. so let's list the content of this folder. if you type the command LS which means to leave the content you will see the same folders in the terminal which you can see here in the file manager. so let's navigate into some of these folders. to navigate into the folders you need to type the command CD which means change directory and let's go into the directory pictures. here again you can type PWD. it will show you that you are located in the directory home you username pictures. so again we've navigated from this directory into the directory pictures and we can change this directory here as well. so if it is the content of the directory pictures you can see we have the folder wallpapers which is here and several images.

so now let's assume you want to go back to your home directory. you need to jump one level up. what I mean I need to go from the directory pictures into the directory ALU. so I type CD and then I type two dots and two dots means to jump one level up. and now I jump back to my home directory. as you can see I am back in ALU directory. so if I type this command again I will jump into my home because I jump one level up again so I jump from here into here. so PWD... yeah now we are located in our home. you can of course use the absolute path. for example if we want to navigate back to our pictures directory we can type CD slash home user name and the word pictures. as you can see we are back into our directory pictures. so which is this one. you can use this navigation by levels up into more complicated way. for example if you type CD two dots then you put slash two dots again it will mean to jump two levels up. so I press ENTER and as you can see I jumped from pictures to ALU and then I jump one level more to my home directory. and you can confirm it here yes we are back in our home directory.

let's go to the very root of our system. so which means CD and the sign / .so now we are at the root of our system and the root of the system... it means top directory of all of these directories all of these directories they amount it to slash well you don't see it here but this is how it works. so in a Linux system if you want to go back to your user home directory you can use this shortcut you just type command CD without any sign you just type it and you press Enter. by default it will always bring you back to your user home directory. for example if I navigate to CD / I don't know... usr bin and we type CD again we've been to this directory and now we are back in our home directory. so we can confirm it again so we are now home directory you can also see in the terminal prompt this sign This sign also means my home directory. for example if I navigate to the pictures directory let's say to the wallpapers which is located in the pictures so now I'm located here in the three structure you can see that I was in my home then I was in my user directory and then I jump into pictures and from pictures and jump into the directory wallpapers. but what if I want to navigate from here quickly for example to My Documents. so there are many ways to do it you can jump one level up one level up and then you can navigate to your

document. so you will need to type three commands or you can type cd without any sign so we will jump back to a username and then you can type CD to downloads. but there is a shorter way. you can type CD and then you put this sign which means to go back to your home directory. so this sign is equivalent to this command. then you type the Documents folder and you see we are in our Documents folder. so just remember if you type CD without any sign you always jump into user directory. if you type CD with this sign it will be equal to empty CD. so you will always jump to your user directory. but if you type CD with this sign then you can use the name of any directory which is located in your home directory. finally I would like to show you one more thing for example if I want to go to your previous working directory. previously we were in the directory of wallpapers and then we jump to the directory documents. and you want to go back to the directory of wallpapers. you just type command CD and you put the dash sign which means to jump to the previous working directory. as you can see now we are back in our wallpapers directory. and you can go back to home if you type CD without any sign.

So let's clean this terminal and repeat the commands again. so PWD prints current working directory. LS lists the content of the current working directory. CD allows you to change your working directory. you can use CD and then you type the name of the directory where you want to navigate. we used pictures in this case. if you want to navigate back to your home directory from anywhere you just type CD without any sign. if you want to go one level up you type CD with two dots. and if you want to navigate back to your previous working directory you type CD with a sign dash. now you know how to navigate in your Linux system using the command line. In this part of lecture you will lean how to manipulate files and folders from the command line. In particular, you will learn How to copy, move and remove files and folders as well as how to create new files and folders Let's get started.

So, let's create a new file. In Linux you need to type command echo Then you type this sign And let's name this file file dot txt. You type LS again. Here is out new file. And you can track these changes on the right side, with the graphical interface. So, now let's also create a new directory. To create a directory, you type MK, which is a short of MAKE and then DIR. it's quite easy to remember ... make... directory And we type a name, new directory. NewDir LS Here is out new directory. and here is our new file. Now, let's copy this file into this new directory. The command is very simple. CP - to copy Then we type the file name you want to copy. and you type the destination where you want to copy it. In our case, it's New Directory. So, we've copied it. Now, let's list the content of the new directory to check if the copying was successful. Here, you can also learn than you don't need to change you directory to list it's content. you can type LS and the path to the directory. which you want to check. In this case, we that file.txt is located in the directory New Directory. Again we can go to the graphical and we can confirm that it's true. But you can also use the command copy. to copy a file with a different name. For example, if you type copy file Then we type again New Directory. And we name it here, file2 for example. So, we list the content of the new directory again. Now we have two file. And you can see them here as well. In the new directory. So, you can copy file with it's name as

before. into new directory, or you can also copy it with a new file name You can also make a copy of this file within the same directory. For example, if you type copy file then you type file three do txt LS Now we have two files - the original one and the new one which has been just created. with the name 3 in it. So, we have created some file, we copied them into our new directory. But you can also move files. So, let's move file3 into the new directory. You type MV which is a short of move. very easy to remember. file 3 do txt New Directory. And again, we check it's content. Now we have three files in the directory New Directory. and you can see them here as well. and the move command works similar to copy. you can also move a file and give it a new name as it was here. for example, if we type move. file dot txt New Directory file, for example, zero TXT. and we type LS New Directory Now, the file has been moved and renamed. Basically, you can use the command move to rename files. Let's demonstrate it again. If I... go inside New Directory and I move file zero to file for example, one dot TXT. Now, you see the file has been renamed. So, the move command is used primarily to move files, but it's kind of similar to renaming files as well. if you give it a new name. as I have done here. Now, let's learn the command to remove files. And to remove a file you can type a very simple command. It's just two letters. RM and the file name you want to remove. for example file dot TXT and then we type LS. Now we have only three files. the file.txt has been removed. and you can see. It's.. it disappeared from here as well. And here I would like you to pay some attention. Because remove in the command line removes files forever. for example, in the file manager you can move them to Trash, which is different from removing a file. Because in your file manager you can also have a command delete. delete will also remove files forever. But move to trash, just moves then to trash. so, you can restore them in a file manager. But in the command line there is no way to restore it. After you removed it, it will disappear. So, be very-very careful with the remove command. we can also remove several files at the same time. for example, if you type the remove command then you put start sign dot TXT. it means to remove all files which have any name but end with dot txt. and all files have disappeared. So, now if we go back to our home directory. So, we have New Directory here left only. So, let's remove it. To remove a directory in the Linux Command line, you need to type command remove dir and the name of the directory. So, the directory has been removed. But here you also need to pay attention because remove directory will only work if your new directory is empty. For example, if we create the New Directory Then let's say we want to create a file inside that directory. we type echo New Directory file dot TXT. So, this will create a file within the directory New Directory So, let's go back to our home. here, and we can check that we have a new file here. You can again copy this path. LS And you can see that it's true. The file is located in the New Directory So, let's remove it now. if you type the command remove new directory. which is not empty anymore. It will complain that the directory is not empty. So, one of the ways to remove this directory. is to go inside into this directory. remove file I'm not gonna do that, but if you remove this file the directory will be empty, so you can go back and remove this... and remove this new directory. but there is an easier way if you type command remove and you add the option –r which

means to remove everything recursively. you can remove this directory and all it's content. But you should also be very careful with this command. Because remember.. it will remove these files forever. So, before you type it make sure you type everything correctly and you know that you want to remove this directory as well as all its content. Press enter. you see everything disappeared from here. And the directory has disappeared from our home. I type LS, there is no directory New Directory And the move command works the same with directories. I didn't show it but let's do it again. Mkdir New Directory and we will move New Directory to... let say Downloads cd Downloads Here is our new directory inside the Downloads folder. We can do the same in graphical. You can see it's here. And let say you want to rename your directory. you type move New Directory and then you type new dir two for example. LS The directory has been renamed. We had newDir before, now it is newDir2 Don't worry if it's overwhelming for you now. we will be using these commands during the whole tutorial series. So, you will lean them Now you know how to manipulate files and folders using the Linux Command Line.

Lecture 6 Computer Network

A computer network is a system in which multiple computers are connected to each other to share information and resources. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet. Advantages of Computer Networks:

- File sharing

The major advantage of a computer network is that allows file sharing and remote file access. A person sitting at one workstation that is connected to a network can easily see files present on another workstation, provided he is authorized to do so.

- Flexible access

A user can log on to a computer anywhere on the network and access his files. This offers flexibility to the user as to where he should be during the course of his routine.

- Entertainment

Many games and other means of entertainment are easily available on the internet. Furthermore, Local Area Networks (LANs) offers and facilitates other ways of enjoyments, such as many players are connected through LAN and play a particular game with each other from a remote location.

- Better connectivity and communications

It allows users to connect and communicate with each other easily. Various communication applications included e-mail and groupware are used. Through e-mail, members of a network can send a message and ensure safe delivery of data to other members, even in their absence.

- Internet access

Computer networks provide internet service over the entire network. Every single computer attached to the network can experience the high-speed internet.

- Inexpensive system

Shared resources mean reduction in hardware costs. Shared files mean reduction in memory requirement, which indirectly means a reduction in file storage expenses. A particular software can be installed only once on the server and made available across all connected computers at once. This saves the expense of buying and installing the same software as many times for as many users.

- Resource sharing

All computers in the network can share resources such as printers, fax machines, modems, and scanners.

- Instant and multiple access

Computer networks are multiple processes. Many users can access the same information at the same time. Immediate commands such as printing commands can be made with the help of computer networks.

Disadvantages of Computer Networks

- Lack of data security and privacy

Because there would be a huge number of people who would be using a computer network to get and share some of their files and resources, a certain user's security would be always at risk. There might even be illegal activities that would occur, which you need to be careful about and aware of.

- Presence of computer viruses and malware

If even one computer on a network gets affected by a virus, there is a possible threat for the other systems getting affected too. Viruses can spread on a network easily, because of the inter-connectivity of workstations. Moreover, multiple systems with common resources are the perfect breeding ground for viruses that multiply.

- Lack of Independence

Since most networks have a centralized server and dependent clients, the client users lack any freedom whatsoever. Centralized decision making can sometimes hinder how a client user wants to use his own computer.

- Lack of Robustness

As previously stated, if a computer network's main server breaks down, the entire system would become useless. Also, if it has a bridging device or a central linking server that fails, the entire network would also come to a standstill.

- Need an efficient handler

For a computer network to work efficiently and optimally, it requires high technical skills and know-how of its operations and administration. A person just having basic skills cannot do this job. Take note that the responsibility to handle such a system is high, as allotting permissions and passwords can be daunting. Similarly, network configuration and connection is very tedious and cannot be done by an average technician who does not have advanced knowledge.

Use (Applications) of Computer Networks

- Financial services

Nowadays, almost all the financial services depend on the computer network. You can access the financial services across the world. For example, a user can transfer money from one place to another by using the electronic fund transfer feature. You can use networking in various financial areas such as ATM, foreign exchange and credit history search.

- Business

Nowadays, most of the works of businesses are done over the computers. To exchange the data and ideas, you need effective data and resources sharing features. To do this, you need to connect the computer with each other through a network. For example, a person of one department of an organization can share or access the electronic data of other departments through a network.

- Email services

A computer network provides you the facility to send or receive emails across the globe in few seconds.

- Mobile applications

By using mobile applications, such as cellular or wireless phones, you can communicate (exchange your views and ideas) with one other.

- Directory services

It provides you the facility to store files on a centralized location to increase the speed of search operation worldwide.

Teleconferencing

It contains voice conferencing and video conferencing which are based on networking. In teleconferencing, the participants need not be presented at the same location.

Types of Computer Networks

LAN (Local Area Network)

- It is privately-owned networks within a single building or campus of up to a few kilometers in

size.

- They are widely used to connect personal computers and workstations in company offices and factories to share resources (e.g., printers) and exchange information.

- LANs are easy to design and troubleshoot

- In LAN, all the machines are connected to a single cable.

- Different types of topologies such as Bus, Ring, Star, and Tree are used.

- The data transfer rates for LAN is up to 10 Gbits/s.

- They transfer data at high speeds. The high transmission rate is possible in LAN because of the short distance between various computer networks.

- They exist in a limited geographical area.

Advantages:

- LAN transfers data at high speed.

- LAN technology is generally less expensive.

- MAN (Metropolitan Area Network)

- MAN is a larger version of LAN which covers an area that is larger than the covered by LAN but smaller than the area covered by WAN.

- A metropolitan area network or MAN covers a city. The best-known example of a MAN is the cable television network available in many cities.

- MAN connects two or more LANs.

- At first, the companies began jumping into the business, getting contracts from city governments to wire up an entire city.

- The next step was television programming and even entire channels designed for cable only.

WAN (Wide Area Network)

- WAN spans a large geographical area, often a country or region.

- WAN links different metropolitan's countries and national boundaries thereby enabling easy communication.

- It may be located entirely within a state or a country or it may be interconnected around the world.

- It contains a collection of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines hosts.

- The communication between different users of WAN is established using leased telephone lines or satel’lite links and similar channels

Internet

- The internet is a type of world-wide computer network.

- The internet is the collection of infinite numbers of connected computers that are spread across the world.

- We can also say that the Internet is a computer network that interconnects hundreds of millions of computing devices throughout the world.

- It is established as the largest network and sometimes called a network of a network that consists of numerous academic, business and government networks, which together carry various information.

- The Internet is a global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardized communication protocols.

- When two computers are connected over the Internet, they can send and receive all kinds of information such as text, graphics, voice, video, and computer programs.

Concept of WWW.

The World Wide Web (WWW)is a global information medium which users can read and write via computer connected to the internet. The Web, or World Wide Web, is basically a system of Internet servers that support specially formatted documents. The documents are formatted in a markup language called HTML (Hypertext Markup Language) that supports links to other documents, as well as graphics, audio, and video files. In short, World Wide Web (WWW)is collection of text pages, digital photographs, music files, videos, and animations you can access over the Internet.

The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. TheWeb also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.

Web Browser and Web Server.

Web server and web browser are the terms which are commonly used for website. The basic purpose of both is to develop a platform for internet web directory. So that any users can anytime access any kind of website. Major difference between them is on their function and how they perform their functions.

Web browser is a client, program, software or tool through which we sent HTTP request to web server. The main purpose of web browser is to locate the content on the World Wide Web and display in the shape of web page, image, audio or video form. Web server is a computer system, which provides the web pages via HTTP (Hypertext Transfer Protocol). IP address and a domain name is essential for every web server. Whenever, you insert a URL or web address into your web browser, this sends request to the web address where domain name of your URL is already saved. Then this server collects the all information of your web page and sends to browser, which you see in form of web page on your browser.

Basics of Information and Network Security.

In daily life we use information for various purposes and use network for communication and exchange information between different parties. In many cases this information is sensitive so we need to take care that only authorized party can get that information. For maintaining such privacy we require some mechanism or physical device which ensures that it is safe. Such mechanism or physical devices are known as security system. Computer Security: The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources.

This definition of computer security introduces three key objectives that are at the heart of computer

security:

1. Confidentiality: It covers two concepts

Data Confidentiality: Assures that private or confidential information is not made available or disclosed to unauthorized individuals.

Privacy: Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

2. Integrity: It covers two concepts

Data Integrity: Assures that information and programs are changed only in a specified and authorize manner. System Integrity: Assures that a system performs its

intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.

3. Availability: Assures that systems work promptly and service is not denied to authorize user.

PART 3. LABORATORY WORKS.

Contributors of EIMCR are not pretended on any copyrights of following materials of laboratory works.

Rules of reports making

Student's instruction:

The option is selected by the number of the record book (by the last two digits). If the record book number is greater than the number of questions, the option is counted as follows: for example, the record number is 38, the options are 20; the number of options should be subtracted from the record book number, the remaining number – 18 – and there is the option number.

Requirements:

- font - Times New Roman, 12 – 14 pt;
- line spacing – from 17 pt to one and a half;
- text alignment – in width;
- pages should be numbered;
- the use of selection – at will, but not overdoing;
- should be placed links to used literary sources – the source number is placed at the end of the sentence before the full stop and taken in square brackets. Example: text text text text text [7] – this means that you learned about text text text text text text from the book / source with number 7 in your references.

The volume of your work should be at least 10 printed sheets of A4 format and includes at least two chapters, introduction, conclusion and list of references.

In the main part of the work, the following «directions» are required:

- theoretical part (answers for theoretical questions);
- practical part (algorithm schemes, pseudocode, etc.);
- conclusion on the work done.

Variants for individual tasks:

According to the student's instruction, select the variant of the task (if it necessary). Sketch of laboratory report must include at least next parts:

BELARUSIAN NATIONAL TECHNICAL UNIVERSITY
INTERNATIONAL INSTITUTE OF DISTANCE EDUCATION
«INFORMATION SYSTEMS AND TECHNOLOGIES»

LABORATORY WORK № ___

Variant ___
«Computer science»

Done by:
student, group № _____
Full name

Checked by: teacher’s name (Vladimir Stepanov)

Minsk 20___

Theoretical part

Theory about necessary topics.

Practical part

Tasks:
1. Reorder of shortcuts of Desktop;
2. Change color of Desktop background.

Reorder of shortcuts of Desktop:

1. Initial order of Desktop shortcuts (figure 1):

Figure 1 – Initial order of Desktop shortcuts

2. Another order of Desktop shortcuts (figure 2):

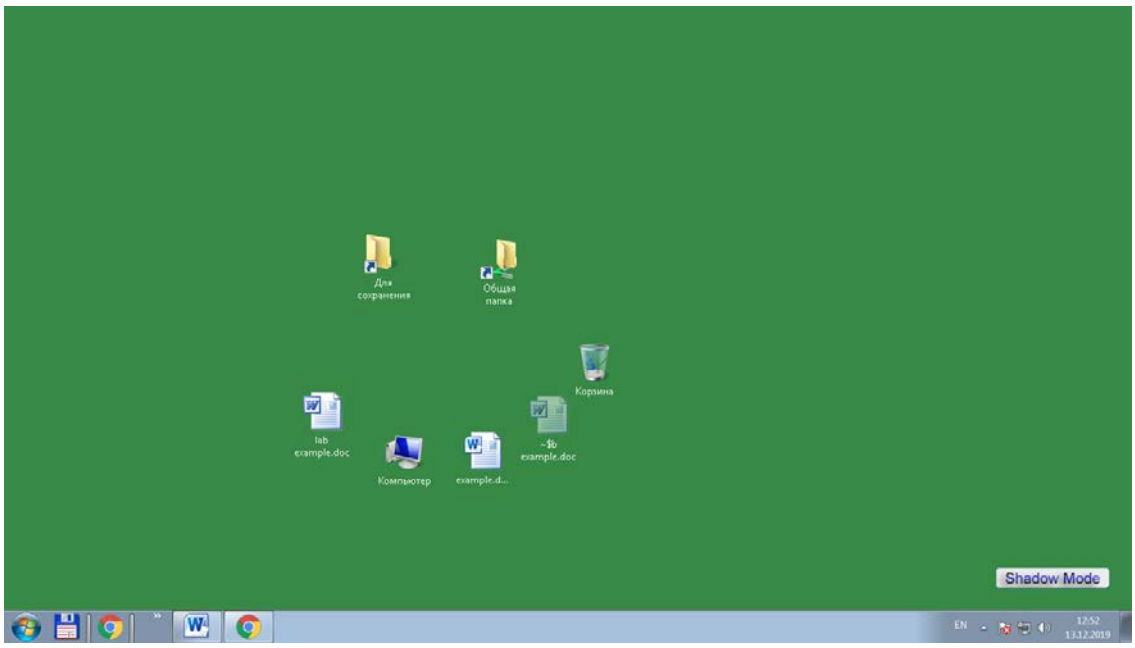


Figure 2 – Another order of Desktop shortcuts

Change color of Desktop background.

1. Open «Personalization» window (figure 3):

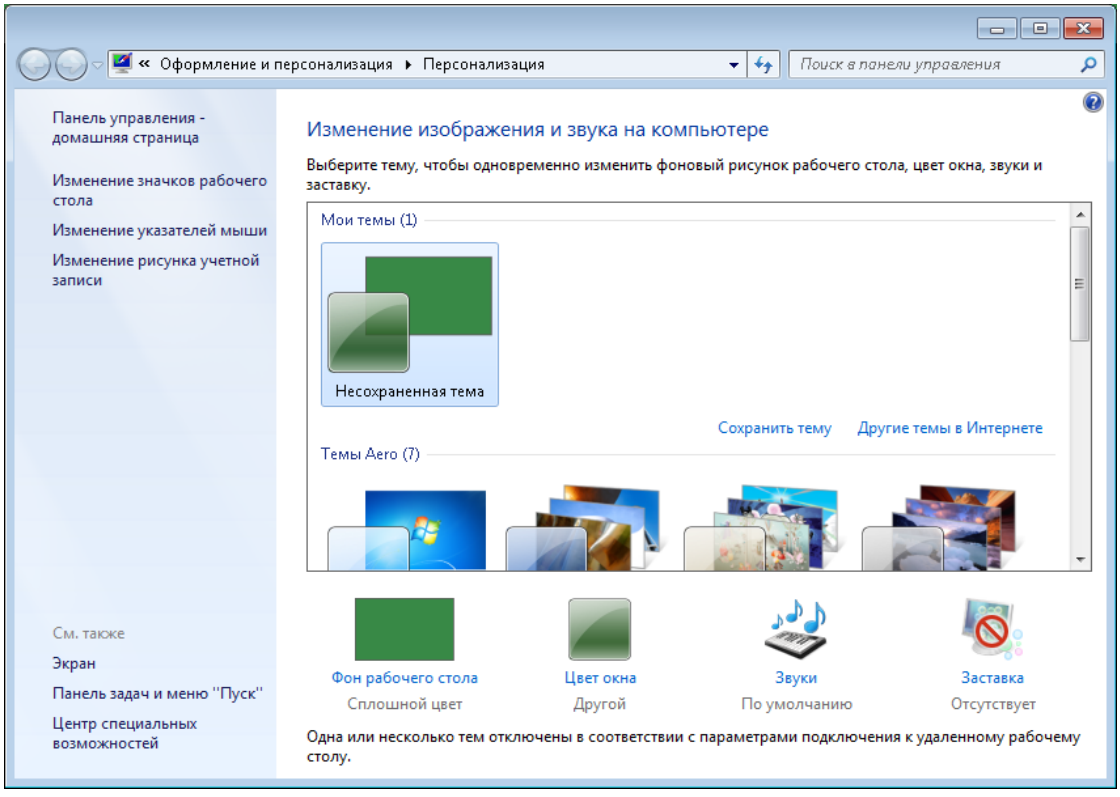


Figure 3 – «Personalization» window

2. Intermediate result (figure 4):

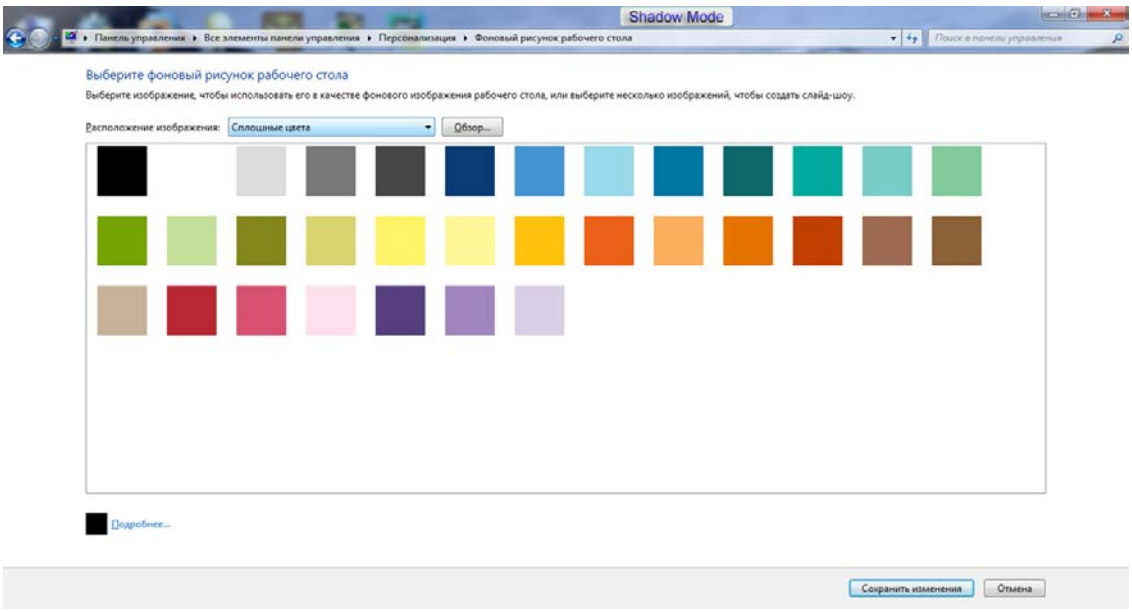


Figure 4 – Intermediate result

3. Choose color from the solid window colors (finish result is shown in figure 5):

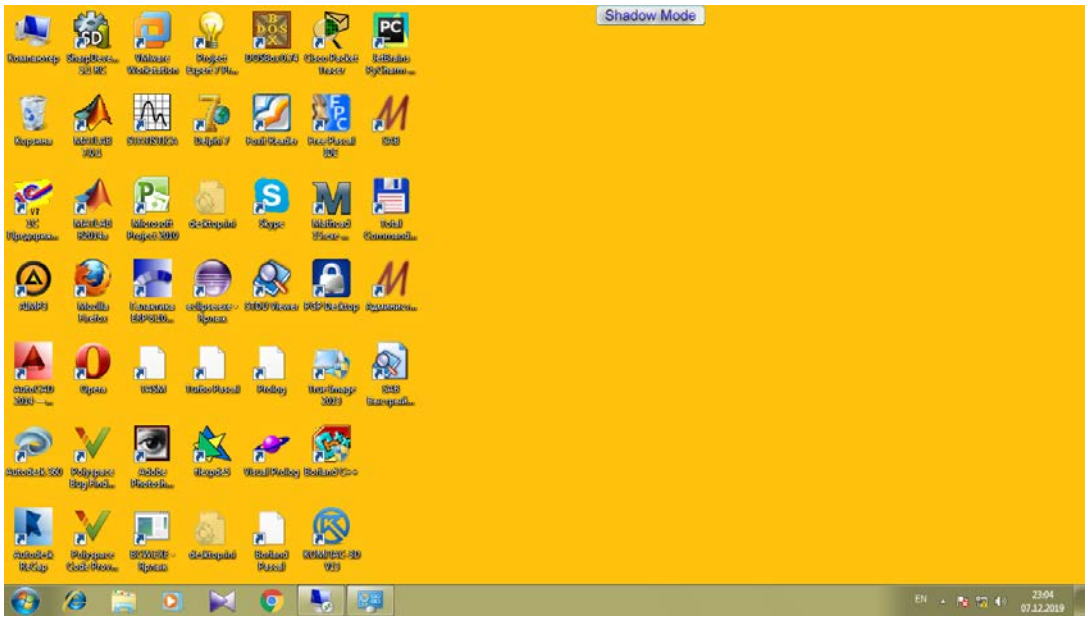


Figure 5 – Finish result

SUMMARY

Summary by laboratory work (all tasks done…)

REFERENCES

List of references…

Laboratory work 1

1. Find common information about Microsoft Word application.
2. Do the exercise:
1. Open a Blank document
2. Save it as WordLabTwo.docx to Documents > MSWordLabs folder (Create the folder if it does not exist)
3. Close WordLabTwo.docx; reopen it
4. Starting at the first line in the document, click Title from the Styles group; type the title “WORKING WITH MS WORD”
5. Center the title “WORKING WITH MS WORD”
6. Press enter twice after the Title; click Heading 1 from the Styles group, then type the heading “Open and Save a Document”; press enter once
7. Type the following paragraphs below the heading “Open and Save a Document”:
8. Open MS Word document. Use the Format Painter to apply Bold to the text.
9. Apply Italic to the paragraphs
10. At the end of the second paragraph, place the Insertion Point, then press Enter once.
11. Click Heading 1 from the styles group located under the home tab
12. Type the text “Copy and Paste in a Document.” Press enter once after the heading, and type the following text:
13. After you complete the above paragraph select the entire paragraph, including the heading, “Copy and Paste in a Document.”
14. Cut the selected text, then Position the Insertion Point below the last paragraph (Hint: Ctrl + End) then press Enter; Paste the text.
15. Bold Copy and Paste; use the Format Painter to apply Bold to the text.
16. Apply Italic to the another paragraph.
17. Zoom in the document by 150%.
18. Save the document.
3. Make a report.

Laboratory work 2

1. Find information about Different Formats in Microsoft Word.
2. Do the exercise:
1. Create a Blank document
2. Save the document as WordLabFour.docx in Documents > MSWordLabs folder
3. Create a Title Page centered in the middle of the page with the following items:
4. a title “Apply Different Formats to Word Documents”
5. your college name

6. your First name and Last name
7. today’s date that updates automatically in the following format: Monday, June 15, 2020
8. Insert a Page Break after today’s date
9. Type =rand() then press enter; this function should create five paragraphs
10. Merge the first and second paragraphs; merge the last two paragraphs
11. Indent all paragraphs ½ inch from the margin using first line indent
12. Change the page margin to Moderate
13. Insert a page number Brackets 2 at the bottom of the page
14. Insert a Dropped Drop Cap for the first paragraph and the last paragraph
15. Insert the following Subtitles for each paragraph

- Professional documents
- Effectiveness of themes
- Additional formats

16. Apply the Retrospect theme to the document
17. Insert a Page Border Orange Accent 2
18. Save document
19. Create a new cover letter from a template
20. Save the document as WordLabSix.docx in Documents > MSWordLabs folder
21. Complete the cover letter based on a job opportunity you found online or in the local newspaper. (Hint: you need to search for a job of your choice)
22. Insert a Next Page Section Break at the end of the cover letter
23. Create a new Basic resume from a template
24. Copy the entire resume, then paste it with Use the Destination theme below the section break in WordLabThree.docx
25. Remove any extra pages and adjust the contents of the resume (You should have a total of 2 Pages only)
26. Change the resume theme to Ion
27. Complete the resume (Fill in the blank fields)
28. Save document

3. Make a report.

Laboratory work 3

1. Find common information about Microsoft Excel application.
2. Do the exercise (repeat the following steps):

This theory explains what a cell address is, how to make absolute and relative references in Excel, how to reference a cell in another sheet.

What is a cell reference in Excel?

A cell reference or cell address is a combination of a column letter and a row number that identifies a cell on a worksheet. For example, A1 refers to the cell at the

intersection of column A and row 1; B2 refers to the second cell in column B, and so on (figure 6).

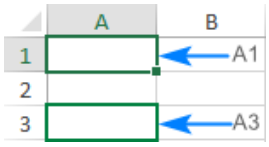


Figure 6 – Refers to the second cell

When used in a formula, cell references help Excel find the values the formula should calculate.

For instance, to pull the value of A1 to another cell, you use this simple formula: =A1. To add up the values in cells A1 and A2, you use this one: =A1+A2. What is a range reference in Excel? In Microsoft Excel, a range is a block of two or more cells. A range reference is represented by the address of the upper left cell and the lower right cell separated with a colon.

For example, the range A1:C2 includes 6 cells from A1 through C2. How to create a reference in Excel

To make a cell reference on the same sheet, this is what you need to do:

- Click the cell in which you want to enter the formula.
- Type the equal sign (=).
- Do one of the following:
- Type the reference directly in the cell or in the formula bar, or
- Click the cell you want to refer to.
- Type the rest of the formula and press the Enter key to complete it.

For example, to add up the values in cells A1 and A2, you type the equal sign, click A1, type the plus sign, click A2 and press Enter (figure 7):

Figure 7 – Adding the values in cells A1 and A2

To create a range reference, select a range of cells on the worksheet.

For example, to add up the values in cells A1, A2 and A3, type the equal sign followed by the name of the SUM function and the opening parenthesis, select the cells from A1 through A3, type the closing parenthesis, and press Enter (figure 8):

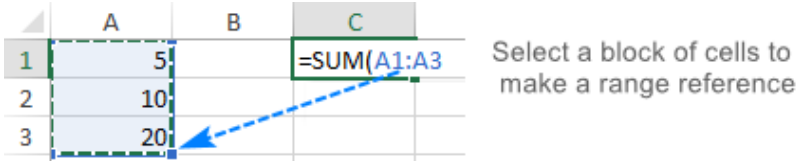


Figure 8 – SUM function

To refer to the whole row or entire column, click the row number or the column letter, respectively.

For instance, to add up all the cells in row 1, start typing the SUM function, and then click the header of the first row to include the row reference in your formula (figure 9):

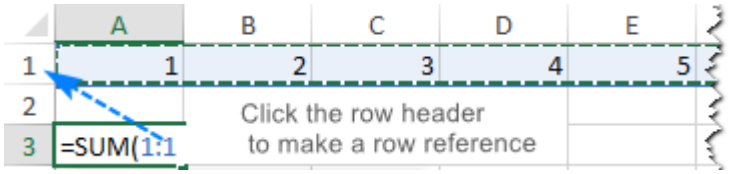


Figure 9 – Row reference

How to change Excel cell reference in a formula

To change a cell address in an existing formula, carry out these steps:

- Click on the cell that contains the formula and press F2 to enter the Edit mode, or double-click the cell. This will highlight each cell/range referenced by the formula with a different color.

- To change a cell address, do any of the following:

- Select the reference in the formula and type a new one.

- Select the reference in the formula, and then select another cell or range on the sheet (figure 10):

Figure 10 – Selection of another cell to changing reference

- To include more or fewer cells in a reference, drag the color-coded border of the cell or range (figure 11):

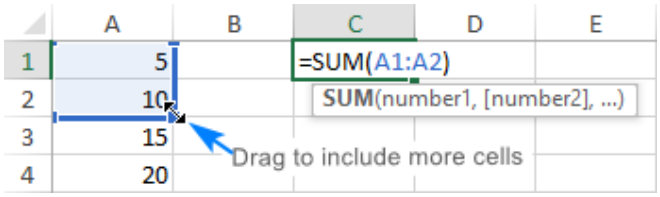


Figure 11 – Including more or fewer cells in a reference

- Press the Enter key.

How to cross reference in Excel

To refer to cells in another worksheet or a different Excel file, you must identify not only the target cell(s), but also the sheet and workbook where the cells are located. This can be done by using so-called external cell reference.

How to reference another sheet in Excel

To refer to a cell or a range of cells in another worksheet, type the name of the target worksheet followed by an exclamation point (!) before the cell or range address.

For example, here's how you can refer to cell A1 on Sheet2 in the same workbook:

=Sheet2!A1

If the name of the worksheet contains spaces or nonalphabetical characters, you must enclose the name within single quotation marks, e.g.:

='Target sheet'!A1

To prevent possible mistakes, you can get Excel to create an external reference for you automatically. Here's how:

- Start typing a formula in a cell.
- Click the sheet tab you want to cross-reference and select the cell or range of cells.
- Finish typing your formula and press Enter.

To refer to a cell or range of cells in a different Excel file, you need to include the workbook name in square brackets, followed by the sheet name, exclamation point, and the cell or a range address. For example:

=[Book1.xlsx]Sheet1!A1

If the file or sheet name contains non-alphabetical characters, be sure to enclose the path in single quotation marks, e.g.

='[Target file.xlsx]Sheet1'!A1

As with a reference to another sheet, you don't have to type the path manually. A faster way is to switch to the other workbook and select a cell or a range of cells there.

Relative, absolute and mixed cell references

There are three types of cell references in Excel: relative, absolute and mixed. When writing a formula for a single cell, you can go with any type. But if you intend to copy your formula to other cells, it is important that you use an appropriate address type because relative and absolute cell references behave differently when filled to other cells.

A relative reference is the one without the $ sign in the row and column coordinates, like A1 or A1:B10. By default, all cell addresses in Excel are relative.

When moved or copied across multiple cells, relative references change based on the relative position of rows and columns. So, if you want to repeat the same calculation across several columns or rows, you need to use relative cell references. For example, to multiply numbers in column A by 5, you enter this formula in B2: =A2*5. When copied from row 2 to row 3, the formula will change to =A3*5 (figure 12):

	A	B	C
1	**Data**	**Result**	**Formula**
2	1	5	=A2*5
3	2	10	=A3*5
4	3	15	=A4*5

Figure 12 – Changing formula during copying

An absolute reference is the one with the dollar sign ($) in the row or column coordinates, like $A$1 or $A$1:$B$10.

An absolute cell reference remains unchanged when filling other cells with the same formula. Absolute addresses are especially useful when you want to perform multiple calculations with a value in a specific cell or when you need to copy a formula to other cells without changing references. For example, to multiply the numbers in column A by the number in B2, you input the following formula in row 2, and then copy the formula down the column by dragging the fill handle: =A2*$B$2

The relative reference (A2) will change based on a relative position of a row where the formula is copied, while the absolute reference ($B$2) will always be locked on the same cell (figure 13):

	A	B	C	D
1	**Number**	**Multiply by**	**Result**	**Formula**
2	10	10	100	=A2*$B$2
3	20		200	=A3*$B$2
4	30		300	=A3*$B$2

Figure 13 – Absolute reference demonstration

Mixed cell reference

A mixed reference contains one relative and one absolute coordinate, like $A1 or A$1.

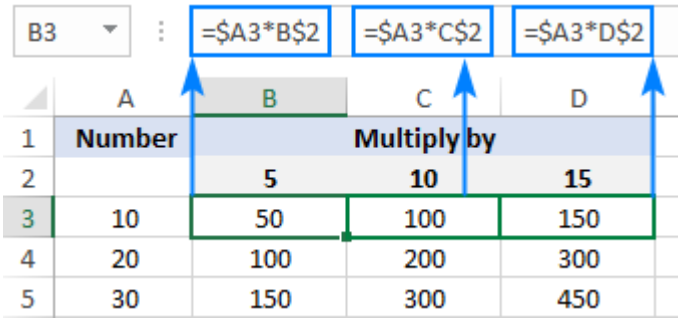
There may be many situations when only one coordinate, column or row, should be fixed.

For example, to multiply a column of numbers (column A) by 3 different numbers (B2, C2 and D2), you put the following formula in B3, and then copy it down and to the right:

=$A3*B$2

In $A3, you lock the column coordinate because the formula should always multiply the original numbers in column A. The row coordinate is relative since it needs to change for other rows. In B$2, you lock the row coordinate to tell Excel always to pick the multiplier in row 2. The column coordinate is relative because the multipliers are in 3 different columns and the formula should adjust accordingly.

As the result, all the calculations are performed with a single formula, which changes properly for each row and column where it is copied (figure 14):



	A	B	C	D
1	Number	Multiply by		
2		5	10	15
3	10	50	100	150
4	20	100	200	300
5	30	150	300	450

Figure 14 – Mixed cell reference demonstration

How to switch between different reference types. To switch from a relative reference to absolute and vice versa, you can either type or delete the $ sign manually, or use the F4 shortcut:

- Double-click the cell that contains the formula.
- Select the reference you want to change.
- Press F4 to toggle between the four reference types.

Repeatedly hitting the F4 key switches the references in this order: A1 > $A$1 > A$1 > $A1. As the result, the references in all or selected formulas will be updated.

3. Make a report.

Laboratory work 4

1. Find information about charts in Microsoft Excel.
2. Do the exercise:

- create a multiplication table;
- create charts of the following types using data from the multiplication table:

Excel has several types of charts, allowing you to choose the one that best fits your data. To use charts effectively, you'll need to understand how different charts are used. Click the arrows in the slideshow below to learn more about the types of charts in Excel (figure 15).

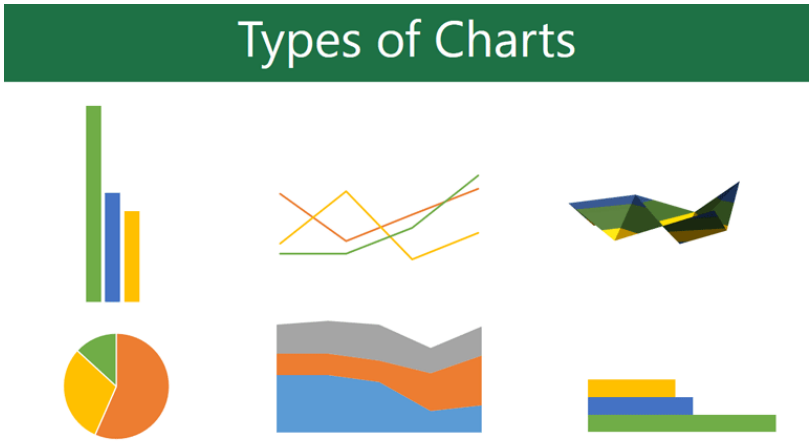


Figure 15 – Types of Charts

Excel has a variety of chart types, each with its own advantages. Click the arrows to see some of the different types of charts available in Excel (figures 16 – 21):

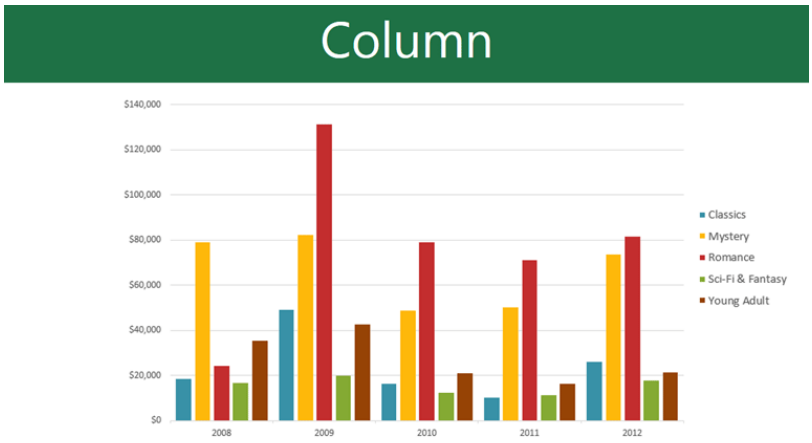


Figure 16 – Column Chart

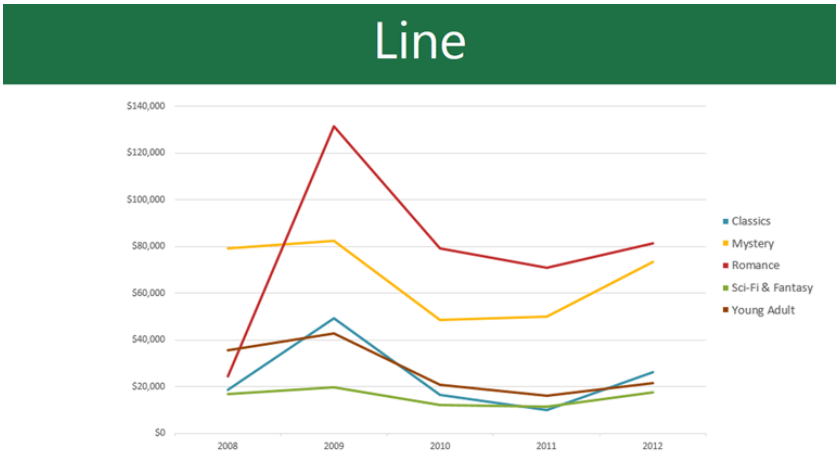


Figure 17 – Line Chart

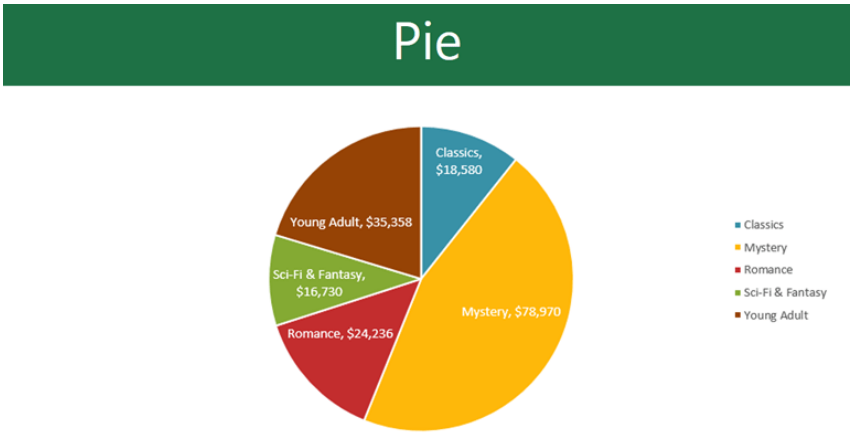


Figure 18 – Pie Chart

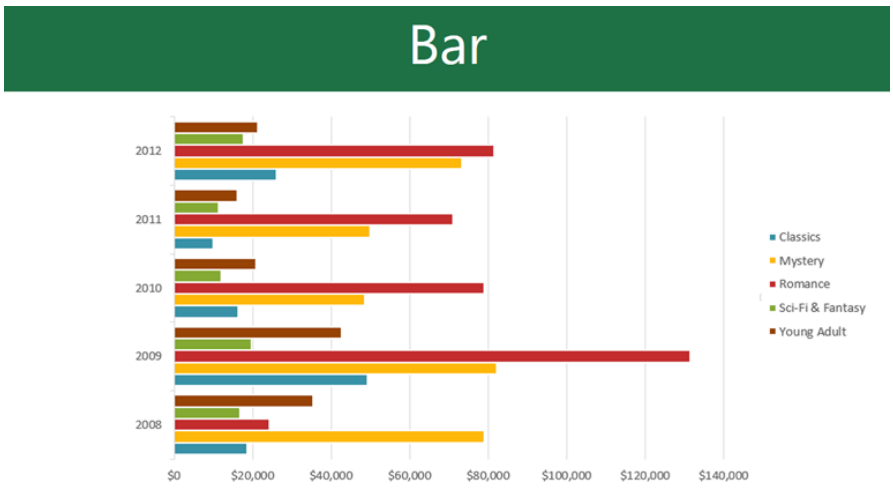


Figure 19 – Bar Chart

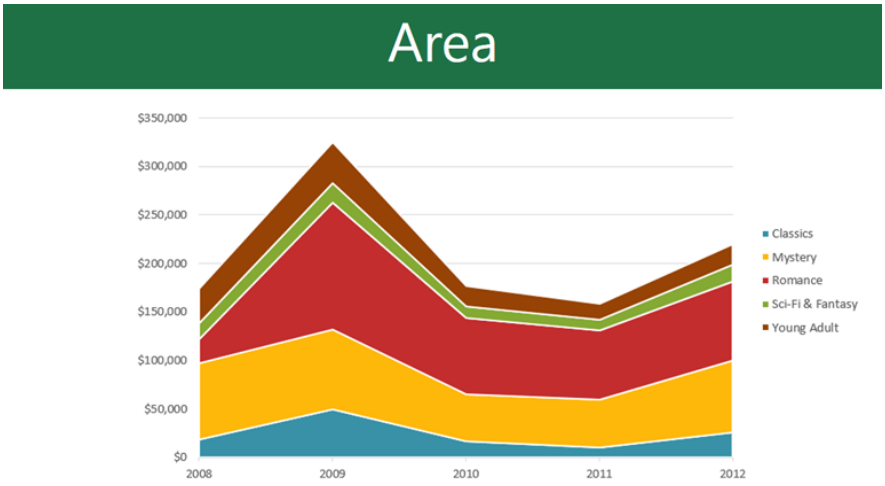


Figure 20 – Area Chart

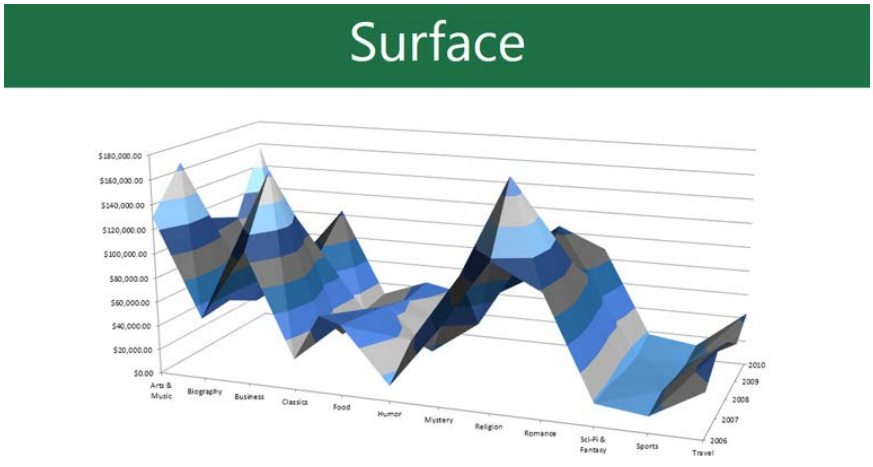


Figure 21 – Surface Chart

In addition to chart types, you'll need to understand how to read a chart. Charts contain several elements, or parts, that can help you interpret data (example on figure 22). Click the buttons in the interactive below to learn about the different parts of a chart

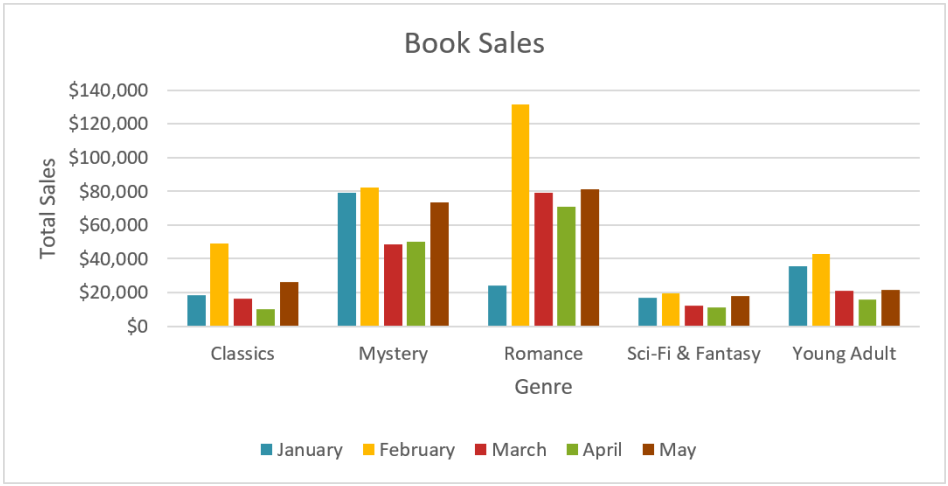


Figure 22 – Example of grouped data

To insert a chart:

- Select the cells you want to chart, including the column titles and row labels (figure 23). These cells will be the source data for the chart. In our example, we'll select cells A1:F6:

	A	B	C	D	E	F	G
1	Genre	January	February	March	April	May	
2	Classics	$18,580	$49,225	$16,326	$10,017	$26,134	
3	Mystery	$78,970	$82,262	$48,640	$49,985	$73,428	
4	Romance	$24,236	$131,390	$79,022	$71,009	$81,474	
5	Sci-Fi & Fantasy	$16,730	$19,730	$12,109	$11,355	$17,686	
6	Young Adult	$35,358	$42,685	$20,893	$16,065	$21,388	
7							
8							

Figure 23 – Selection of cells

- From the Insert tab, click the desired Chart command (figure 24). In our example, we'll select Column (figure 25).

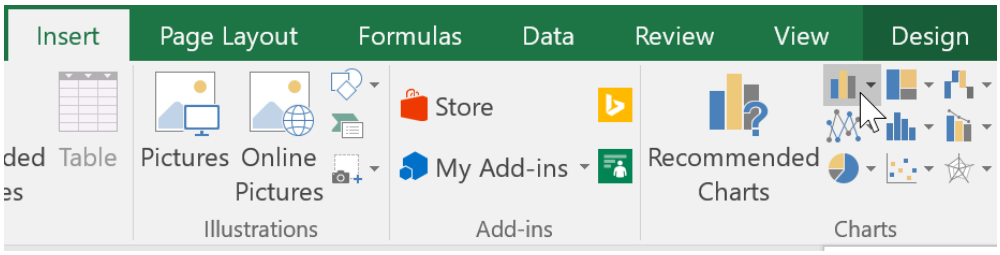


Figure 24 – Selection of cells

- Choose the desired chart type from the drop-down menu.

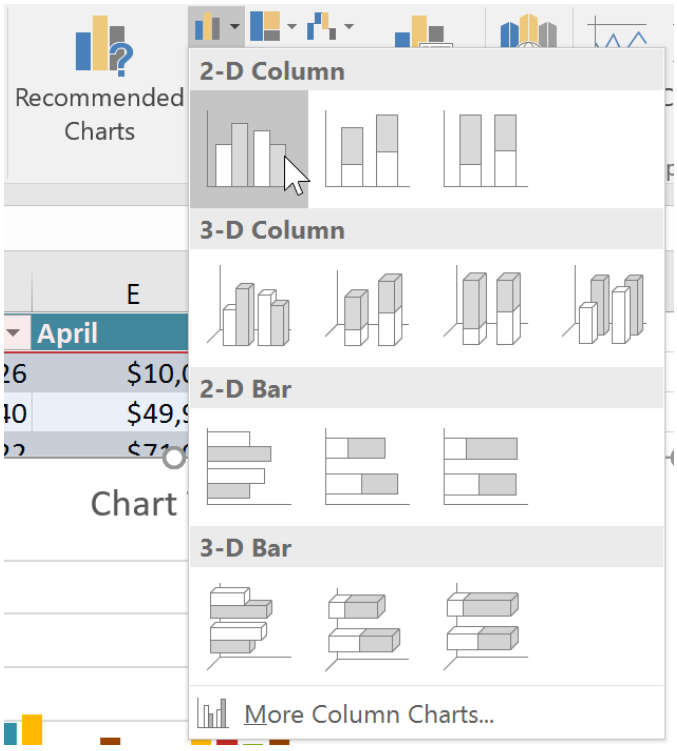


Figure 25 – Column chart selection

- The Selected chart will be inserted into the worksheet (figure 26):

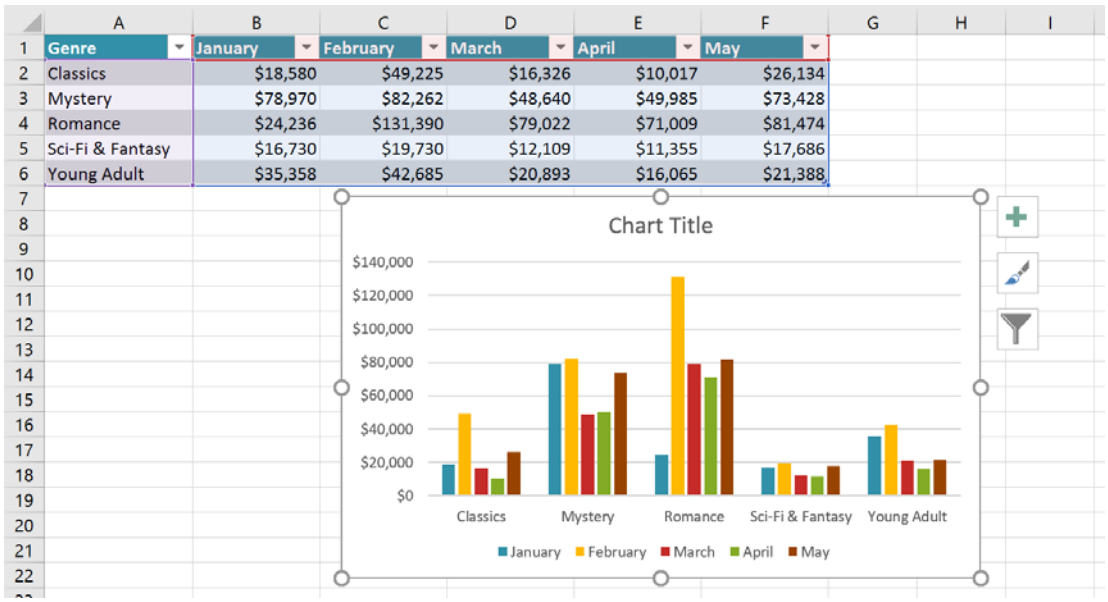


Figure 26 – Column chart selection

If you're not sure which type of chart to use, the Recommended Charts command will suggest several charts based on the source data (figure 27).

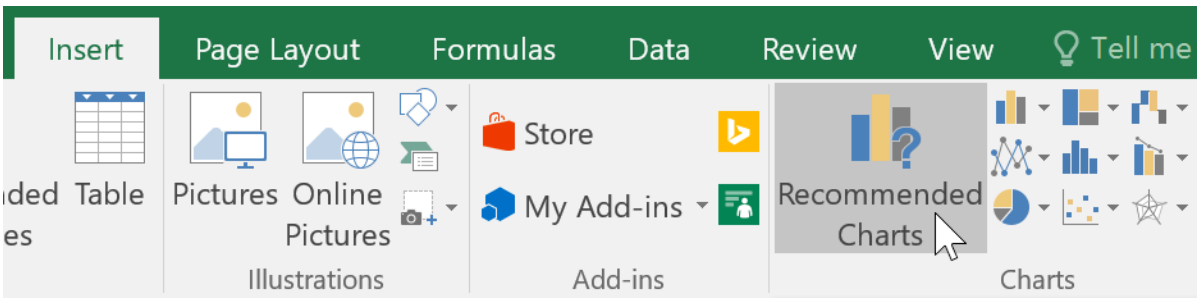


Figure 27 – Recommended Chart command

Chart and layout style.

After inserting a chart, there are several things you may want to change about the way your data is displayed. It's easy to edit a chart's layout and style from the Design tab.

Excel allows you to add chart elements–including chart titles, legends, and data labels–to make your chart easier to read. To add a chart element, click the Add Chart Element command on the Design tab, then choose the desired element from the drop-down menu (figure 28):

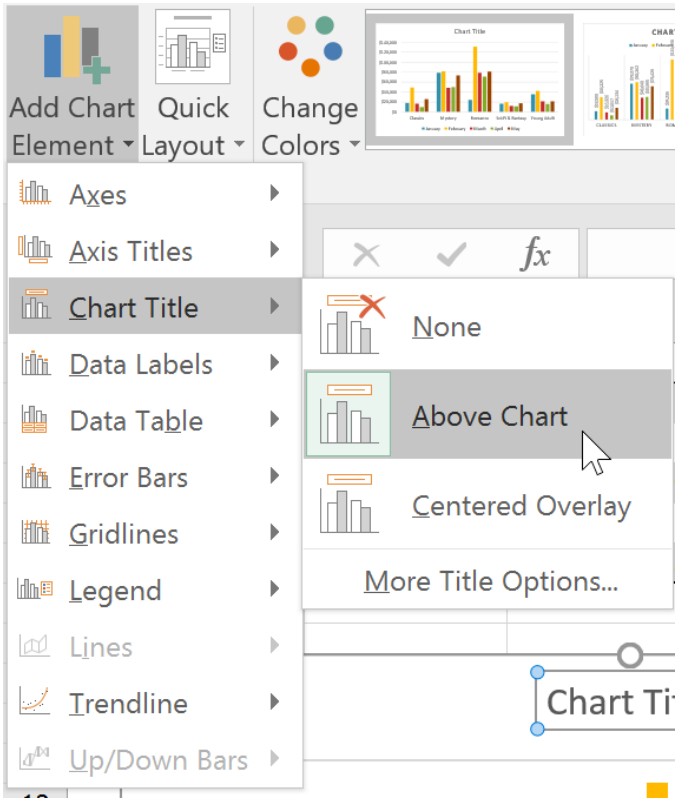


Figure 28 – Adding of desired Chart element

To edit a chart element, like a chart title, simply double-click the placeholder and begin typing (figure 29):

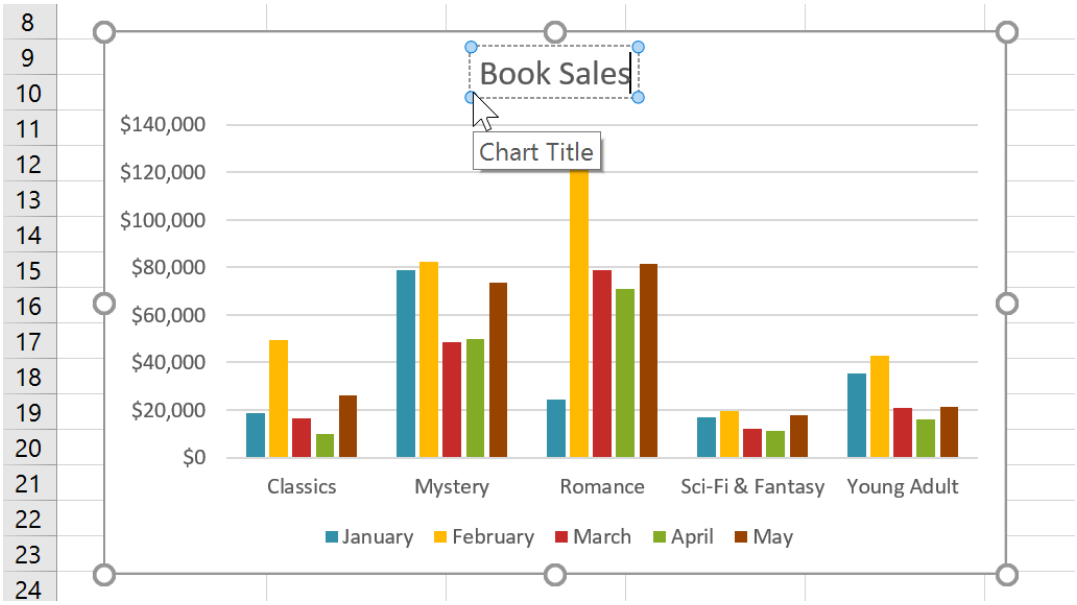


Figure 28 – Adding of desired Chart element

If you don't want to add chart elements individually, you can use one of Excel's predefined layouts. Simply click the Quick Layout command, then choose the desired layout from the drop-down menu (figure 29):

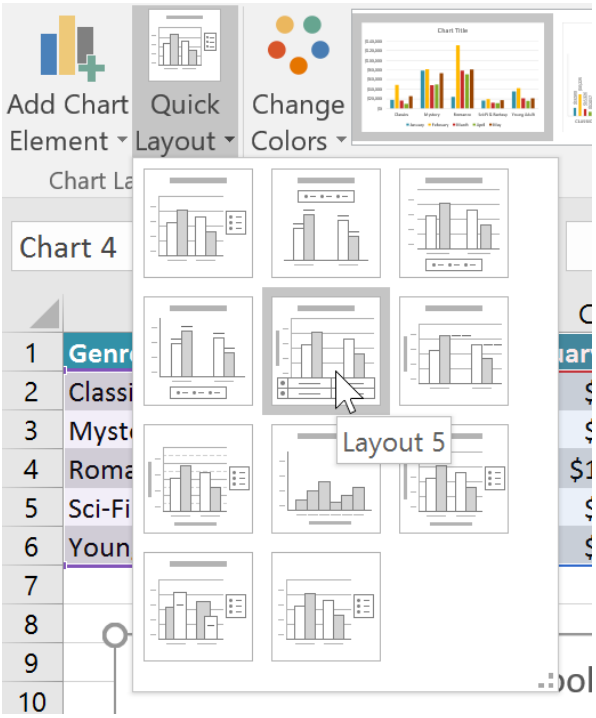


Figure 29 – Choice of desired Layout

Excel also includes several chart styles, which allow you to quickly modify the look and feel of your chart. To change the chart style, select the desired style from the Chart styles group. You can also click the drop-down arrow on the right to see more styles (figure 30):

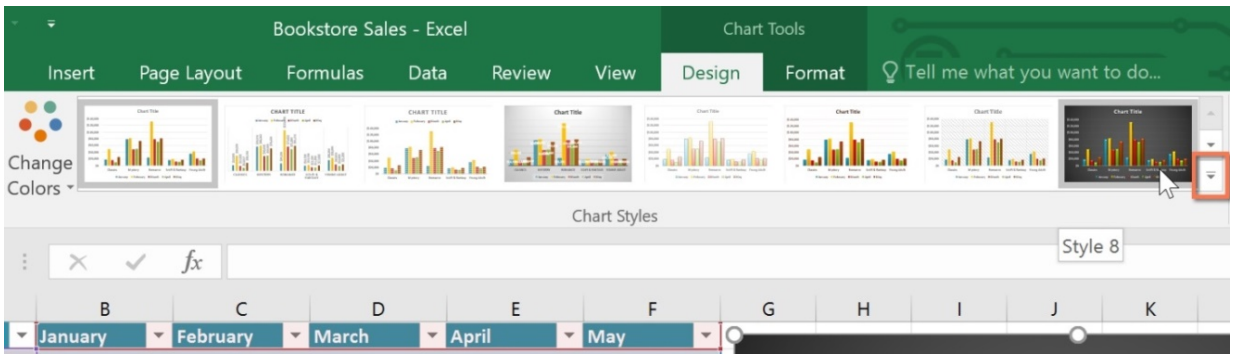


Figure 30 – Choice of styles

You can also use the chart formatting shortcut buttons to quickly add chart elements, change the chart style, and filter chart data.

Other chart options.

There are many other ways to customize and organize your charts. For example, Excel allows you to rearrange a chart's data, change the chart type, and even move the chart to a different location in a workbook.

To switch row and column data:

Sometimes you may want to change the way charts group your data. For example, in the chart below Book Sales data is grouped by genre, with columns for each month. However, we could switch the rows and columns so the chart will group the data by month, with columns for each genre. In both cases, the chart contains the same data – it's just organized differently (figure 31).

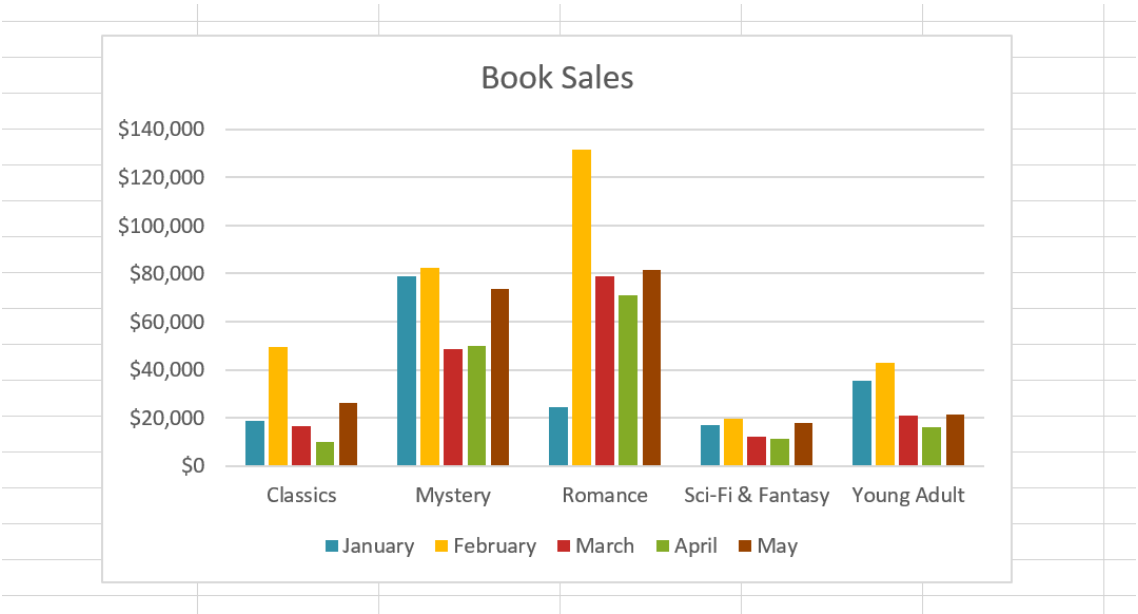


Figure 31 – Different organization of the same data

- Select the chart you want to modify.
- From the Design tab, select the Switch Row/Column command (figure 32).

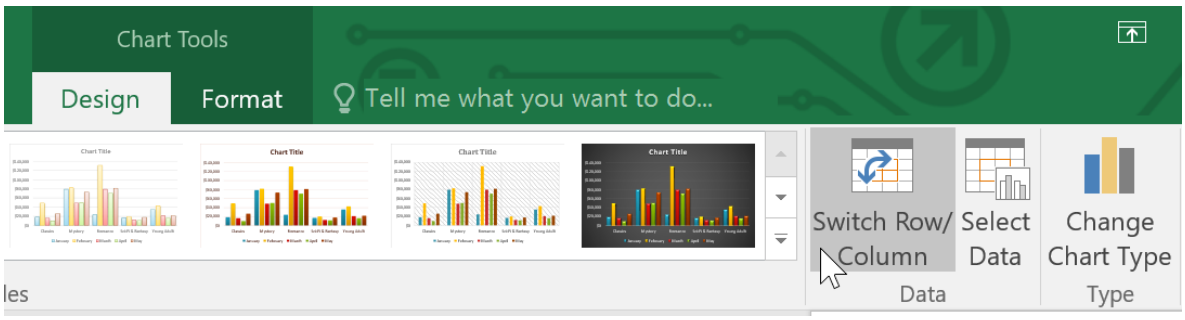


Figure 32 – Switch Row/Column command

- The rows and columns will be switched. In our example, the data is now grouped by month, with columns for each genre (figure 33):

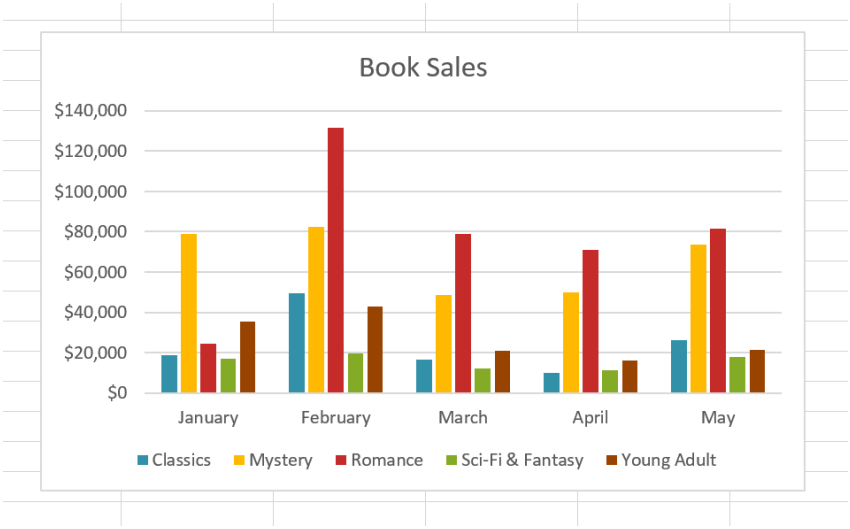


Figure 33 – Grouping data (by month in this example)

To change the chart type:

If you find that your data isn't working well in a certain chart, it's easy to switch to a new chart type. In our example, we'll change our chart from a column chart to a line chart.

- From the Design tab, click the Change Chart Type command (figure 34):

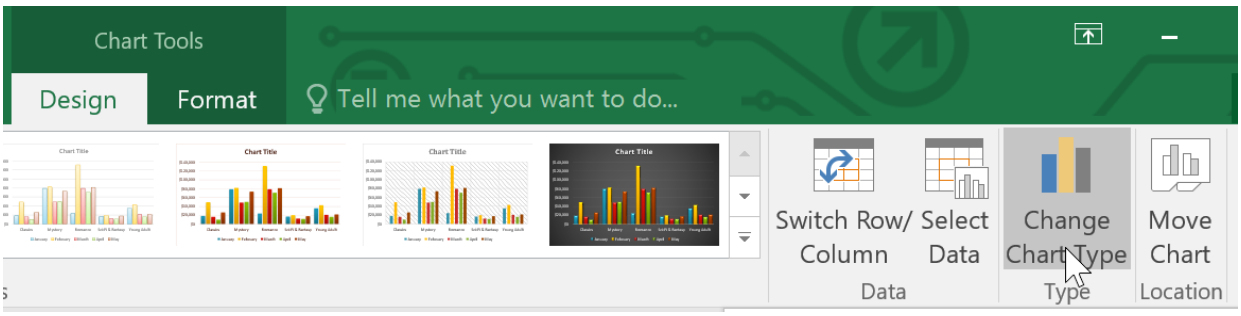


Figure 34 – Change Chart Type command

-The Change Chart Type dialog box will appear. Select a new chart type and layout, then click OK. In our example, we'll choose a Line chart (figure 35):

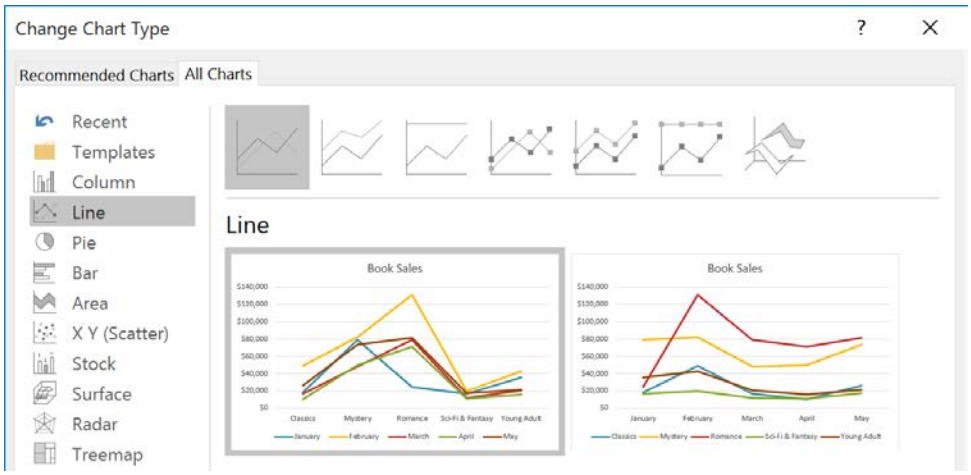


Figure 35 – Line chart

- The selected chart type will appear. In our example, the line chart makes it easier to see trends in sales data over time (figure 36):

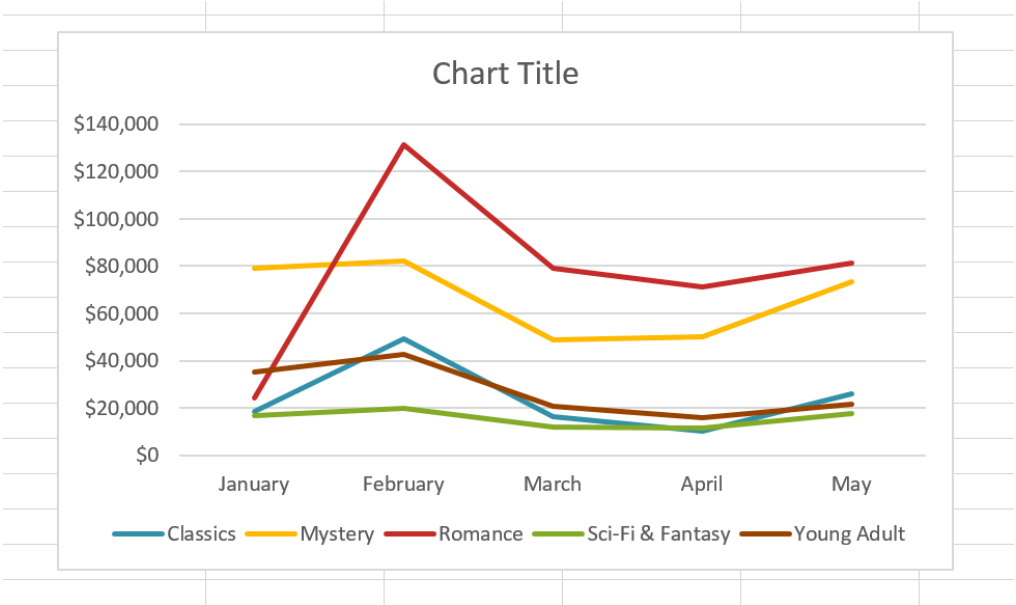


Figure 36 – Trends in sales data over time

To move a chart:

Whenever you insert a new chart, it will appear as an object on the same worksheet that contains its source data. You can easily move the chart to a new worksheet to help keep your data organized.

- Select the chart you want to move.
- Click the Design tab, then select the Move Chart command (figure 37):

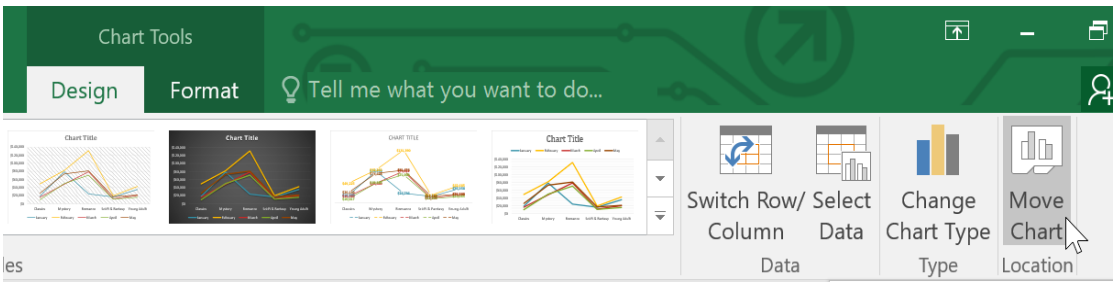


Figure 37 –Move Chart command in the Design tab

- The Move Chart dialog (figure 38) box will appear. Select the desired location for the chart. In our example, we'll choose to move it to a New sheet, which will create a new worksheet.

- Click OK:

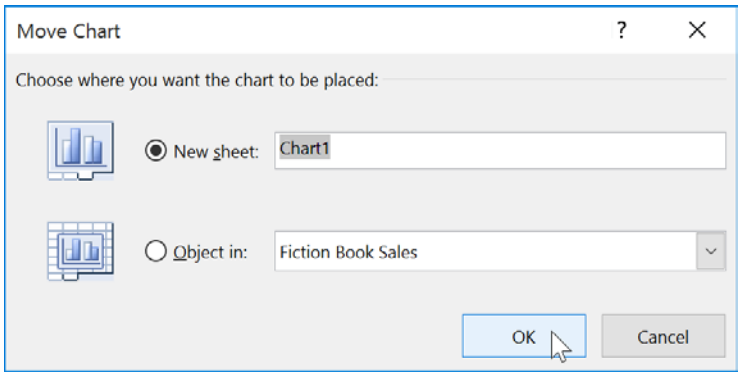


Figure 38 – Move Chart dialog

- The chart will appear in the selected location. In our example, the chart now appears on a new worksheet (figure 39):

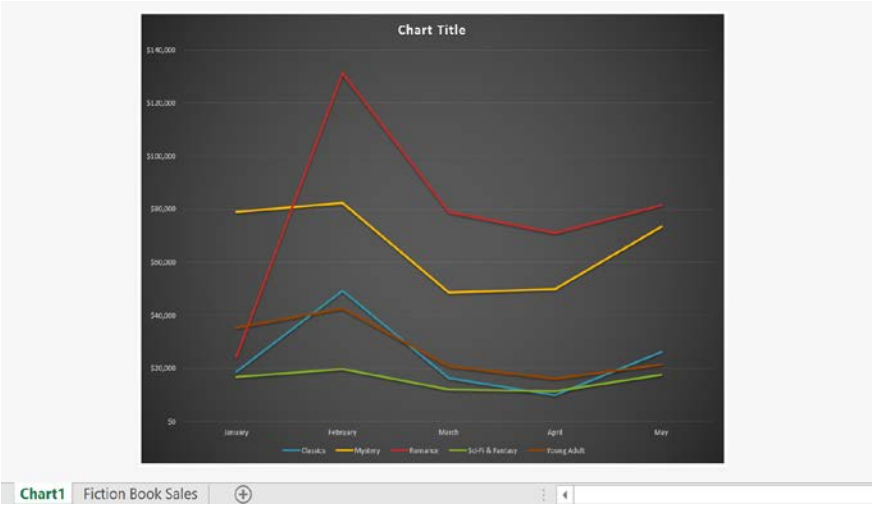


Figure 39 – Appearing the chart appears on a new worksheet

3. Make a report.

Laboratory work 5

1. Find common information about Microsoft Powerpoint.

2. Do the exercise:

1. Create a Blank presentation.
2. Add information about yourself.
3. Change the document theme to Trek. Note that the Trek theme uses all capital letters for the title text. On the title slide, use your name in place of Student Name and bold and italicize your name.
4. Increase the title text font size to 44 point.
5. Then change paragraphs three and four (Start and end your day with water; Do not wait until you are thirsty) to third-level paragraphs.
6. Check the spelling, and then display the revised presentation in grayscale.
7. Add the Print button to the Quick Access Toolbar and then click this button to print the slides
8. Save the document.
9. Make ten slides.
10. Change the document theme from Metro.
11. Move Slide 2 to the end of the presentation so that it becomes the new Slide 10.
12. Use the spell checker to correct the misspellings. Analyze the slides for other word usage errors that the spell checker did not find (if it found them of course).
13. 5. On Slide 2, increase the Slide 2 title font size to 40. Make the indent levels for paragraphs 2, 4, and 6 the same level.
14. On Slide 3, change the font color of the number, 760, to green and the number, 620, to red.

3. Make a report.

Laboratory work 6

1. Read intro materials.

Searching the internet can be a frustrating business. You enter a word or a phrase into a search engine and up comes a stack of irrelevant information.

What you need is the ability to refine your search to get exactly what you want.

We explore seven steps that you can take to pinpoint specific information online.

- Vary Your Search Engine

Search engines sort through about 625 million active websites to provide you with content. You may favor one, but don't let habit restrict you. No search engine is perfect, and they all have different blind spots.

The most widely used search engines are Google®, Bing® and Yahoo®.

Google usually returns the greatest variety of results, and has by far the largest catalog of pages.

Bing, however, has more extensive autocomplete results (where the search engine tries to narrow the search for you).

Yahoo offers search as part of a wider range of services that includes news and shopping. Other engines such as DuckDuckGo® and Dogpile® also have their devotees.

- Use Specific Keywords

Keywords are the terms that you use to find content on the internet. Making your keywords as specific as possible will help your search engine to track down the information that you want.

Say, for example, that you want to find a local supplier that can design an exhibition stand for your company. If you type stand design into your search engine, the results will include many pages about other types of stand, whereas typing exhibition stand designer will return a more concise range of companies.

You can further refine your search by including other specific keywords. If you add your location, for example, you'll likely find someone local.

- Simplify Your Search Terms

Some engines include stop words in their searches. These are frequently used words such as prepositions (in, of, on), conjunctions (and, but) and articles (a, the), which mean that you'll end up with more pages in your search results than you need.

So, it's usually best to eliminate stop words from your internet searches. The main exception is if you're looking for a specific title or name that includes them.

Also, use the simplest form of the keywords that you're looking for, by avoiding plurals and verb forms with suffixes such as -ing, -s or -ed. For example, you would improve the quality of your search results by searching for service rather than services, or finance rather than financed or financing.

- Use Quotation Marks

Enclosing a search term within quotation marks prompts the search engine to search for that specific word or phrase.

If the term is a single word, using quotation marks will cut out stemmed variations of it. For example, if you search for the word director, you'll likely receive a lot of results for direct, direction, directions, and so on, too. Typing "director" (with quotation marks), however, will ensure that you only get results for that stem word.

Tip:

Some search engines allow you to search for specific words by preceding them with the + symbol. Google no longer uses this function, but Yahoo, for example, does.

If the search term is a phrase, your search will be for that specific phrase, rather than for all the component words as individual items. So, for example, if you search for the phrase director of human resources, without quotation marks, your search will return results based on all of the words in the phrase (except of, which is a stop word.) Surrounding the term with quotation marks, however, will generate results that feature this specific term.

- Remove Unhelpful Words

Inserting a hyphen/small dash/minus sign immediately before a word excludes it from a search.

So imagine, for example, that you're looking to find out more about marketing. However, you want to concentrate on traditional marketing techniques, whereas the internet appears to be full of references to digital and social media marketing, all of which are appearing in your search.

Typing in marketing -digital will exclude digital from the search, making it easier for you to find the information you're looking for.

2. Try to find information about your city or country on the Internet using steps and tips from the material before:

3. Make a report.

Laboratory work 7

1. Read intro materials:

Algorithms.

In mathematics and computer science, an algorithm is a sequence of instructions, typically to solve a class of problems or perform a computation. Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.

As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.

The concept of algorithm has existed since antiquity. Arithmetic algorithms, such as a division algorithm, was used by ancient Babylonian mathematicians and Egyptian mathematicians. Greek mathematicians later used algorithms in the sieve of Eratosthenes for finding prime numbers (is a natural number greater than 1 that has no positive divisors other than 1 and itself), and the Euclidean algorithm for finding the greatest common divisor of two numbers. Arabic mathematicians such as Al-Kindi in the 9th century used cryptographic algorithms for code-breaking, based on frequency analysis.

The word algorithm itself is derived from the 9th-century Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī, Latinized Algoritmi. A partial formalization of what would become the modern concept of algorithm began with attempts to solve the Entscheidungsproblem (decision problem) posed by David Hilbert in 1928. Later formalizations were framed as attempts to define "effective calculability" or "effective method". Those formalizations included the Gödel–Herbrand–Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of

1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines of 1936–37 and 1939.

Algorithm design refers to a method or mathematical process for problem-solving and engineering algorithms. The design of algorithms is part of many solution theories of operation research, such as dynamic programming and divide-and-conquer. Techniques for designing and implementing algorithm designs are also called algorithm design patterns such as the template method pattern and decorator pattern.

Typical steps in the development of algorithms:

- Problem definition.
- Development of a model.
- Specification of the algorithm.
- Designing an algorithm.
- Checking the correctness of the algorithm.
- Analysis of algorithm.
- Implementation of algorithm.
- Program testing.
- Documentation preparation.
- Algorithm example.

An animation of the quicksort algorithm sorting an array of randomized values. The red bars mark the pivot element; at the start of the animation, the element farthest to the right-hand side is chosen as the pivot.

One of the simplest algorithms is to find the largest number in a list of numbers of random order. Finding the solution requires looking at every number in the list. From this follows a simple algorithm, which can be stated in a high-level description in English prose, as:

High-level description:

If there are no numbers in the set then there is no highest number.

Assume the first number in the set is the largest number in the set.

For each remaining number in the set: if this number is larger than the current largest number, consider this number to be the largest number in the set.

When there are no numbers left in the set to iterate over, consider the current largest number to be the largest number of the set.

(Quasi-)formal description: Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

Algorithm LargestNumber
Input: A list of numbers L.
Output: The largest number in the list L.
if L.size = 0 return null
largest ← L[0]
for each item in L, do
if item > largest, then
largest ← item

return largest

Logical

An algorithm may be viewed as controlled logical deduction. This notion may be expressed as: Algorithm = logic + control. The logic component expresses the axioms that may be used in the computation and the control component determines the way in which deduction is applied to the axioms. This is the basis for the logic programming paradigm. In pure logic programming languages, the control component is fixed and algorithms are specified by supplying only the logic component. The appeal of this approach is the elegant semantics: a change in the axioms produces a well-defined change in the algorithm.

Serial, parallel or distributed

Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time. Those computers are sometimes called serial computers. An algorithm designed for such an environment is called a serial algorithm, as opposed to parallel algorithms or distributed algorithms. Parallel algorithms take advantage of computer architectures where several processors can work on a problem at the same time, whereas distributed algorithms utilize multiple machines connected with a computer network. Parallel or distributed algorithms divide the problem into more symmetrical or asymmetrical subproblems and collect the results back together. The resource consumption in such algorithms is not only processor cycles on each processor but also the communication overhead between the processors. Some sorting algorithms can be parallelized efficiently, but their communication overhead is expensive. Iterative algorithms are generally parallelizable. Some problems have no parallel algorithms and are called inherently serial problems.

Deterministic or non-deterministic

Deterministic algorithms solve the problem with exact decision at every step of the algorithm whereas non-deterministic algorithms solve problems via guessing although typical guesses are made more accurate through the use of heuristics.

Exact or approximate

While many algorithms reach an exact solution, approximation algorithms seek an approximation that is closer to the true solution. The approximation can be reached by either using a deterministic or a random strategy. Such algorithms have practical value for many hard problems. One of the examples of an approximate algorithm is the Knapsack problem, where there is a set of given items. Its goal is to pack the knapsack to get the maximum total value. Each item has some weight and some value. Total weight that can be carried is no more than some fixed number X. So, the solution must consider weights of items as well as their value.

Quantum algorithm

They run on a realistic model of quantum computation. The term is usually used for those algorithms which seem inherently quantum, or use some essential feature of Quantum computing such as quantum superposition or quantum entanglement.

By design paradigm

Another way of classifying algorithms is by their design methodology or paradigm. There is a certain number of paradigms, each different from the other. Furthermore, each of these categories includes many different types of algorithms. Some common paradigms are:

Brute-force or exhaustive search

This is the naive method of trying every possible solution to see which is best.

Divide and conquer

A divide and conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments. A simpler variant of divide and conquer is called a decrease and conquer algorithm, that solves an identical subproblem and uses the solution of this subproblem to solve the bigger problem. Divide and conquer divides the problem into multiple subproblems and so the conquer stage is more complex than decrease and conquer algorithms. An example of a decrease and conquer algorithm is the binary search algorithm.

Search and enumeration

Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.

Randomized algorithm

Such algorithms make some choices randomly (or pseudo-randomly). They can be very useful in finding approximate solutions for problems where finding exact solutions can be impractical (see heuristic method below). For some of these problems, it is known that the fastest approximations must involve some randomness. Whether randomized algorithms with polynomial time complexity can be the fastest algorithms for some problems is an open question known as the P versus NP problem. There are two large classes of such algorithms:

Monte Carlo algorithms return a correct answer with high-probability. E.g. RP is the subclass of these that run in polynomial time.

Las Vegas algorithms always return the correct answer, but their running time is only probabilistically bound, e.g. ZPP.

Reduction of complexity

This technique involves solving a difficult problem by transforming it into a better-known problem for which we have (hopefully) asymptotically optimal algorithms. The goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithm's. For example, one selection algorithm for finding the median in an unsorted list involves first sorting the list (the expensive portion) and then pulling out the middle element in the sorted list (the cheap portion). This technique is also known as transform and conquer.

Back tracking

In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution.

Optimization problems

For optimization problems there is a more specific classification of algorithms; an algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

Linear programming

When searching for optimal solutions to a linear function bound to linear equality and inequality constraints, the constraints of the problem can be used directly in producing the optimal solutions. There are algorithms that can solve any problem in this category, such as the popular simplex algorithm. Problems that can be solved with linear programming include the maximum flow problem for directed graphs. If a problem additionally requires that one or more of the unknowns must be an integer then it is classified in integer programming. A linear programming algorithm can solve such a problem if it can be proved that all restrictions for integer values are superficial, i.e., the solutions satisfy these restrictions anyway. In the general case, a specialized algorithm or an algorithm that finds approximate solutions is used, depending on the difficulty of the problem.

Dynamic programming

When a problem shows optimal substructures–meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems–and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called dynamic programming avoids recomputing solutions that have already been computed. For example, Floyd–Warshall algorithm, the shortest path to a goal from a vertex in a weighted graph can be found by using the shortest path to the goal from all adjacent vertices. Dynamic programming and memoization go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming. The difference between dynamic programming and straightforward recursion is in caching or memoization of recursive calls. When subproblems are independent and there is no repetition, memoization does not help; hence dynamic programming is not a solution for all complex problems. By using memoization or maintaining a table of subproblems already solved, dynamic programming reduces the exponential nature of many problems to polynomial complexity.

The greedy method

A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution, which may be given or have been constructed in some way, and improve it by making small modifications. For some problems they can find the optimal solution while for others they stop at local optima, that is, at solutions that cannot be improved by the algorithm but are not optimum. The most popular use of greedy algorithms is for finding the minimal spanning tree

where finding the optimal solution is possible with this method. Huffman Tree, Kruskal, Prim, Sollin are greedy algorithms that can solve this optimization problem.

The heuristic method

In optimization problems, heuristic algorithms can be used to find a solution close to the optimal solution in cases where finding the optimal solution is impractical. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Their merit is that they can find a solution very close to the optimal solution in a relatively short time. Such algorithms include local search, tabu search, simulated annealing, and genetic algorithms. Some of them, like simulated annealing, are non-deterministic algorithms while others, like tabu search, are deterministic. When a bound on the error of the non-optimal solution is known, the algorithm is further categorized as an approximation algorithm.

By field of study

See also: List of algorithms

Every field of science has its own problems and needs efficient algorithms. Related problems in one field are often studied together. Some example classes are search algorithms, sorting algorithms, merge algorithms, numerical algorithms, graph algorithms, string algorithms, computational geometric algorithms, combinatorial algorithms, medical algorithms, machine learning, cryptography, data compression algorithms and parsing techniques.

Fields tend to overlap with each other, and algorithm advances in one field may improve those of other, sometimes completely unrelated, fields. For example, dynamic programming was invented for optimization of resource consumption in industry but is now used in solving a broad range of problems in many fields.

By complexity

See also: Complexity class and Parameterized complexity

Algorithms can be classified by the amount of time they need to complete compared to their input size:

Constant time: if the time needed by the algorithm is the same, regardless of the input size. E.g. an access to an array element.

Linear time: if the time is proportional to the input size. E.g. the traverse of a list.

Logarithmic time: if the time is a logarithmic function of the input size. E.g. binary search algorithm.

Polynomial time: if the time is a power of the input size. E.g. the bubble sort algorithm has quadratic time complexity.

Exponential time: if the time is an exponential function of the input size. E.g. Brute-force search.

Some problems may have multiple algorithms of differing complexity, while other problems might have no algorithms or no known efficient algorithms. There are also mappings from some problems to other problems. Owing to this, it was found to be more suitable to classify the problems themselves instead of the algorithms into equivalence classes based on the complexity of the best possible algorithms for them.

Pseudo code is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm. Simply, we can say that it’s the cooked up representation of an algorithm. Often at times, algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is. Pseudo code, as the name suggests, is a false code or a representation of code which can be understood by even a layman with some school level programming knowledge.

Algorithm: It’s an organized logical sequence of the actions or the approach towards a particular problem. A programmer implements an algorithm to solve a problem. Algorithms are expressed using natural verbal but somewhat technical annotations.

Pseudo code: It’s simply an implementation of an algorithm in the form of annotations and informative text written in plain English. It has no syntax like any of the programming language and thus can’t be compiled or interpreted by the computer.

Advantages of Pseudocode

- Improves the readability of any approach. It’s one of the best approaches to start implementation of an algorithm.

- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that’s where a pseudo-code proves vital.

- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

How to write a Pseudo-code?

- Arrange the sequence of tasks and write the pseudocode accordingly.

- Start with the statement of a pseudo code which establishes the main goal or the aim.

Example:

This program will allow the user to check
the number whether it's even or odd.

- The way the if-else, for, while loops are indented in a program, indent the statements likewise, as it helps to comprehend the decision control and execution mechanism. They also improve the readability to a great extent.

Example:

```
if "1"
 print response
 "I am case 1"
if "2"
 print response
 "I am case 2"
```

Use appropriate naming conventions. The human tendency follows the approach to follow what we see. If a programmer goes through a pseudo code, his approach will be the same as per it, so the naming must be simple and distinct.

Use appropriate sentence casings, such as CamelCase for methods, upper case for constants and lower case for variables.

Elaborate everything which is going to happen in the actual code. Don't make the pseudo code abstract.

Use standard programming structures such as 'if-then', 'for', 'while', 'cases' the way we use it in programming.

Check whether all the sections of a pseudo code is complete, finite and clear to understand and comprehend.

Don't write the pseudo code in a complete programmatic manner. It is necessary to be simple to understand even for a layman or client, hence don't incorporate too many technical terms.

Examples:

- If student's grade is greater than or equal to 60
Print "passed"
else
Print "failed"

- Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Set the class average to the total divided by ten
Print the class average.

- Initialize total to zero
Initialize counter to zero
Input the first grade
while the user has not as yet entered the sentinel
add this grade into the running total
add one to the grade counter
input the next grade (possibly the sentinel)
if the counter is not equal to zero
set the average to the total divided by the counter
print the average
else
print 'no grades were entered'

- initialize passes to zero
initialize failures to zero
initialize student to one

while student counter is less than or equal to ten
input the next exam result
if the student passed
add one to passes
else
add one to failures
add one to student counter
print the number of passes
print the number of failures
if eight or more students passed
print "raise tuition"

Some Keywords That Should be Used

For looping and selection, The keywords that are to be used include Do While...EndDo; Do Until...Enddo; Case...EndCase; If...Endif; Call ... with (parameters); Call; Return ....; Return; When; Always use scope terminators for loops and iteration.

As verbs, use the words Generate, Compute, Process, etc. Words such as set, reset, increment, compute, calculate, add, sum, multiply, ... print, display, input, output, edit, test , etc. with careful indentation tend to foster desirable pseudocode.

Do not include data declarations in your pseudocode.

2. Try to find description in native language (English) of 3 any sorting methods on the Internet and make a pseudocode representation of each:

3. Make a report.

Laboratory work 8

1. Read intro materials:

Windows command line (Cmd.exe).

This material covers the basics of navigating and using the Microsoft Windows command line.

Keep in mind that there are over 100 different commands used in MS-DOS and the Windows command line.

Get into the Windows command line

Open a Windows command line window by following the steps below. If you need additional information or alternative methods for all versions of Windows, see our how to get into DOS and Windows command line page.

Click Start.

In the Search or Run line, type cmd (short for command), and press Enter.

Understanding the prompt

After following the above steps, the Windows command line should be shown (similar to the example below). Windows often starts you at your user directory. This prompt tells us we're in the C: drive (default hard drive letter) and currently in the current user directory, a subdirectory of the Users directory.

Key tips

MS-DOS and the Windows command line are not case sensitive.

The files and directories shown in Windows are also found in the command line.

When working with a file or directory with a space, surround it in quotes. For example, the directory My Documents would be "My Documents" when typed.

File names can have a long file name of 255 characters and a three character file extension.

When a file or directory is deleted in the command line, it is not moved into the Recycle Bin.

If you need help with any of command, type /? after the command. For example, dir /? would give the options available for the dir command.

Command-line syntax key

The following table describes the notation used to indicate command-line syntax.

Notation	Description
Text without brackets or braces	Items you must type as shown.
<Text inside angle brackets>	Placeholder for which you must supply a value.
[Text inside square brackets]	Optional items.
{Text inside braces}	Set of required items. You must choose one.
Vertical bar (|)	Separator for mutually exclusive items. You must choose one.
Ellipsis (…)	Items that can be repeated and used multiple times.

Listing the files

Let's learn your first command. Type dir at the prompt to list files in the current directory. You should get an output similar to the example image below. Without using any dir options, this is how dir output appears. As can be seen, you are given lots of useful information including the creation date and time, directories (<DIR>), and the name of the directory or file. In the example below, there are 0 files listed and 14 directories as indicated by the status at the bottom of the output.

Every command in the command line has options, which are additional switches and commands that can be added after the command. For example, with the dir command, you can type dir /p to list the files and directories in the current directory one page at a time. This switch is useful to see all the files and directories in a directory that has dozens or hundreds of files. Each of the command options and switches is listed in our DOS command overview. We offer guides for individual commands, as well. For example, if you want to see all the options for the dir command, refer to our dir command overview for a complete option listing.

The dir command can also be used to search for specific files and directories by using wildcards. For example, to list files or directories that begin with the letter "A" you could type dir a* to list only the AppData directory, in this above example. See the wildcard definition for other examples and help with using wildcards.

Moving into a directory

Now that we've seen a list of directories (shown below) in the current directory, move into one of those directories. To move into a directory, we use the cd command, so to move into the Desktop type cd desktop and press Enter. Once you've moved into a new directory, the prompt should change. Now in this desktop directory, see what files are found in this directory by typing the dir command again.

Understand the files

In Windows, you are familiar with files having icons that help represent the file type. In the command line, the same thing is accomplished by the file extensions.

Most users will only be concerned with executable files, is a file that ends with .exe, .com, and .bat. When the name of these files are typed into the command line, the program runs, which is the same as double-clicking a file in Windows.

Note

Keep in mind that if the executable file you are trying to run is not in the current directory, you'll get an error. Unless you have set a path for the directory that contains the executable file, which is how the command line finds external commands.

If you want to view the contents of a file, most versions of the command line use the edit command. For example, if we wanted to look at the log file hijackthis.log we would type edit hijackthis.log at the prompt. For 64-bit versions of Windows that do not support this command, you can use the start command, for example, type start notepad hijackthis.log to open the file in Notepad. Further information about opening and editing a file from the command line can also be found on the link below.

Moving back a directory

You learned earlier the cd command can move into a directory. This command also allows you to go back a directory by typing cd.. at the prompt. When this command is typed, you'll be moved out of the Desktop directory and back into the user directory. To move back to the root directory type cd\ to get to the C:\> prompt. If you know the name of the directory you want to move into, you can also type cd\ and the directory name. For example, to move into C:\Windows> type cd\windows at the prompt.

Creating a directory

Now with your basic understanding of navigating the command line let's start creating new directories. To create a directory in the current directory, use the mkdir command. For example, create a directory called "test" by typing mkdir test at the prompt. If created successfully, you should be returned to the prompt with no error message. After the directory is created, move into that directory with the cd command.

Switching drives

In some circumstances, you may want to copy or list files on another drive. To switch drives in the Windows command line, type the drive letter of the drive followed by a colon. For example, if your CD-ROM drive is the D drive, you would type d: and press Enter. If the drive exists, the prompt changes to that drive letter. If the drive does not exist or is not accessible (e.g., no disc in CD-ROM drive), you get an error.

Windows command prompt

The following examples use the Windows command prompt to copy files from one drive to another. For general information about using the command line, see our Windows command line guide.

copy c:\myfile.txt d:

The above command would copy the file "myfile.txt" on the C: drive to the D: drive.

copy *.txt e:

The above command uses a wildcard to copy all text files in the current directory to the E: drive.

copy f:\example.xls

The above command would copy the file "example.xls" on the F: drive to the current directory. Notice that we did not specify a destination; if the destination is not specified, the current directory is used by default.

For example, if your command prompt says C:\>, you are in the root of the C:\ drive, so the above command would copy F:\example.xls to the destination C:\example.xls.

Tip

If you need to switch between drives, type the letter of the drive followed by a colon at the command line. For example, to switch to the 'I' drive, type "i:". If done correctly, it should change your prompt to "I:\>".

See the copy command for further information and help with this command.

Using the xcopy command

xcopy command

Using the xcopy command, you can copy the all the files from one drive to another drive. Example:

xcopy /h /c /k /e /r /y c:\ d:\

The above command copies all of the files on the C:\ drive to the D:\ drive. There are many options specified here (the letters with a slash before them). Here's what they do:

Option Meaning

/h Copy hidden and system files. Normally xcopy skips these files, but if you specify this option, they are copied.

/c Continue copying, even if an error is encountered. Doing so can be helpful if you need to step away from the computer, and you know that you won't need to stop the copy operation, even if there are errors.

/k Keep read-only attribute. If you specify this option, read-only files will retain the read-only file attribute when they are copied.

/e Copy empty directories. Normally, empty directories are not copied. If this option is specified, xcopy will copy all directories, even if they are empty.

/r Overwrite read-only files in destination. Normally, if read-only files exist in your destination which would be overwritten by the copy, xcopy will stop and tell you "Access denied." If this option is specified, it will overwrite them and continue copying.

/y Assume yes to all overwrites. Normally, if a file would be overwritten by the copy, xcopy will ask you to confirm before it overwrites. If this option is specified, it will overwrite them without asking you.

Creating a new file

You can create a new file from the command line using the edit command, copy con command, or using the start command to open a file.

Moving and copying a file

Now that we've created a file let's move it into an alternate directory. To help make things easier, create another directory for the files. So, type mkdir dir2 to create a new directory in the test directory called dir2. After the new directory is created, use the move command to move the example.bat file into that directory. To do this type move example.bat dir2 at the prompt, if done successfully you should get a message indicated the file was moved. You could also substitute the move command for the copy command to copy the file instead of moving it.

Rename a file

After the file is moved into the dir2 directory, move into that directory with the cd command to rename the file. In the dir2 directory, use the rename command to rename the example file into an alternate name. Type rename example.bat first.bat at the prompt to rename the file to first.bat. Now when using the dir command, you should see the first.bat as the only file.

Tip

When renaming any file, make sure the file has the same file extension. If you were to rename the .bat file to a .txt file, it is no longer an executable file only a text file. Also, keep in mind that renaming the file to a different file extension does not convert the file. For example, if you renamed the file as an .MP3, it may look like an MP3 in Windows, but it's not going to play music.

Deleting a file

Now that we've had our fun with our new file, delete the file with the del command. Type del first.bat to delete the first.bat file. If successful, you are returned to the prompt with no errors, and the dir command shows no files in the current directory.

Tip. When deleting files, you can also use wildcards to delete multiple files at once. For example, if the directory contained several .GIF image files you could type del *.gif to delete all files ending with the .gif file extension.

Renaming a directory

Go back one directory to get back into the test directory by using the cd.. command mentioned earlier. Now rename our dir2 directory to something else using the same rename command we used earlier. At the prompt, type rename dir2 hope to rename the directory to hope. After this command is completed, type dir and you should now see one directory called hope.

Removing a directory

While still in the test directory, remove the hope directory by using the rmdir command. At the prompt, type rmdir hope to remove the hope directory.

Tip. If the directory you are trying to remove contains any files or directories, you'll receive an error. To prevent this error, use the /s option. For example, if the hope directory still had the first.bat file, you would need to tyRunning a program

Any file that is an executable file can be run from the command line by typing the name of the file. For example, if you listed files using the dir command and see a file named "myfile.exe" typing "myfile" at the command line runs that program.pe rmdir /s hope at the prompt.

How to list available commands

After getting a good understanding of using the command line from the steps shown above, you can move on to other available commands by typing help at the command line. Typing "help" gives you a listing of available commands with a brief description of each of the commands.

Closing or exiting the command line window

After you are done with the Windows command line, you can type exit to close the window.

2. Do the exercise:

- Open Windows Command Line.
- Change background color of Command Line to white and foreground (text color) to black.
- Create new Directory (Dir1) and subdirectory (Dir2) in current directory.
- Copy 5 any files from drive C to Dir2 directory.
- Rename one from copied files.
- Remove 3 any files from Dir2.
- Remove Dir1 with subdirectory.
- Try to change current date.
- Revert date to initial value.

3. Make a report.

Laboratory work 9

1. Read intro materials:

Flowcharts.

Flowchart is graphical/pictorial representation which is use to understand programing instructions (PROGRAMMING is set of instructions) . Flowchart also use for short explaining process

Let us take an example: it's a fact “human brain is easily understand and store picture instead of wordings” that's why we use flowchart as well.

Some standard rules for flowchart which everyone must be follow to make flowchart. And rules are very easy

Every software engineer must know algorithm and flowchart of it's code. If haven't so it's not big issue but they didn't able to explain his/her program.

Flowchart is a very simple yet powerful tool to improve productivity in both our personal and work life. Here are some ways flowchart can be helpful:

- Document a process.

- Present a solution.
- Brainstorm an idea.
- Design a system.
- Explain a decision making process.
- Store information.

It is not clear who was the true inventor of flowcharts, but the first standardized documentation on flow chart was first introduced by Frank and Lillian Gilbreth. In 1921, the couple presented the graphic-based method in a presentation titled: “Process Charts: First Steps in Finding the One Best Way to do Work”, to members of the American Society of Mechanical Engineers (ASME).

After that, in 1930s, Allan H. Mogensen, an industrial engineer trained some participants in his Work Simplification Conferences in New York. Participants from this conference such as Art Spinanger and Ben Grahamthen began to use flowchart in their respective fields, which helped propagate the usage of flowchart.

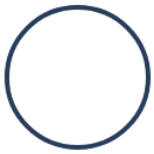
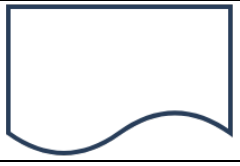
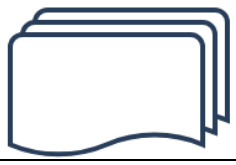
In 1947, ASME adopted a symbol set derived from Gilbreth’s original work as the “ASME Standard: Operation and Flow Process Charts.”

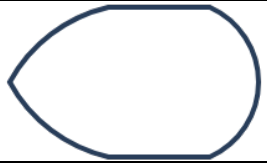
In the year 1949, flowchart began to be used for planning computer programs and quickly became one of the most popular tools in designing computer algorithms and programs. Nowadays, flow chart is an important productivity tool, serving employees in various industries and functions.

Flowchart is a very intuitive method to describe processes. As such, in most cases, you don’t need to worry too much about the standards and rules of all the flowchart symbols. In fact, a simple flowchart, constructed with just rectangular blocks and flowlines, can already get most jobs done.

However, if you want to get technical and precise, there are preset rules and standards you can follow about flow chart symbols. Specifically, the American National Standards Institute (ANSI) set standards for flowcharts and their symbols in the 1960s. Afterwards, the International Organization for Standardization (ISO) adopted the ANSI symbols in 1970. In general, flow charts flow from top to bottom and left to right.

Shape	Name	Description
	Flowline	Shows the process’ direction. Each flowline connects two blocks.
	Terminal	Indicates the beginning or end of a flowchart
	Process	Represent a step in a process. This is the most common component of a flowchart.

	Decision	Shows a step that decides the next step in a process. This is commonly a yes/no or true/false question.
	Input/Output	Indicates the process of inputting or outputting external data.
	Annotation / Comment	Indicates additional information regarding a step in a process.
	Predefined Process	Shows named process which is defined elsewhere.
	On-page Connector	Pairs of on-page connecter are used to replace long lines on a flowchart page.
	Off-page Connector	An off-page connector is used when the target is on another page.
	Delay	Any delay period that is part of a process
	Alternate process	An alternate to the normal process step. Flow lines to an alternate process block is usually dashed.
	Data	Data input or output
	Document	A document
	Multi-document	Multiple documents

	Preparation	A preparation step
	Display	A machine display
	Manual input	Manually input data or information into a system
	Manual operation	A process step that isn't automated

2. Do the exercise:

Create flowcharts of pseudocode from the 7th laboratory work theory part (marked as italic).

3. Make a report.

Laboratory work 10

1. Read intro materials:

Types of cyber threats

The threats countered by cyber-security are three-fold:

1. Cybercrime includes single actors or groups targeting systems for financial gain or to cause disruption.

2. Cyber-attack often involves politically motivated information gathering.

3. Cyberterrorism is intended to undermine electronic systems to cause panic or fear.

So, how do malicious actors gain control of computer systems? Here are some common methods used to threaten cyber-security:

Malware

Malware means malicious software. One of the most common cyber threats, malware is software that a cybercriminal or hacker has created to disrupt or damage a legitimate user's computer. Often spread via an unsolicited email attachment or legitimate-looking download, malware may be used by cybercriminals to make money or in politically motivated cyber-attacks.

There are a number of different types of malware, including:

- Virus: A self-replicating program that attaches itself to clean file and spreads throughout a computer system, infecting files with malicious code.

- Trojans: A type of malware that is disguised as legitimate software. Cybercriminals trick users into uploading Trojans onto their computer where they cause damage or collect data.

- Spyware: A program that secretly records what a user does, so that cybercriminals can make use of this information. For example, spyware could capture credit card details.

- Ransomware: Malware which locks down a user's files and data, with the threat of erasing it unless a ransom is paid.

- Adware: Advertising software which can be used to spread malware.

- Botnets: Networks of malware infected computers which cybercriminals use to perform tasks online without the user's permission.

SQL injection

An SQL (structured language query) injection is a type of cyber-attack used to take control of and steal data from a database. Cybercriminals exploit vulnerabilities in data-driven applications to insert malicious code into a databased via a malicious SQL statement. This gives them access to the sensitive information contained in the database.

Phishing

Phishing is when cybercriminals target victims with emails that appear to be from a legitimate company asking for sensitive information. Phishing attacks are often used to dupe people into handing over credit card data and other personal information.

Man-in-the-middle attack

A man-in-the-middle attack is a type of cyber threat where a cybercriminal intercepts communication between two individuals in order to steal data. For example, on an unsecure WiFi network, an attacker could intercept data being passed from the victim's device and the network.

Denial-of-service attack

A denial-of-service attack is where cybercriminals prevent a computer system from fulfilling legitimate requests by overwhelming the networks and servers with traffic. This renders the system unusable, preventing an organization from carrying out vital functions.

Cyber safety tips - protect yourself against cyberattacks

How can businesses and individuals guard against cyber threats? Here are our top cyber safety tips:

- Update your software and operating system: This means you benefit from the latest security patches.

- Use anti-virus software: Security solutions like Kaspersky Total Security will detect and removes threats. Keep your software updated for the best level of protection.

- Use strong passwords: Ensure your passwords are not easily guessable.

- Do not open email attachments from unknown senders: These could be infected with malware.

- Do not click on links in emails from unknown senders or unfamiliar websites:This is a common way that malware is spread.

- Avoid using unsecure WiFi networks in public places: Unsecure networks leave you vulnerable to man-in-the-middle attacks.

Encryption and decryption. Data that can be read and understood without any special measures is called plaintext or cleartext. The method of disguising plaintext in such a way as to hide its substance is called encryption. Encrypting plaintext results in unreadable gibberish called ciphertext. You use encryption to ensure that information is hidden from anyone for whom it is not intended, even those who can see the encrypted data. The process of reverting ciphertext to its original plaintext is called decryption. What is cryptography?

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.

While cryptography is the science of securing data, cryptanalysis is the science of analyzing and breaking secure communication. Classical cryptanalysis involves an interesting combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination, and luck. Cryptanalysts are also called attackers.

Cryptology embraces both cryptography and cryptanalysis.

How does cryptography work?

A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a key – a word, number, or phrase – to encrypt the plaintext. The same plaintext encrypts to different ciphertext with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key. A cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a cryptosystem.

Conventional cryptography. In conventional cryptography, also called secret-key or symmetric-key encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) is an example of a conventional cryptosystem that is widely employed by the Federal Government (figure 40):

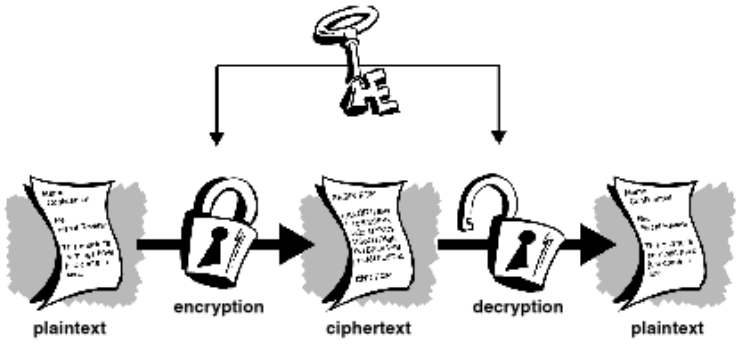


Figure 40 – The Data Encryption Standard

Caesar's Cipher

An extremely simple example of conventional cryptography is a substitution cipher. A substitution cipher substitutes one piece of information for another. This is most frequently done by offsetting letters of the alphabet. Two examples are Captain Midnight's Secret Decoder Ring, which you may have owned when you were a kid, and Julius Caesar's cipher. In both cases, the algorithm is to offset the alphabet and the key is the number of characters to offset it.

For example, if we encode the word "SECRET" using Caesar's key value of 3, we offset the alphabet so that the 3rd letter down (D) begins the alphabet.

So starting with

ABCDEFGHIJKLMNOPQRSTUVWXYZ

and sliding everything up by 3, you get

DEFGHIJKLMNOPQRSTUVWXYZABC

where D=A, E=B, F=C, and so on.

Using this scheme, the plaintext, "SECRET" encrypts as "VHFUHW." To allow someone else to read the ciphertext, you tell them that the key is 3.

Obviously, this is exceedingly weak cryptography by today's standards, but hey, it worked for Caesar, and it illustrates how conventional cryptography works.

Key management and conventional encryption

Conventional encryption has benefits. It is very fast. It is especially useful for encrypting data that is not going anywhere. However, conventional encryption alone as a means for transmitting secure data can be quite expensive simply due to the difficulty of secure key distribution.

Recall a character from your favorite spy movie: the person with a locked briefcase handcuffed to his or her wrist. What is in the briefcase, anyway? It's probably not the missile launch code/ biotoxin formula/ invasion plan itself. It's the key that will decrypt the secret data.

For a sender and recipient to communicate securely using conventional encryption, they must agree upon a key and keep it secret between themselves. If they are in different physical locations, they must trust a courier, the Bat Phone, or some other secure communication medium to prevent the disclosure of the secret key during transmission. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all information encrypted or authenticated with that key. From DES to Captain Midnight's Secret Decoder Ring, the persistent problem with conventional encryption is key distribution: how do you get the key to the recipient without someone intercepting it?

Public key cryptography

The problems of key distribution are solved by public key cryptography, the concept of which was introduced by Whitfield Diffie and Martin Hellman in 1975. (There is now evidence that the British Secret Service invented it a few years before Diffie and Hellman, but kept it a military secret – and did nothing with it. [J H Ellis: The Possibility of Secure Non-Secret Digital Encryption, CESG Report, January 1970])

Public key cryptography is an asymmetric scheme that uses a pair of keys for encryption: a public key, which encrypts data, and a corresponding private, or secret key for decryption. You publish your public key to the world while keeping your private key secret. Anyone with a copy of your public key can then encrypt information that only you can read. Even people you have never met.

It is computationally infeasible to deduce the private key from the public key. Anyone who has a public key can encrypt information but cannot decrypt it. Only the person who has the corresponding private key can decrypt the information (figure 41):

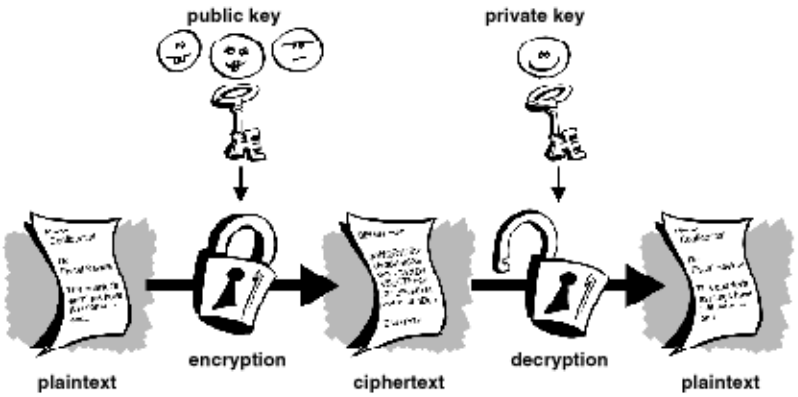


Figure 41 – Demonstration of encryption-decryption process using private and public keys

The primary benefit of public key cryptography is that it allows people who have no preexisting security arrangement to exchange messages securely. The need for sender and receiver to share secret keys via some secure channel is eliminated; all communications involve only public keys, and no private key is ever transmitted or shared. Some examples of public-key cryptosystems are Elgamal (named for its inventor, Taher Elgamal), RSA (named for its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman), Diffie-Hellman (named, you guessed it, for its inventors), and DSA, the Digital Signature Algorithm (invented by David Kravitz).

Because conventional cryptography was once the only available means for relaying secret information, the expense of secure channels and key distribution relegated its use only to those who could afford it, such as governments and large banks (or small children with secret decoder rings). Public key encryption is the technological revolution that provides strong cryptography to the adult masses. Remember the courier with the locked briefcase handcuffed to his wrist? Public-key encryption puts him out of business (probably to his relief).

Keys

A key is a value that works with a cryptographic algorithm to produce a specific ciphertext. Keys are basically really, really, really big numbers. Key size is measured in bits; the number representing a 1024-bit key is darn huge. In public key cryptography, the bigger the key, the more secure the ciphertext.

However, public key size and conventional cryptography's secret key size are totally unrelated. A conventional 80-bit key has the equivalent strength of a 1024-bit

public key. A conventional 128-bit key is equivalent to a 3000-bit public key. Again, the bigger the key, the more secure, but the algorithms used for each type of cryptography are very different and thus comparison is like that of apples to oranges.

While the public and private keys are mathematically related, it's very difficult to derive the private key given only the public key; however, deriving the private key is always possible given enough time and computing power. This makes it very important to pick keys of the right size; large enough to be secure, but small enough to be applied fairly quickly. Additionally, you need to consider who might be trying to read your files, how determined they are, how much time they have, and what their resources might be.

Larger keys will be cryptographically secure for a longer period of time. If what you want to encrypt needs to be hidden for many years, you might want to use a very large key. Of course, who knows how long it will take to determine your key using tomorrow's faster, more efficient computers? There was a time when a 56-bit symmetric key was considered extremely safe.

Digital signatures

Amajor benefit of public key cryptography is that it provides a method for employing digital signatures. Digital signatures enable the recipient of information to verify the authenticity of the information's origin, and also verify that the information is intact. Thus, public key digital signatures provide authentication and data integrity. A digital signature also provides non-repudiation, which means that it prevents the sender from claiming that he or she did not actually send the information. These features are every bit as fundamental to cryptography as privacy, if not more.

A digital signature serves the same purpose as a handwritten signature. However, a handwritten signature is easy to counterfeit. A digital signature is superior to a handwritten signature in that it is nearly impossible to counterfeit, plus it attests to the contents of the information as well as to the identity of the signer (figure 42).

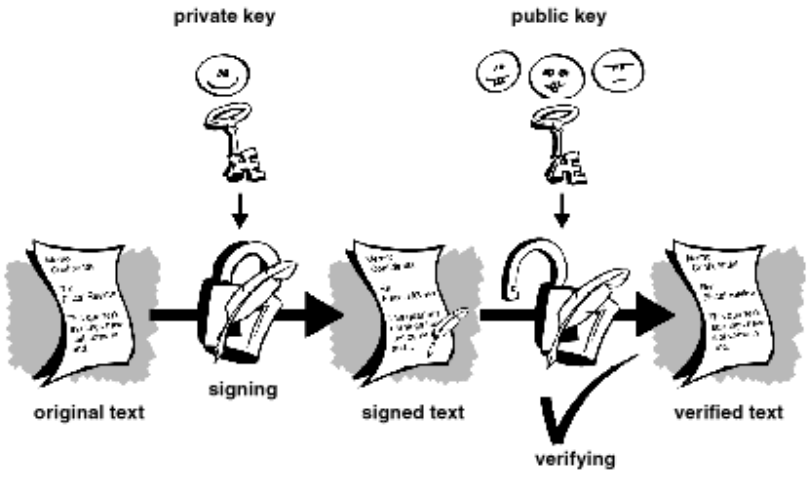


Figure 42 – Digital signature

Some people tend to use signatures more than they use encryption. For example, you may not care if anyone knows that you just deposited $1000 in your account, but you do want to be darn sure it was the bank teller you were dealing with

Digital certificates

One issue with public key cryptosystems is that users must be constantly vigilant to ensure that they are encrypting to the correct person's key. In an environment where it is safe to freely exchange keys via public servers, man-in-the-middle attacks are a potential threat. In this type of attack, someone posts a phony key with the name and user ID of the user's intended recipient. Data encrypted to – and intercepted by – the true owner of this bogus key is now in the wrong hands.

In a public key environment, it is vital that you are assured that the public key to which you are encrypting data is in fact the public key of the intended recipient and not a forgery. You could simply encrypt only to those keys which have been physically handed to you. But suppose you need to exchange information with people you have never met; how can you tell that you have the correct key?

Digital certificates, or certs, simplify the task of establishing whether a public key truly belongs to the purported owner.

A certificate is a form of credential. Examples might be your driver's license, your social security card, or your birth certificate. Each of these has some information on it identifying you and some authorization stating that someone else has confirmed your identity. Some certificates, such as your passport, are important enough confirmation of your identity that you would not want to lose them, lest someone use them to impersonate you.

A digital certificate is data that functions much like a physical certificate. A digital certificate is information included with a person's public key that helps others verify that a key is genuine or valid. Digital certificates are used to thwart attempts to substitute one person's key for another.

A digital certificate consists of three things:

A public key.

Certificate information. ("Identity" information about the user, such as name, user ID, and so on.)

One or more digital signatures.

The purpose of the digital signature on a certificate is to state that the certificate information has been attested to by some other person or entity. The digital signature does not attest to the authenticity of the certificate as a whole; it vouches only that the signed identity information goes along with, or ** the public key.

Thus, a certificate is basically a public key with one or two forms of ID attached, plus a hearty stamp of approval from some other trusted individual.

Certificate distribution

Certificates are utilized when it's necessary to exchange public keys with someone else. For small groups of people who wish to communicate securely, it is easy to manually exchange diskettes or emails containing each owner's public key. This is manual public key distribution, and it is practical only to a certain point. Beyond that point, it is necessary to put systems into place that can provide the

necessary security, storage, and exchange mechanisms so coworkers, business partners, or strangers could communicate if need be. These can come in the form of storage-only repositories called ** or more structured systems that provide additional key management features and are called Public Key Infrastructures (PKIs).

Certificate servers

A certificate server, also called a cert server or a key server, is a database that allows users to submit and retrieve digital certificates. A cert server usually provides some administrative features that enable a company to maintain its security policies – for example, allowing only those keys that meet certain requirements to be stored.

Public Key Infrastructures

A PKI contains the certificate storage facilities of a certificate server, but also provides certificate management facilities (the ability to issue, revoke, store, retrieve, and trust certificates). The main feature of a PKI is the introduction of what is known as a Certification Authority, orCA, which is a human entity – a person, group, department, company, or other association – that an organization has authorized to issue certificates to its computer users. (A CA's role is analogous to a country's government's Passport Office.) A CA creates certificates and digitally signs them using the CA's private key. Because of its role in creating certificates, the CA is the central component of a PKI. Using the CA's public key, anyone wanting to verify a certificate's authenticity verifies the issuing CA's digital signature, and hence, the integrity of the contents of the certificate (most importantly, the public key and the identity of the certificate holder).

Validity and trust

Every user in a public key system is vulnerable to mistaking a phony key (certificate) for a real one. Validity is confidence that a public key certificate belongs to its purported owner. Validity is essential in a public key environment where you must constantly establish whether or not a particular certificate is authentic.

When you've assured yourself that a certificate belonging to someone else is valid, you can sign the copy on your keyring to attest to the fact that you've checked the certificate and that it's an authentic one. If you want others to know that you gave the certificate your stamp of approval, you can export the signature to a certificate server so that others can see it.

Trust models

In relatively closed systems, such as within a small company, it is easy to trace a certification path back to the root CA. However, users must often communicate with people outside of their corporate environment, including some whom they have never met, such as vendors, customers, clients, associates, and so on. Establishing a line of trust to those who have not been explicitly trusted by your CA is difficult.

Companies follow one or another trust model, which dictates how users will go about establishing certificate validity. There are three different models:

Direct Trust

Hierarchical Trust

A Web of Trust

Key splitting

They say that a secret is not a secret if it is known to more than one person. Sharing a private key pair poses such a problem. While it is not a recommended practice, sharing a private key pair is necessary at times. Corporate signing Keys, for example, are private keys used by a company to sign – for example – legal documents, sensitive personnel information, or press releases to authenticate their origin. In such a case, it is worthwhile for multiple members of the company to have access to the private key. However, this means that any single individual can act fully on behalf of the company.

In such a case it is wise to split the key among multiple people in such a way that more than one or two people must present a piece of the key in order to reconstitute it to a usable condition. If too few pieces of the key are available, then the key is unusable.

Some examples are to split a key into three pieces and require two of them to reconstitute the key, or split it into two pieces and require both pieces. If a secure network connection is used during the reconstitution process, the key's shareholders need not be physically present in order to rejoin the key.

2. Do the exercise:

- Find the algorithm of Caesar’s cipher.
- Create table of codes from the English alphabet (Letter A – number one, letter B – number two, etc.). For specifying the offset of Caesar’s cipher in range from 1 till 6 use your variant in student’s book and algorithm from Student's instruction of the Rules of reports making.
- Try to encrypt your full name using Caesar’s cipher.
- Explain how strong Caesar’s cipher.
- Explain the difference between symmetric and asymmetric ciphers.

3. Make a report.

PART 4. EXAMINATION

Test by discipline

Choose the correct answer.

1. The brain of any computer system is

A. ALU
B. Memory
C. CPU
D. Control unit

2. What difference does the 5th generation computer have from other generation computers?

A. Technological advancement
B. Scientific code
C. Object Oriented Programming
D. All of the above

3. The two kinds of main Memory are:

A. Primary and secondary
B. Random and sequential
C. ROM and RAM
D. All of the above
E. None of the above

4. When an input electrical signal A=10100 is applied to a NOT gate, its output signal is

A. 00101
B. 10001
C. 10101
D. 01011

5. With respect to a network interface card, the term 10/100 refers to

A. Protocol speed
B. Megabits per seconds
C. A fiber speed

6. Which of the following is used for modulation and demodulation?

A. modem
B. protocols
C. gateway

7. The step-by-step instructions that solve a problem are called _____.

A. An algorithm
B. A list
C. A sequential structure
D. None of the above

8. A problem's _____ will answer the question, "What information will the computer need to know in order to either print or display the output times?"

A. Output
B. Input

9. The Internet is the global system of interconnected computer networks. Is it true?
A. No
B. Yes

10. An if statement in programming is:
A. Cycle statement
B. Conditional statement

11. What keyword is describes “logical” data type?
A. Boolean
B. Integer

12. What does “pseudocode” mean?
A. Wrong code
B. An informal high-level description of the operating principle of a computer program or other algorithm

13. What extension has Microsoft Word document?
A. .doc (.docx)
B. .ppt (.pptx)
C. .txt

14. What is the name of program for surfing in the Internet?
A. Microsoft Powerpoint
B. Internet Explorer (Edge)
C. Total Commander

15. Variables declared outside a block are called _____
A. global
B. universal
C. stellar
D. external

16. What is the name of programming language?
A. Delphi
B. Delfi

17. What “scanner” as a piece of computer hardware is?
A. Input device
B. Output device

18. Software is:
A. A collection of instructions that enable the user to interact with a computer
B. Physical parts or components of a computer

19. Short name of standard storage device in personal computer is:
A. USB
B. HDD
C. PC
D. None of the above

20. What does abbreviation "ip" mean (in computer science)?
A. Internet portal
B. Intellectual property
C. Instructor pilot
D. Internal protocol
E. None of the above

Explain your answers briefly but cogently in the space provided. Sketches might be helpful.

Questions for credit by discipline

1. What is Computer Science?
2. What is Boolean Logic?
3. How computers store and represent numerical data?
4. How Computers Calculate - the ALU?
5. The CPU and Von Neumann Architecture
6. What is an Algorithm?
7. What is Data Structure?
8. What is Software Engineering?
9. What is an Operating System?
10. Basic computer organization
11. What is a file and file system?
12. Linux File System Types
13. Linux File System Directories
14. Basic Linux Commands
15. What is a Computer Network?
16. Types of Computer Networks
17. What is the Internet?
18. Basic World Wide Web Concepts
19. Web Browser and Web Server
20. Basics of Information and Computer Security

RECOMMENDED LITERATURE

1 Algorithms Unlocked. MIT Press, 2015.
2 Thomas H. Cormen, Introduction to Algorithms. MIT Press, 2009.
3 X. Yang et al. Quicksilver: Fast predictive image registration–a deep learning approach. NeuroImage, 158:378–396, 2017.
4 Harold Abelson, Gerald Jay Sussman, Julie Sussman, Structure and Interpretation of Computer Programs 2nd Edition, 1996.
5 Brian Christian, Tom Griffiths, Algorithms to Live By The Computer Science of Human Decisions, 2016.
6 Roselyn Teukolsky M.S., Barron's AP Computer Science A, 8th Edition with Bonus Online Tests, 2018.
7 Erol Gelenbe, Paolo Campegiani, Tadeusz Czachórski, Sokratis K. Katsikas, First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, February 26-27, Security in Computer and Information Sciences, 2018.
8 Mr. Kevin P Hare, Computer Science Principles. The Foundational Concepts of Computer Science - For AP® Computer Science Principles, 2018.
9 Jeff Erickson, Algorithms, 2019.
10 Roselyn Teukolsky M.S., AP Computer Science A With 6 Practice Tests, 2019.
11 Robert Sedgewick, Kevin Wayne, Computer Science An Interdisciplinary Approach, 2016.
12 Nell Dale, John Lewis, Computer Science Illuminated, 2019.
13 Daniel Drescher, Blockchain Basics A Non-Technical Introduction in 25 Steps, 2017.
14 Steven S. Skiena, The Data Science Design Manual, 2018.
15 Sjaak Laan, IT Infrastructure Architecture Infrastructure Building Blocks and Concepts Third Edition, 2017.
16 Wladston Ferreira Filho, Raimondo Pictet, Computer Science Distilled. Learn the Art of Solving Computational Problems, 2017.
17 Geoff Mulgan, Big Mind. How Collective Intelligence Can Change Our World, 2017.
18 Chris Bernhardt, Turing's Vision. The Birth of Computer Science, 2017.
19 Simson L Garfinkel, Rachel H. Grunspan, The Computer Book. From the Abacus to Artificial Intelligence, 250 Milestones in the History of Computer Science, 2018.
20 Max Bramer, Principles of Data Mining, 2016.
21 Ernest Dainow, Understanding Computers, Smartphones and the Internet, 2018.
22 Bradley Efron, Trevor Hastie, Computer Age Statistical Inference. Algorithms, Evidence, and Data Science, 2016.
23 David Watson, Helen Williams, Computer Science Workbook, 2016.
24 Shane Torbert, Applied Computer Science, 2016.
25 Michio Shibuya, Takashi Tonagi, Office Sawa, The Manga Guide to Microprocessors, 2017.