БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Энергетический

Кафедра Тепловые электрические станции

СОГЛАСОВАНО
Заведующий кафедрой

_____Н.Б. Карницкий

__ _____ 2016 г.

СОГЛАСОВАНО
Декан

_____ К.В. Добрего

__ _____ 2016 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

ИНФОРМАТИКА

для специальностей 1-43 01 04 «Тепловые электрические станции»
1-53 01 04 «Автоматизация и управление теплоэнергетическими процессами»

Составители:

к.т.н., доцент Тарасевич Л.А., ст. преподаватель Пронкевич Е.В.

Рассмотрено и утверждено

на заседании Совета энергетического факультета «24»марта 2016 г.,

протокол № 7

Перечень материалов

1. Теоретический раздел:

– «Информатика» - курс лекций;

2. Практический раздел:

– «Информатика» - задания для выполнения лабораторных работ;

3. Контроль знаний:

– «Информатика» - перечень вопросов, выносимых на экзамен и задания для курсовой работы;

4. Вспомогательный раздел:

– «Информатика» - типовая учебная программа для учреждения высшего образования.

Пояснительная записка

Целью создания ЭУМК является обучение методам решения информационных задач, приобретение навыков работы на современных вычислительных средствах, изучение новых информационных технологий.

В качестве базового учебного языка программирования выбран объектно-ориентированный язык C++, позволяющий осваивать классические приемы и современные технологии программирования. Полученные базовые навыки далее развиваются посредством обучения визуальному и объектно-ориентированному программированию с использованием языка Java.

Материалы данного электронного учебно-методического комплекса можно использовать при выполнении курсовой работы.

Задачами ЭУМК является решение задач математического моделирования, обработки массивов данных, представленных в виде таблиц или списков, представления результатов обработки в виде отчетов, программирование на алгоритмическом языке C++; применение стандартных программ для компьютерного моделирования технических задач;

Особенности структурирования и подачи учебного материала:

- теоретическая часть включает в себя конспект лекций по дисциплине «Информатика и интегрированные прикладные системы» и содержит два раздела. Первый раздел – объектно-ориентированное программирование на

языках программирования С++, Java. Второй раздел –численные методы решения нелинейных уравнений и систем линейных и нелинейных алгебраических систем на языке программирования С++.

- практическая часть состоит из набора заданий для выполнения лабораторных работ по дисциплине;

- раздел контроля знаний содержит вопросы к экзамену и перечень заданий для выполнения курсовых работ;

- вспомогательный раздел содержит типовую учебную программу по дисциплине «Информатика и интегрированные прикладные системы».

Рекомендации по организации работы с УМК (ЭУМК): Материалы данного электронного учебно-методического комплекса можно использовать при выполнении лабораторных и курсовых работ, посвященных объектно-ориентированному программированию на языке программирования С++. И численным методам решения нелинейных уравнений и систем линейных и нелинейных алгебраических уравнений на языке программирования С++.

ОГЛАВЛЕНИЕ

1. КУРС ЛЕКЦИЙ 9
РАЗДЕЛ I. ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ 9
1. 1.1. ВВЕДЕНИЕ 9
1.1. 1.1. Основные понятия и обозначения: алгоритмы, языки, программы 9
1.1. 1. 2. Современное объектно-ориентированное программирование: языки С и С++, достоинства и недостатки 11
1.1. 2. ПРОСТЫЕ ТИПЫ ДАННЫХ 13
1.1. 2. 1.Элементы языка С++ 13
1.1.2.2. Операции 18
1.1.3. ФУНКЦИИ 23
1.1.3.1. Объявление, определение, вызов 23
1.1.3.2. Способы передачи параметров, их типы 25
1.1.3.3. Перегрузка функций, указатели на функцию 26
1.1.3.4. Рекурсия 27
1.1.3.5. Математические функции 28
1.1.4. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ 30
1.1.4.1 . Статические и динамические массивы 30
1.1.4.2. Инициализация массива 32
1.1.4.3. Строки 36
1.1.4.4. Структуры 39
1.1.4.5. Объединения 41
1.1.4.6. Поля битов 43
1.1.4.7. Перечисление. Структуры. Определения 44
1.1.4.8. Сортировка массивов 45
1.1.4.9. Нахождение суммы элементов массива 46
1.1.4.10. Нахождение произведения элементов массива 46
1.1.5. ФАЙЛЫ 47
1.1.5.1. Объявление, создание, чтение, корректировка 47
1.1.5.2. Типы файлов 49
1.1.5.3. Потоки 53
1.1.6. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ 54
1.1.6.1. Абстракция данных 54
1.1.6.2. Три принципа объектно-ориентированного программирования 55
1.1.6.3. Функции-шаблоны и классы-шаблоны 55

1.1.6.4. Классы 57
1.1.7.ВВОД-ВЫВОД 60
1.1.7.1. Компененты и функции, используемые для ввода-вывода 60
1.1.8. УПРАВЛЯЮЩИЕ КОМПОНЕНТЫ, МЕНЮ 66
1.1. 9. РАБОТА С ТЕКСТОМ 82
1.1.9.1. Стандартные процедуры и функции для работы со строками 82
1.1.10. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ 91
1.1.10.1. Рисование элементарных фигур 91
1.1.11. ЯЗЫК JAVA 100
1.1.11.1. Апплеты Java 101
1.1.11.2. Базовые типы 114
1.1.11.5. Элементы управления 123
1.1.11.6. Сети 130
РАЗДЕЛ II. ЧИСЛЕННЫЕ МЕТОДЫ 131
1.2.1. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ 131
1.2.1.1. Метод бисекции 131
1.2.1.2. Метод хорд 132
1.2.1.3. Метод простой итерации 132
1.2.1.4. Метод Ньютона (метод касательных) 136
1.2.2. ИНТЕРПОЛЯЦИЯ 140
1.2.2.1.Системы функций Чебышева 140
1.2.2.2. Формула Лагранжа 141
1.2.2.3. Линейная интерполяция 143
1.2.3. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ГАУССА 145
1.2.4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДАМИ ПРОСТОЙ ИТЕРАЦИИ И МЕТОДОМ ЗЕЙДЕЛЯ 147
1.2.5. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ НЬЮТОНА 149
1.2.6. МЕТОД ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ 150
1.2.6.1. Формула трапеций 152
1.2.6.2. Формулы прямоугольников 153
1.2.6.3. Формула Симпсона 155
1.2.7. АППРОКСИМАЦИЯ 157

1.2.8. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПЕРВОГО ПОРЯДКА. 159
1.2.8.1. Метод Эйлера (метод Рунге-Кутта 1-го порядка) 160
1.2.8.2. Модифицированный метод Эйлера 165
(метод Рунге-Кутта 2-го порядка). 165
Метод Эйлера-Коши 165
1.2.8.3.Метод усредненных точек 168
1.2.8.4. Метод Рунге-Кутта 4 порядка 170
1.2.8.5. Общая характеристика методов 172
1.2.9. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ВЫСШИХ ПОРЯДКОВ 173
1.2.10. БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ ФУНКЦИЙ 174
1.2.10.1. Метод Фибоначчи 174
1.2.10.2. Метод золотого сечения 177
2.ЛАБОРАТОРНЫЕ ЗАДАНИЯ 180
Лабораторная работа № 1.ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЯ 180
Лабораторная работа № 2. ОПЕРАТОР IF 182
Лабораторная работа № 3. ОПЕРАТОР SWITCH 185
Лабораторная работа № 4. ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ 188
Лабораторная работа № 5. ВЫЧИСЛЕНИЕ КОНЕЧНЫХ СУММ 190
Лабораторная работа №6. ОДНОМЕРНЫЕ МАССИВЫ 192
Лабораторная работа № 7. ДВУМЕРНЫЕ МАССИВЫ 193
Лабораторная работа №8. ПОДПРОГРАММЫ 195
Лабораторная работа №9. ФАЙЛЫ 197
Лабораторная работа №10. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ 199
Лабораторная работа №10. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ 201
Лабораторная работа №11. ИТЕРАЦИОННЫЕ МЕТОДЫ 203
Лабораторная работа №12. ИНТЕРПОЛИРОВАНИЕ 206
Лабораторная работа №13. РЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ НЬЮТОНА 208
Лабораторная работа №14. РЕШЕНИЕ ЗАДАЧИ АППРОКСИМАЦИИ 210
Лабораторная работа №15. ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННОГО ИНТЕГРАЛА 212
Лабораторная работа №16 214

РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПЕРВОГО ПОРЯДКА 214
Лабораторная работа №17 216
РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ВЫСШИХ ПОРЯДКОВ 216
Лабораторная работа №18. ОПТИМИЗАЦИЯ. ЗАДАЧА МИНИМИЗАЦИИ 218
3.1. ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ 221
3.2. ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕМ КУРСОВОГО ПРОЕКТИРОВАНИЯ 224
4. ТИПОВАЯ ПРОГРАММА 226

Электронный учебно-методический комплекс

Теоретический раздел

ИНФОРМАТИКА

Курс лекций

Минск 2016

1. КУРС ЛЕКЦИЙ

РАЗДЕЛ I. ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

1. 1.1. ВВЕДЕНИЕ

1.1. 1.1. Основные понятия и обозначения: алгоритмы, языки, программы

Решение задач на компьютере основано на понятии алгоритма. Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к исходному результату.

Алгоритм означает точное описание некоторого процесса, инструкцию по его выполнению. Разработка алгоритма является сложным и трудоемким процессом. Алгоритмизация – это техника разработки (составления) алгоритма для решения задач на ЭВМ.

Для записи алгоритма решения задачи применяются следующие изобразительные способы их представления:

1. Словесно- формульное описание.
2. Блок-схема (схема графических символов).
3. Алгоритмические языки.
4. Операторные схемы.
5. Псевдокод.

Для записи алгоритма существует общая методика:

1. Каждый алгоритм должен иметь имя, которое раскрывает его смысл.
2. Необходимо обозначить начало и конец алгоритма.
3. Описать входные и выходные данные.
4. Указать команды, которые позволяют выполнять определенные действия над выделенными данными.

Общий вид алгоритма:

- название алгоритма;
- описание данных;
- начало;
- команды;
- конец.

Формульно-словесный способ записи алгоритма характеризуется тем, что описание осуществляется с помощью слов и формул. Содержание последовательности этапов выполнения алгоритмов записывается на естественном профессиональном языке предметной области в произвольной форме.

Графический способ описания алгоритма (блок - схема) получил самое широкое распространение. Для графического описания алгоритмов используются схемы алгоритмов или блочные символы (блоки), которые соединяются между собой линиями связи.

Каждый этап вычислительного процесса представляется геометрическими фигурами (блоками). Они делятся на арифметические или вычислительные (прямоугольник), логические (ромб) и блоки ввода-вывода данных (параллелограмм).

Порядок выполнения этапов указывается стрелками, соединяющими блоки. Геометрические фигуры размещаются сверху вниз и слева на право. Нумерация блоков производится в порядке их размещения в схеме.

Алгоритмические языки - это специальное средство, предназначенное для записи алгоритмов в аналитическом виде. Алгоритмические языки близки к математическим выражениям и к естественным языкам. Каждый алгоритмический язык имеет свой словарь. Алгоритм, записанный на алгоритмическом языке, выполняется по строгим правилам этого конкретного языка.

Операторные схемы алгоритмов. Суть этого способа описания алгоритма заключается в том, что каждый оператор обозначается буквой (например, А – арифметический оператор, Р – логический оператор и т.д.).

Операторы записываются слева направо в последовательности их выполнения, причем, каждый оператор имеет индекс, указывающий порядковый номер оператора. Алгоритм записывается в одну строку в виде последовательности операторов.

Псевдокод – система команд абстрактной машины. Этот способ записи алгоритма с помощью операторов близких к алгоритмическим языкам.

Типы алгоритмических процессов

По структуре выполнения алгоритмы и программы делятся на три вида:

- линейные;
- ветвящиеся;
- циклические;

Линейные вычислительные процессы

Линейный алгоритм (линейная структура) – это такой алгоритм, в котором все действия выполняются последовательно друг за другом и только один раз. Схема представляет собой последовательность блоков, которые располагаются сверху вниз в

порядке их выполнения. Первичные и промежуточные данные не оказывают влияния на направление процесса вычисления.

Алгоритмы разветвляющейся структуры

На практике часто встречаются задачи, в которых в зависимости от первоначальных условий или промежуточных результатов необходимо выполнить вычисления по одним или другим формулам.

Такие задачи можно описать с помощью алгоритмов разветвляющейся структуры. В таких алгоритмах выбор направления продолжения вычисления осуществляется по итогам проверки заданного условия. Ветвящиеся процессы описываются оператором IF (условие).

Циклические вычислительные процессы

Для решения многих задач характерно многократное повторение отдельных участков вычислений. Для решения таких задач применяются алгоритмы циклической структуры (циклические алгоритмы). Цикл – последовательность команд, которая повторяется до тех пор, пока не будет выполнено заданное условие. Циклическое описание многократно повторяемых процессов значительно снижает трудоемкость написания программ.

Особенностью первой схемы является то, что проверка условия выхода из цикла проводится до выполнения тела цикла. В том случае, если условие выхода из цикла выполняется, то тело цикла не выполняется ни разу.

Особенностью второй схемы является то, что цикл выполняется хоты бы один раз, так как первая проверка условия выхода из цикла осуществляется после того, как тело цикла выполнено.

Существуют циклы с известным числом повторений и итерационные циклы. При итерационном цикле выход из тела цикла, как правило, происходит при достижении заданнойточности вычисления.

1.1. 1. 2. Современное объектно-ориентированное программирование: языки С и С++, достоинства и недостатки

Языки программирования – это искусственные языки записи алгоритмов для исполнения их на ЭВМ. Программирование (кодирование) - составление программы по заданному алгоритму.

Классификация языков программирования. В общем, языки программирования делятся на две группы: операторные и функциональные. К функциональным относятся ЛИСП, ПРОЛОГ и т.д.

Операторные языки делятся на процедурные и непроцедурные (Smalltalk, QBE). Процедурные делятся на машино - ориентированные и машино – независимые.

К машино – ориентированным языкам относятся: машинные языки, автокоды, языки символического кодирования, ассемблеры.

К машино – независимым языкам относятся:

1. Процедурно – ориентированные (Паскаль, Фортран и др.).
2. Проблемно – ориентированные (ЛИСП и др.).
3. Объектно-ориентированные (Си++, Visual Basic, Java и др.).

Язык С++ был разработан в Bell Laboratories(США) в 1983г. как расширение языка С. Символы “++” – представляют оператор инкрементации в языке С, что в действительности означает выполнимость С - программ в среде С++. Еще одним источником С++ разработчик языка Б.Страуструп называет язык Simula67, из которого позаимствованы понятия класса, виртуальных функций. Класс соответствует введенному пользователем типу, для которого обеспечивается набор данных, операции над данными, их сокрытие и др. В настоящее время классы рассматриваются как носители таких черт объектно-ориентированного программирования как инкапсуляция, наследование, полиморфизм.

Существует несколько реализаций системы, поддерживающих стандарт С++, из которых можно выделить реализации Visual C++(Microsoft), Borland C++ (Inprise), а также реализацию в системе UNIX. Отличия относятся, в основном, к используемым библиотекам классов и интегрируемым средам.

Языки С и С++

В С++ привычные операторы и обозначения используются совсем иначе. По сравнению с "классическим" С Страуструп ввел всего несколько дополнительных ключевых слов и операторов, но "выжал" из них максимально много. При поверхностном взгляде программа на С++ может выглядеть очень похожей на программу на С, но операторы и выражения в ней сочетаются совсем не так.

Рассмотрим простую программу, которая требует, чтобы пользователь ввел целое число и символьную строку, а затем выводит только что принятую информацию.

Вариант на стандартном С	Вариант на С++
`#include "stdio.h"` `int i;`	`#include "iostream.h"` `int i;`

```
char temp[80], buff[80];

printf ("Введите число:");
gets(temp); i = atoi(temp);
printf ("Введите символьную
строку:");
gets (buff);
printf ("Вы ввели число: %d\n", i);
printf ("Вы ввели строку: %s\n",
buff);
```

```
char buff[80];

cout << "Введите число:";
cin >> i;
cout << "Введите символьную
строку:";
cin >> buff;
cout << "Вы ввели число:" << i <<
"\n";
cout << "Вы ввели строку:" << buff
<< "\n";
```

Эта коротенькая программа представляет собой превосходную иллюстрацию объектно-ориентированной природы C++. Здесь **cin** - это объект, воспринимающий ввод пользователя, **cout** - объект, обеспечивающий форматный вывод. Когда Вы обмениваетесь сообщениями с объектами **cin** и **cout**, форма и тип передаваемых данных определяют действия, которые требуется выполнить (*полиморфизм*), содержание выполняемых операций в вызывающей программе никак не отражается (*инкапсуляция*), и оба используемых объекта представляют собой конкретные реализации одного и того же порождающего класса **iostream** (*наследование*).

1.1. 2. ПРОСТЫЕ ТИПЫ ДАННЫХ

1.1. 2. 1.Элементы языка C++

Символы

Используются:

- a..z
- A..Z
- 0..9
- специальные символы: **. , : ; ‘ “ #**
- { } [ ] ( )
- < > & | ^ ! _ + - / * % =
- управляющие символы:

\a – сигнал тревоги (щелчок);

\b – backspace;

\f – переход на следующую страницу;

\n – переход на следующую строку;

\r – return;

\t – горизонтальная табуляция;

\v – вертикальная табуляция;

\0(ноль) – пустой символ;

\ddd – код символа в 8-ой(восьмеричной) с/с;

\xdd – код символа в 16-ой(шестнадцатеричной) с/с.

Идентификаторы

Идентификаторы – это слова, которые используются как имена переменных, функций, типов данных. Идентификатор состоит из букв, цифр, подчёркиваний и не может начинаться с цифры. Длина его произвольна.

Прописные и строчные буквы различаются.

Для разделения идентификаторов служат пробелы, табуляции, “на новую строку”, “на новую страницу”.

Инструкции (Statements)

Любое синтаксически правильно составленное предложение языка С, которое заканчивается символом ‘ **;** ‘, называется инструкцией.

Любое число инструкций, объединённых в { }, называется блоком.

Комментарии

- однострочные: //
- многострочные: /*…*/

Комментарии не могут быть вложенными.

Типы данных

Тип	**Длина в байтах**	**Диапазон**
char	8	-128..127
Int	16	-32768..32767
float	32	$3.4*10^{-38}..3.4*10^{38}$
double	64	$1.7*10^{-308}..1.7*10^{308}$

В языке С существует тип данных, который обозначает пустое значение: void. Он используется для 2-ух целей:

- для определения функций, которые не возвращают значения;
- для создания родовых(generic) указателей, то есть указателей на выделенную программой память. В дальнейшем этот указатель может быть преобразован в другой тип.

Для более точного выбора типов данных используются модификаторы типов:

- модификаторы знака:

  signed - со знаком

  unsigned - без знака

- модификаторы длины:

  short - короткие

  long - длинные

С типом char используются только модификаторы знака:

Signed char (=char)	8	-128..127
unsigned char	8	0..255

А с типом integer используются как модификаторы знака, так и модификаторы длины:

short int	8 или 16	со знаком
Short		
Signed short int		
Signed short		

unsigned short int unsigned short	8 или 16	Без знака
Signed int Signed Int	16	со знаком
unsigned int unsigned	16	Без знака
long int Long Signed long int Signed long	32 или 16	-2147483648..2147483647

С типом float модификаторы не работают.

С типом double используется только модификатор long:

Long double	80	$3.4*10^{-4932}$.. $3.4*10^{4932}$

Константы

Не изменяющиеся данные определённого типа называются константами.

Замечание:

любой символ может быть представлен последовательностью 3-ёх восьмеричных цифр или 2-мя шестнадцатеричными.

Замечание:

целые константы могут определяться в восьмеричной с/с и шестнадцатеричной с/с:

- 8-ая с/с: 0dd…d $012 = 10_{10}$
- 16-ая с/с: 0xdd…d $0x12 = 18_{10}$

Переменные

Переменная – это поименованная область памяти, в которой хранятся данные определённого типа.

Синтаксис:

<тип> <список идентификаторов>

Пример:

```
float     d;
int       i, j, k;
```

При своём объявлении значение переменной может быть инициализировано:

Пример:

```
int       i = 1;
```

Модификаторы доступа

1. const - говорит компилятору о том, что переменная не может изменяться в процессе исполнения программы:

Пример:

```
const  int     i = 1;
```

2. volatile (изменчивый) - говорит о том, что переменная может быть изменена внепрограммно.

Пример:

```
volatile int i;
```

Тип	**Пример констант**
char	‘a’ ‘\n’ ‘9’
int	1 123 -234
float	123.3 $4.3e^{-3}$ 0.3E5
double	123.3 $4.3e^{-3}$ 0.3E5 (то же, что и для float)

Допустимо одновременное использование const и volatile:

Пример:

```
volatile const int  i = 1;
```

1.1.2.2. Операции

Операторы сравнения и логические операторы

В термине *оператор сравнения* слово "сравнение" относится к значениям операндов. В термине *логический оператор* слово "логический" относится к способу, которым устанавливаются эти отношения. Поскольку операторы сравнения и логические операторы тесно связаны друг с другом, мы рассмотрим их вместе.

Операторы сравнения и логические операторы приведены в таблице

Таблица

Оператор	Обозначения в математике	Действие
Операторы сравнения		
>	>	Больше
>=	≥	Больше или равно
<	<	Меньше
<=	≤	Меньше или равно
==	=	Равно
!=	≠	Не равно
Логические операторы		
&&	∧	И
||	∨	ИЛИ
!	¬	НЕ

Оператор присваивания

Оператор присваивания (=) не обязан стоять в отдельной строке и может входить в более крупные выражения.

Составные операторы присваивания объединяют логические и арифметические операторы с оператором присваивания.

Условный оператор

Условный оператор обозначается 2-мя символами ‘?’ и ‘:’

Синтаксис:

<условие> ? <выражение1> : <выражение2>

/* если условие = true, то выполняется выражение1, иначе – выражение2 */

Пример:

```
a = 1 ? 2 : 3;    // a = 2
b = 0 ? 2 : 3;    // b = 3
c = -1 ? 2 : 3;   // c = 2
```

Управляющие инструкции

Инструкции if , if …else

Синтаксис:

if (условие)

<инструкции;>

Пример 1:

```
if (x<y)
    x++;
```

Пример 2:

```
if (x<y)
{
    x++;
    y--;
}
```

Синтаксис:

if (условие)

<инструкция1;>

else

<инструкция2;>

Инструкция switch.

Синтаксис:

```
switch (выражение)
{
case конст1: инстр.1
case конст2: инстр.2
…..
case констN: инстр.N
default: инстр.
}
```

Пример 3:

```
int a=2, b=0;
switch(a)
{
    case 0:
    case 1:b++;
    case 2:b+=2;
    case 3:b+=3;
}
//b=5;
```

Замечание 1:

инструкция с меткой default.

Замечание 2:

вариант default - в любое место.

Инструкция while

Синтаксис:

```
while (выражение)
<инструкции;>
```

Инструкция do…while

Синтаксис:

```
do
{
<инструкции;>
}
while (выражение);
```

Пример :

```
int a=-2;

do
    a++
while(a);
//a=?
```

Инструкция for

Синтаксис:

```
for (инициализация; условие ; выражение )
<инструкции;>
```

Пример:

```
int x;
For (x=0;x<100;x++)
Printf("%d", x);
```

где:

- инициализация выполняется один раз перед началом цикла.
- выражение вычисляется после каждой операции цикла.
- условие проверяется каждый раз перед выполнением инструкции.

Замечание:

инициализация, условие, выражение могут отсутствовать.

Инструкция break

Прекращает работу блока switch и циклов.

Пример:

```
int a=2, b=0;
switch(a);
```

```
{
    case 0: break;
    case 1:
            {
                b++;
                breake;
            }
}// b=2;
```

Пример:

```
int a=-2, b=1;
while (a);
{
    a++;
    b--;
    if (!b)
            break;
} //a=-1;
```

Инструкция continue

Завершает текущую операцию цикла и передает управление на вычисление условия цикла.

Пример:

```
int a=-2, b=1;
while (a)
{
    a++;
    if (!(a+b))
            continue;
    b--;
}// a=0, b=0
```

Инструкция goto

Выполняет безусловный переход к другой инструкции внутри того же файла.

Замечание:

goto используется только для выхода наверх из нескольких вложенных управляющих инструкций.

Пример:

```
while (a)
{
    …
    while (b)
    {
            …
            while(c)
            {
                    …
                    if (!c) goto m;
            }
    }
}
m: prints(“error”);
```

1.1.3. ФУНКЦИИ

1.1.3.1. Объявление, определение, вызов

Функции – это строительные блоки для всех программ на языке C++. **Общий вид функции** выглядит следующим образом.

```
тип_возвращаемого_знач имя_функции(список_параметров)
 {
 тело функции
 }
```

Тип_возвращаемого_значения определяет тип переменной, которую возвращает функция. Функция может возвращать переменные любого типа, кроме массива. В списке параметров перечисляются типы аргументов и их имена, разделенные запятыми. Если функция не имеет аргументов, ее список_параметров пуст. В этом случае скобки все равно необходимы.

В языке C с помощью одного оператора можно объявлять несколько переменных одинакового типа, разделяя их запятыми. В противоположность этому, каждый параметр

функции должен быть объявлен отдельно. Таким образом, **общий вид объявления параметров** в заголовке функции выглядит так:

f(тип имя_перем1, тип имя_перем2, ..., тип имя_перемN)

Таким образом:

f(int i, int k, int j) /* Правильно */

f(int i, k, float j) /* Неправильно */

Определим функцию, выводящую на консоль число в обратном порядке .

Опишем эту функцию в программе и вызовем несколько раз с различными параметрами (функцию необходимо описать до основной программы – функции main):

```
#include <stdio.h>
void reverse(unsigned long int number)
{
        while (number) {
                printf("%d",number % 10);
                number /= 10;
        }
        printf("\n");
}
void main()
{
                int x = 1245;
                reverse(x);
                reverse(4329);
                reverse(17328);
}
```

Приведем пример функции возвращающей минимальное из двух значений:

```
#include <stdio.h>
double minimum(double a, double b)
{
        return (a < b)? a: b;
}
void main()
{
        double x = 3.5, y = -4.6, min;
        min = minimum(x,y);
```

```
    printf("%5.2f\n", min);
    min = minimum(x,1.8);
    printf("%5.2f\n", min);
    printf("%5.2f\n", minimum(-5.4,x));
    printf("%5.2f\n", minimum(6.6,9.2));
}
```

1.1.3.2. Способы передачи параметров, их типы

В языках программирования существуют два способа передачи аргументов в подпрограмму. Первый из них известен как передача *параметров по значению*. В этом случае формальному параметру подпрограммы присваивается копия значения аргумента, и все изменения параметра никак не отражаются на аргументе.

Второй способ называется u1087 передачей *параметров по ссылке*.
При использовании этого метода параметру присваивается адрес аргумента. Внутри подпрограммы этот адрес открывает доступ к фактическому аргументу. Это значит, что все изменения, которым подвергается параметр, отражаются на аргументе.

По умолчанию в языке C++ применяется передача по значению. Как правило, это означает, что код, образующий тело функции, не может изменять аргументы, указанные при ее вызове. Но на практике такие изменения очень часты (приведенный пример тому подтверждение).

Хотя в языке C++ по умолчанию применяется передача параметров по значению, их можно передавать и по ссылке. Для этого в функцию вместо аргумента нужно передать указатель на него. Поскольку функция получает адрес аргумента, ее код может изменять значение фактического аргумента вне функции.

Указатели передаются функции как обычные переменные. Естественно, они должны быть объявлены как указатели. Рассмотрим в качестве примера функцию swap():

```
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y
    *y = temp;
}
```

Функция swap() может менять местами значения двух переменных, на которые ссылаются указатели x и y, поскольку она получает их адреса, а не копии.

Помните, что функции swap() передаются адреса аргументов. Рассмотрим фрагмент, который демонстрирует правильный вызов функции.

```
swap():
int i, j;
i = 10; j = 20;
swap(&i, &j); // Передаются адреса переменных i и j
printf("%d %d",i,j);
```

В этом примере переменной i присваивается значение 10, а переменной j – число 20. Затем функции swap() передаются адреса переменных i и j, а не их значения (для этого используется унарный оператор получения адреса &). После возврата управления из функции swap() переменные x и y оказываются переставленными.

1.1.3.3. Перегрузка функций, указатели на функцию

Вариант 1

```
int sum_int(int a, int b) // Функция sum_int
{
return a+b;
}

float sum_float(float a, float b) //Функция sum_float
{
return a+b;
}
```

Вариант 2

```
int sum(int a, int b) //Функция sum
{
return a+b;
}

float sum(float a, float b)//Функция sum
```

```
{
return a+b;
```

Вариант 3 Перегрузка функций. Число параметров разное

```
int sum(int a, int b) //Функция sum с двумя параметрами
{
return a+b;
}

float sum(int a, int b, float c)//Функция sum с тремя параметрами
{
c=1/a+a/b; //Суммирование некоторых значений
return c*3; //Возвращаем сумму некоторых вычисленных значений умноженную на 3
}
```

1.1.3.4. Рекурсия

В языке C++ функция может вызывать саму себя. Функции, вы-зывающие сами себя, называются рекурсивными. Рекурсия – это процесс определения какого-либо понятия через него же. Иногда его называют круговым определением.

Простым примером рекурсивной функции является функция factr(), вычисляющая факториал целого числа.

Факториалом называется число n, представляющее собой произведение всех целых чисел от 1 до n. Например, факториал числа 3 равен 1×2×3 = 6.

Рассмотрим рекурсивный и итеративный варианты функции factr():

```
/* Рекурсивный вариант */
unsigned long int factr(unsigned int n)
{
        if (n == 0) return 1;
        else return factr(n - 1)*n; // рекурсивный вызов
}
/* Итеративный вариант */
unsigned long int fact(unsigned int n)
{
        unsigned int t;
        unsigned long int answer;
```

```
    for(t = answer = 1; t <= n; answer *= t, t++);
    return(answer);
}
```

Итеративный вариант функции factr() выглядит проще. В нем используется цикл, счетчик которого пробегает значения от 1 до n и последовательно умножается на предыдущий результат.

Рекурсивная версия функции factr() несколько сложнее. Если функция factr() вызывается с аргументом 0, она возвращает число 1. В противном случае она возвращает значение factr(n–1)*n. Чтобы вычислить это выражение, функция factr() вызывается n–1 раз до тех пор, пока значение переменной n не станет равным 0.

В этот момент начнется последовательное выполнение операторов return, относящихся к разным вызовам функции factr().

При вычислении факториала числа 2 первый вызов функции factr() порождает второй вызов этой функции – теперь уже с аргументом, равным 1. Второй вызов порождает третий – с аргументом, равным 0. Он вернет число 1, которое будет умножено сначала на 1, а потом на 2 (исходное значение аргумента n). Попробуйте сами проследить за вызовами функции factr() при вычислении факториала числа 3. Чтобы увидеть уровень каждого вызова функции factr(), можно вставить в нее вызов функции printf() и вывести промежуточные результаты.

Когда функция вызывает саму себя, в стеке размещается новый набор ее локальных переменных и параметров, и функция выполняется с начала. Рекурсивный вызов не создает новую копию функции. Копируются лишь переменные, с которыми она работает. При каждом рекурсивном возврате управления копии переменных и параметров удаляются из стека, а выполнение программы возобновляется с места вызова функции.

Основное преимущество рекурсивных функций заключается в том, что они упрощают и делают нагляднее некоторые алгоритмы.

1.1.3.5. Математические функции

Декларации математических функций языка C++ содержатся в файле <math.h>. В последующих записях аргументы x и y имеют тип double, параметр n имеет тип int.

Аргументы тригонометрических функций задаются в радианах (2π радиан = 360°). Все приведенные математические функции возвращают значение (результат) типа double.

Таблица

Математическая функция	**Имя функции в языке С++**
$\sqrt{x}$	sqrt(x)
|x|	int abs(int n); double fabs(double x)
e^x	exp(x)
x^y	pow(x,y)
ln(x)	log(x)
$lg_{10}(x)$	log10(x)
sin(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsin(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
$sh(x)=1/2\ (e^x-e^{-x})$	sinh(x)
$ch(x)=1/2\ (e^x+e^{-x})$	cosh(x)
tgh(x)	tanh(x)
Остаток от деления x на y	fmod(x,y)
Наименьшее целое, которое >=x	ceil(x)
Наибольшее целое, которое <=x	floor(x)

1.1.4. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

1.1.4.1 . Статические и динамические массивы

Статический массив (array) – это совокупность переменных, имеющих одинаковый тип и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

Массивы могут быть одномерными и многомерными. Наиболее распространенным массивом является строка, завершающаяся нулевым байтом. Она представляет собой обычный массив символов, последним элементом которого является нулевой байт.

Массивы и указатели тесно связаны между собой. Трудно описывать массивы, не упоминая указатели, и наоборот.

Одномерные массивы

Объявление одномерного массива выглядит следующим образом:

тип имя_переменной[размер]

Как и другие переменные, массив должен объявляться явно, чтобы компилятор мог выделить память для него. Здесь тип объявляет базовый тип массива, т.е. тип его элементов, а размер определяет, сколько элементов содержится в массиве. Вот как выглядит объявление массива с именем balance, имеющего тип double и состоящего из 100 элементов:

double balance[100];

Доступ к элементу массива осуществляется с помощью имени массива и индекса. Для этого индекс элемента указывается в квадратных скобках после имени массива. Например, оператор, приведенный ниже, присваивает третьему элементу массива balance значение 12.23:

balance[3] = 12.23;

Индекс первого элемента любого массива в языке С равен нулю. Следовательно, оператор char p[10];

объявляет массив символов, состоящий из 10 элементов – от p[0] до p[9].

Следующая программа заполняет целочисленный массив числами от 0 до 99:

```
#include <stdio.h>
```

```
void main()
{
int x[100]; /* Объявление целочисленного массива, состоящего из 100 элементов
            */
int t;
/* Заполнение массива числами от 0 до 99 */
for(t = 0; t < 100; ++t)
    x[t] = t;
/* Вывод на экран элементов массива x */
for(t = 0; t < 100; ++t)
    printf("%d ", x[t]);
}
```

Объем памяти, необходимый для хранения массива, зависит от его типа и размера. Размер одномерного массива в байтах вычисляется по формуле:

количество_байт = sizeof(тип) * количество_элементов

В языке C++ не предусмотрена проверка выхода индекса массива за пределы допустимого диапазона. Иными словами, во время выполнения программы можно по ошибке выйти за пределы памяти, отведенной для массива, и записать данные в соседние ячейки, в которых могут храниться другие переменные и даже программный код. Ответственность за предотвращение подобных ошибок лежит на программисте.

Например, фрагмент программы, приведенный ниже, будет скомпилирован без ошибок, однако во время выполнения программы индекс массива выйдет за пределы допустимого диапазона:

```
int count[10], i;
/* Выход индекса массива за пределы диапазона */
for(i=0; i<100; i++)
   count[i] = i;
```

По существу, одномерный массив представляет собой список переменных, имеющих одинаковый тип и хранящихся в смежных ячейках памяти в порядке возрастания их индексов.

Ниже показано (таблица), как хранится в памяти массив a, начинающийся с адреса 1000 и объявленный с помощью оператора

```
char a[7];
```

Таблица

Элемент	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Адрес	1000	1001	1002	1003	1004	1005	1006

1.1.4.2. Инициализация массива

В языке C++ допускается инициализация массивов при их объявлении. Общий вид инициализации массива не отличается от инициализации обычных переменных:

тип имя_массива[размер1]...[размерN] = {список_значений}

Список_значений представляет собой список констант, разделенных запятыми. Тип констант должен быть совместимым с типом массива. Первая константа присваивается первому элементу массива, вторая – второму и т.д. Обратите внимание на то, что после закрывающейся фигурной скобки } обязательно должна стоять точка с запятой.

Рассмотрим пример, в котором целочисленный массив, состоящий из 10 элементов, инициализируется числами от 1 до 10:

int i[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

Здесь элементу i[0] присваивается значение 1, а элементу i[9] – число 10. Символьные массивы можно инициализировать строковыми константами:

char имя_массива[размер] = "строка";

Например, следующий фрагмент инициализирует массив str
строкой "Я изучаю C":

```
char str[10] = "Я изучаю C";
```

То же самое можно переписать иначе:

```
char str[10] = { 'Я', ' ', 'и', 'з', 'у', 'ч', 'а','ю',' ', 'C', '\0'};
```

Поскольку строки завершаются нулевым байтом, размер массива должен быть достаточным, поэтому размер массива str равен 11, хотя строка "Я изучаю C" состоит из 10 символов. Если массив инициализируется строковой константой, компилятор автоматически добавляет нулевой байт.

Инициализация безразмерного массива

При инициализации сообщений об ошибках, каждое из которых хранится в одномерном массиве, необходимо подсчитать точное количество символов в каждом сообщении:

char e1[18] = "Ошибка при чтении";

char e2[24] = "Невозможно открыть файл";

Компилятор может сам определить размер массива. Для этого не нужно указывать размер массива:

char e1[] = "Ошибка при чтении";

char e2[] = "Невозможно открыть файл";

Такой массив называется безразмерным. Инициализация безразмерных массивов, в данном случае, позволяет изменять сообщения, не заботясь о размере массивов.

Двумерные массивы

Матрицы представляются двумерным массивом или одномерным массивом, элементами которого являются одномерные массивы. Матрицу удобно трактовать как таблицу из n строк и m столбцов, записанную в виде:

$$A = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0m-1} \\ a_{10} & a_{11} & \dots & a_{1m-1} \\ \dots & \dots & \dots & \dots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,m-1} \end{bmatrix}.$$

Элементы матрицы A размерности $n \times m$ можно записать в виде $a_{ij}, i = \overline{0, n-1}, j = \overline{0, m-1}$. Здесь первый индекс i указывает положение элемента матрицы в строке, а второй индекс j – в столбце. Очевидно, что для просмотра всех элементов матрицы необходимо использование вложенных циклов.

<тип><имя_массива>[n][m]

n – максимальное количество строк.

m – максимальное количество столбцов.

Имя массива является указателем на начало массива, т.е. на элемент a[0][0]. При описании массива можно одновременно инициализировать его элементы.

int a[2][2]={1,2,3,4};

a[0][0]=1

a[0][1]=2

a[1][0]=3

a[1][1]=4

Если инициализированы не все элементы массива, то используют фигурные скобки.

```
int[][2] ={   {1},
              {2},
              {3}
          }
```

В соответствии с этим описанием элементы первой строки:

a[0][0]=1

a[1][0]=2

a[2][0]=3

Все остальные элементы получают значение равное нулю.

Динамическими в C++ могут быть не только одномерные массивы, но также и многомерные. Из тех, что входят в класс последних, рассмотрим двумерные динамические массивы (динамические матрицы). Ключевым объектом для работы с ними являются указатели.

При работе с динамическими матрицами в C++ можно использовать обычные указатели. После описания указателя, необходимо будет выделить память для хранения **NxM** элементов (**N** — число строк, **M** — число столбцов). Так выделяется память для хранения целочисленной матрицы размером **NxM**:

```
int *A, n, m;
A=(int *) calloc(N*M, sizeof(int));
```

Для выделения памяти можно использовать операцию **new:**

```
A=new int(N*M);
```

После выделения памяти, остается найти способ обратиться к элементу матрицы. Все элементы двумерного массива хранятся в одномерном массиве размером **NxM** элементов. Сначала в этом массиве расположена 0-я строка матрицы, затем 1-я и т. д. Поэтому для обращения к элементу $\mathbf{A_{i,\,j}}$ необходимо по номеру строки **i**-й номеру столбца **j** вычислить номер элемента **k** в динамическом массиве. Учитывая, что в массиве элементы нумеруются с нуля **k = iM + j**, обращение к элементу **A[i] [j]** будет таким ***(A + i*m+j)**.

В качестве примера работы с динамическими матрицами рассмотрим пример.

Пример:

Заданы две матрицы матрицы вещественных чисел **A(N, M)** и **B(N, M)**. Вычислить матрицу **C = A + B**.

Решение:

Как известно, суммой матриц одинаковой размерности называется матрица, элементы которой получаются сложением соответствующих элементов исходных матриц.

Код программы сложения матриц:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
setlocale (LC_ALL, "RUS");
int i, j, N, M; double *a,*b,*c;
//ввод размеров матрицы
cout<<"N = "; cin>>N;
cout<<"M = "; cin>>M;
//выделение памяти для матриц
a=new double[N*M];
b=new double[N*M];
c=new double[N*M];
//ввод матрицы A
cout<<"введите матрицу A"<<endl;
for (i=0; i<N; i++)
```

```
for (j=0; j<M; j++)
{
cin>>*(a+i*M+j);
}
//ввод матрицы B
cout<<"введите матрицу B"<<endl;
for (i=0; i<N; i++)
for (j=0; j<M; j++)
{
cin>>*(b+i*M+j);
}
//вычисление матрицы C=A+B
for (i=0; i<N; i++)
for (j=0; j<M; j++)
*(c+i*M+j)=*(a+i*M+j)+*(b+i*M+j);
//ввод матрицы C
cout<<"матрица C:"<<endl;
for (i=0; i<N; cout<<endl, i++)
for (j=0; j<M; j++)
cout<<*(c+i*M+j)<<"\t";
//освобождение памяти
delete []a;
delete []b;
delete []c;
system("pause");
return 0;
}
```

1.1.4.3. Строки

Определение и инициализация строк

Строкой называется массив символов, который заканчивается символом "пусто" "\0" (нулевой байт). Поэтому строка объявляется как обычный символьный массив:

```
char *str;
char str[10];
```

Литерал – строковые константы, заключенные в двойные ковычки в отличии от символов.

При инициализации строки размерность массива лучше не указывать.

Пример:

```
char *str="This is a string";
char str[]="This is a string";
```

Функции преобразования строки в числовые данные

Определены в <stdlib.h>

Синтаксис:

```
int atoi(const char *str); //ascii to int
```

Преобразует строку в число типа int.

Пример:

```
int n;
char *str="-123";
n=atoi(str); //n=-123
```

Синтаксис:

```
long int atol(const char *str);
```

Пребразует строку str в число типа long int

Пример:

```
long int n;
char *str="-123";
n=atoi(str); //n=-123
```

Синтаксис:

```
double atof(const char *str);
```

Следующие функции выполняют действия аналогичны предыдущим функциям, но предоставляют более широкие возможности:

Синтаксис:

long int strtol(const char *st, char **endptr, int base);

Преобразет строку str в число типа long int, если base=0, то представление зависит от первых двух символов строки str:

- если 1й: 1..9, то преполагается что числа в 10 с/с.
- если 1й=0, а 2й: 1..7, то в 8с/с.
- если 1й=0, 2й=x или X, то в 16с/с.

Если base = числу от 2 до 36, то это значение принимается за основание системы счислдения, и любой символ выходящий за рамки этой системы счислдения, прекращает преобразование.

Для системы счислдения от 11 до 36 используется символы A..Z и a..z

Значение аргумента endptr устанавливается функцией strtol. Это значение содержит указатель на символ, который прервал преобразование.

Пример:

```
int main()
{
    long int n;
    char *p;
    n=strtol("12a",&p,0);
    printf(n=%ld, stop %c\n",n,*p);
    n=strtol("012b",&p,0);
    printf(n=%ld, stop %c\n",n,*p);
    n=strtol("0x12z",&p,0);
    printf(n=%ld, stop %c\n",n,*p);
    n=strtol("01117",&p,2);
    printf(n=%ld, stop %c\n",n,*p);
    return 1;
}
//n=12 stop=a
//n=10 stop=b
//n=18 stop=z
//n=7 stop=7
```

Синтаксис:

unsigned long int strtoul(...);

double strtod(const char *str, char ** endptr);

Замечание 1:

все функции перечисленные в этом параграфе, прекращают свою работу при встрече символа, который не подходит под параметры этого числа.

Замечание 2:

в случае если символ представлениия превосходит допустимый диапазон, то функции atof, strol, strtoul, strtod устанавливают переменную errno в значение ERANGE(в <math.h>), при этом функции atof и strtod возвращают значение HUGE_VAL, функция strtol возвращает long_max или long_min, а функция strtoul возвращает long_max.

1.1.4.4. Структуры

Структурой называется упорядоченное множество данных. Данные входящие в структуру называются элементами (полями). Тип каждого элемента может быть произвольным.

Синтаксис:

```
 struct имя_типа
{
список элементов;
}
```

Пример:

```
struct emp
{
int empno;
char value;
double salary;
};

struct emp a; // определение структуры в языке C
emp a; // определение структуры в языке C++
```

Инициализируются также как и массивы.

Пример:

```
struct emp a={10,"Paul",200+e};
```

Так как элементы структуры описываются в блоке, то их имена принадлежат локальной области видимости (внутри этого блока). Для доступа к элементу используется оператор (**.**).

Пример:

```
a.empno=20;
strcpy(a.value,"John");
a.salaty=3000;
```

Для структур одного типа можно использовать оператор присваивания, который копирует поля структуры.

Пример:

```
struct emp b;
b=a;
```

Замечание:

элементами структуры могут быть массивы или другие структуры.

Передача структур в функции

Когда структура используется как параметр функции она передается по значению.

Пример:

```
struct emp
{
    …..
}
//печать emp;
void print(strucn emp a)
{
    printf("%d,%s,%f\n",a.empno,a.value,a.salary);
}
```

1.1.4.5. Объединения

Объединением называется область памяти, использующаяся для хранения данных разных типов. Причем одновременно в объединении могут храниться данные только одного определенного типа. Данные входящие в объединение называются его полями (элементами). Тип описывающий объединение также называется объединением.

Синтаксис:

```
union имя_типа
{
список элементов;
}
```

Пример:

```
union num
{
    int n;
    double f;
}

union num d; //d-переменная типа num (язык C)
num d; // (C++)
```

В переменной d могут храниться целые и плавающие числа, но не оба одновременно. Длина переменно d = длине max элемента (в данном случае double).

При объявлении типа объединения его можно инициализировать значением, которое имеет тип 1-го элемента объединения.

Пример:

```
union num e={1};//правильно e=1
union num g={2,7};//предупреждение о преобразовании g=2
```

Имена элементов переменной типа объединения принадлежат локальной области видимости внутри блока. Для доступа к элементам объединения используется оператор (**.**)

Пример:

```
num e,g;
e.n=1;
e.f=2,7;
```

Над объединениями одного типа возможна операция присваивания.

Пример:

```
G=e;
```

Для доступа к элементам объединения на которые указывает указатель используется оператор(->)

Пример:

```
void print(char c, union *n)
{
        switch(c)
        {
        case ',':
        {
                printf("n=%d\n",n->n);
                break;
        }
        case "f":
        {
                printf("n=%d\n",n->f);
                break;
        }
        default:
                printf("union\n");
        }
}
```

Объединения рассматриваются как вид структур, поэтому допускается вложенность объединений и структур.

1.1.4.6. Поля битов

Битовыми полями называются элементы структуры или объединения, которые определяется как последовательность битов и описывается следующим образом.

Синтаксис:

тип[имя] : длина в битах;

Тип может принимать одно из следующих значений:

- Int unsigned
- Int signed

Пример:

```
struct demo
{
    int n;
    unsigned a1:1; //1 бит
    unsigned a2:2; //2 бита
    double d;
    signed b1:3; // 3 бита
    signed b2:4; // 4 бита
    char sh[20];
    int c1:5; //5 бит
    int c2:6;//6 бит
}
```

Инициализируются битовые поля как обычные элементы структуры.

Пример:

```
struct s
{
    unsigned n1:1;
    unsigned n2:2
};
s as={0,1}; //n1=0,n2=1
```

Используются для установки флагов. Если надо обрабатывать только некоторые биты из поля, то остальные биты можно не нумеровать.

Пример:

```
struct s
{
        unsigned b1:1;
        unsigned : 8; // не нумерованное поле
        unsigned b2:1;
};
```

Доступ к битам поля осуществляется так как к обычному элементу структуры.

Пример:

```
void print_ints(struct demo s)
{
        printf("s.a1=%d\n",s->a1);
        printf(……..);
}

int main()
{
        demo s;
        s.a1=+1;
        s.b1=-1;
        s.b2=-4;
        print_ints(&s);
        return 1;
}
```

1.1.4.7. Перечисление. Структуры. Определения

В C++ переменные типа перечисления, структуры или объединения могут объявляться, используя только имя типа.

Пример

```
struct emp
{
    ...;
};
emp a;
```

Допускается объявление анонимных объединений. Доступ к элементам такого объединения выполняется по их именам, которы должны быть уникальны внутри области видимости. Глобальные анонимные объединения должны объявляться как статические.

Пример:

```
static union
{
    int n;
    float f;
};
int main()
{
    ...;
    n=1;
}
```

1.1.4.8. Сортировка массивов

При решении задачи сортировки обычно выдвигается требование минимального использования дополнительной памяти, из которого вытекает недопустимость применения дополнительных массивов.

Для оценки быстродействия алгоритмов различных методов сортировки, как правило, используют два показателя:

– количество присваиваний;

– количество сравнений.

Все методы сортировки можно разделить на две большие группы:

– прямые методы сортировки;

– улучшенные методы сортировки.

Прямые методы сортировки по принципу, лежащему в основе метода, в свою очередь разделяются на четыре подгруппы:

1) сортировка выбором;

2) сортировка вставкой;

3) сортировка заменой;

4) сортировка обменом ("пузырьковая" сортировка).

Улучшенные методы сортировки основываются на тех же принципах, что и прямые, но используют некоторые оригинальные идеи для ускорения процесса сортировки.

Прямые методы на практике используются довольно редко, так как имеют относительно низкое быстродействие. Однако они хорошо показывают суть основанных на них улучшенных методов.

Кроме того, в некоторых случаях (как правило, при небольшой длине массива и/или особом исходном расположении элементов массива) некоторые из прямых методов могут даже превзойти улучшенные методы.

1.1.4.9. Нахождение суммы элементов массива

Для обработки массивов существует несколько стандартных алгоритмов. Одним из них является процесс нахождения суммы элементов массива. Он состоит из двух этапов: присваивания сумме нулевого значения; увеличение в цикле суммы на элемент массива, номер которого совпадает с номером итерации цикла:

```
int a[9] = {34, –3, 44, 16, 53, –55, 1, –21, 2};
int i, Sum = 0;
for (i = 0; i < 9; i++)
    Sum += a[i];
```

Алгоритм нахождения суммы можно модифицировать, если, например, находить суммы только положительных элементов:

```
int i, Sum = 0;
for (i = 0; i < 9; i++)
if (a[i] > 0) Sum += a[i];
```

1.1.4.10. Нахождение произведения элементов массива

Процесс нахождения произведения элементов массива также состоит из двух этапов: присваивания произведению единичного значения; умножение в цикле произведения на элемент массива, номер которого совпадает с номером итерации цикла:

```
int i, P = 1;
```

```
for (i = 0; i < 9; i++)
P *= a[i];
```

Алгоритм нахождения произведения также можно модифицировать, если, например, находить произведение только положительных элементов:

```
int i, P = 1;
for (i = 0; i < 9; i++)
if (a[i] > 0) P *= a[i];
```

1.1.5. ФАЙЛЫ

1.1.5.1. Объявление, создание, чтение, корректировка

Файл – это структурированный и взаимозависимый объём данных, записанный на физический диск в определённом формате данных. Все без исключения пользователи компьютера хоть раз в жизни сталкивались с файлами. Благодаря файлам существуют все наши текущие программы, в том числе операционные системы. При работе с компьютером мы постоянно создаём файлы (будь то явно сохранённые нами файлы, или автоматически сохраняемые файлы операционной системы или Microsoft Office, других программ). Таким образом, если мы хотим сохранить некую информацию для дальнейшего использования после запуска программы, мы должны владеть основами создания и работы с файлами в языках программирования.

В C++ работа с файлами основана на операциях с потоками данных. Как и в других языках программирования, при записи в файл важно соблюдать уже привычные правила соответствия типов данных и записываемых данных. Рассмотрим пример:

```
#include <fstream> // файловый ввод/вывод
#include <iostream>
#include <string>
Using namespace std;
Int main ()
{
Char ch = ‘G’;
Int j = 44;
String str = ‘File1’;
Ofstream outfile (“file1.txt”); // создание файла
```

```
Outfile <<ch // вставка символа в файл
<<j
<<' '
<<'Zapis'
<<' '
<<'v'
<<' '
<<str;
Cout << "File record completed!";
Return 0;
}
```

В данном примере следует обратить внимание на несколько особенностей. Во-первых, для разделения пробелами слов или других данных следует использовать пробел, заключённый в одинарные кавычки – ' '. Во-вторых, стринговые переменные следует заполнять без пробелов. В-третьих, при операциях с числовыми данными следует в обязательном порядке разделять их пробелом или нечисловым символом, как было описано выше для текстовых переменных.

Для чтения файлов используются следующие команды:

```
Ifstream infile ("file1.txt");
Infile >>ch>>j>>str1>>str2>>str;
Cout << ch<<endl
<<j<<endl
<<str1<<endl
<<str2<<endl
<<str<<endl;
```

В данном примере переменные str1 и str2 типа string использованы для чтения введённых вручную данных 'Zapis' и 'v'. Все остальные переменные имеют тот же тип данных, что и в предыдущем примере с заполнением файла.

1.1.5.2. Типы файлов

Файлы позволяют пользователю считывать большие объемы данных непосредственно с диска, не вводя их с клавиатуры. Существуют два основных типа файлов: **текстовые и двоичные**.

Текстовыми называются файлы, состоящие из любых символов. Они организуются по строкам, каждая из которых заканчивается символом «*конца строки*». Конец самого файла обозначается символом «*конца файла*». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде.

В ***двоичных*** файлах информация считывается и записывается в виде блоков определенного размера, в которых могут храниться данные любого вида и структуры.

Для работы с файлами используются специальные типы данных, называемые *потоками.* Поток **ifstream** служит для работы с файлами в режиме чтения, а **ofstream** в режиме записи. Для работы с файлами в режиме как записи, так и чтения служит поток **fstream**.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки **iostream** и **fstream**.

Для того чтобы записывать данные в текстовый файл, необходимо:

1. описать переменную типа **ofstream**.
2. открыть файл с помощью функции **open**.
3. вывести информацию в файл.
4. обязательно закрыть файл.

Для считывания данных из текстового файла, необходимо:

1. описать переменную типа **ifstream**.
2. открыть файл с помощью функции **open**.
3. считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла.
4. закрыть файл.

Запись информации в текстовый файл

Как было сказано ранее, для того чтобы начать работать с текстовым файлом, необходимо описать переменную типа **ofstream**. Например, так:

ofstream F;

Будет создана переменная **F** для записи информации в файл. На следующим этапе файл необходимо открыть для записи. В общем случае оператор открытия потока будет иметь вид:

F.**open**(**«file»**, **mode**);

Здесь **F** — переменная, описанная как **ofstream**, **file** — полное имя файла на диске, **mode** — режим работы с открываемым файлом. Обратите внимание на то, что при указании полного имени файла нужно ставить двойной слеш. Для обращения, например к файлу **accounts.txt,** находящемуся в папке **sites** на диске **D**, в программе необходимо указать: **D:\\sites\\accounts.txt**.

Файл может быть открыт в одном из следующих режимов:

- *ios::in* — открыть файл в режиме чтения данных; режим является режимом по умолчанию для потоков **ifstream**;
- *ios::out* — открыть файл в режиме записи данных (при этом информация о существующем файле уничтожается); режим является режимом по умолчанию для потоков **ofstream**;
- *ios::app* — открыть файл в режиме записи данных в конец файла;
- *ios::ate* — передвинуться в конец уже открытого файла;
- *ios::trunc* — очистить файл, это же происходит в режиме ios::out;
- *ios::nocreate* — не выполнять операцию открытия файла, если он не существует;
- *ios::noreplace* — не открывать существующий файл.

Параметр mode может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока.

После удачного открытия файла (в любом режиме) в переменной **F** будет храниться **true**, в противном случае **false**. Это позволит проверить корректность операции открытия файла.

Открыть файл (в качестве примера возьмем файл **D:\\sites\\accounts.txt**) в режиме записи можно одним из следующих способов:

```
//первый способ
ofstream F;
F.open("D:\\sites\<strong>\</strong>accounts<strong>.</strong>txt", ios::out);

//второй способ, режим ios::out является режимом по умолчанию
//для потока ofstream

ofstream F;
F.open("D:\\game\\noobs.txt");

//третий способ объединяет описание переменной и типа поток
//и открытие файла в одном операторе

ofstream F ("D:\\game\\noobs.txt", ios::out);
```

После открытия файла в режиме записи будет создан пустой файл, в который можно будет записывать информацию.

Если вы хотите открыть существующий файл в режиме дозаписи, то в качестве режима следует использовать значение **ios::app**.

После открытия файла в режиме записи, в него можно писать точно так же, как и на экран, только вместо стандартного устройства вывода **cout** необходимо указать имя открытого файла.

Например, для записи в поток **F** переменной **a**, оператор вывода будет иметь вид:

F<<a;

Для последовательного вывода в поток **G** переменных **b**, **c**, **d** оператор вывода станет таким:

G<<b<<c<<d;

Закрытие потока осуществляется с помощью оператора:

F.close();

Чтение информации из текстового файла

Для того чтобы прочитать информацию из текстового файла, необходимо описать переменную типа **ifstream**. После этого нужно открыть файл для чтения с помощью оператора **open**.

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры, только вместо **cin** нужно указать имя потока, из которого будет происходить чтение данных.

Например, для чтения данных из потока **F** в переменную **a**, оператор ввода будет выглядеть так:

F>>a;

Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда программисту заранее известно, сколько и какие значения хранятся в текстовом файле. Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для решения данной проблемы необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла. А поможет сделать это функция **F.eof()**. Здесь **F** — имя потока функция возвращает логическое значение: **true** или **false**, в зависимости от того достигнут ли конец файла.

Следовательно, цикл для чтения содержимого всего файла можно записать так:

```
//организуем для чтения значений из файла, выполнение
//цикла прервется, когда достигнем конец файла,
//в этом случае F.eof() вернет истину
while (!F.eof())
{
//чтение очередного значения из потока F в переменную a
F>>a;
//далее идет обработка значения переменной a
}
```

1.1.5.3. Потоки

Текстовые потоки

Текстовый поток представляет собой последовательность символов. Стандарт языка C++ позволяет (но не требует) организовывать потоки в виде строк, заканчивающихся символом перехода.

В последней строке символ перехода указывать не обязательно (на самом деле большинство компиляторов языка C не завершают текстовые потоки символом перехода на новую строку). В зависимости от окружения некоторые символы в текстовых потоках могут подвергаться преобразованиям. Например, символ перехода на новую строку может быть заменен парой символов, состоящей из символа возврата каретки и прогона бумаги. Следовательно, между символами, записанными в текстовом потоке, и символами, выведенными на внешние устройства, нет взаимно однозначного со-ответствия. По этой же причине количество символов в текстовом потоке и на внешнем устройстве может быть разным.

Бинарные потоки

Бинарный поток – это последовательность байтов, однозначно соответствующая последовательности байтов, записанной на внешнем устройстве. Кроме того, количество записанных (или считанных) байтов совпадает с количеством байтов на внешнем устройстве. Однако бинарный поток может содержать дополнительные нулевые байты, количество которых зависит от конкретной реализации. Эти байты применяются для выравнивания записей,
например для того, чтобы данные заполняли весь сектор на диске.

1.1.6. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

1.1.6.1. Абстракция данных

Концепция **абстрактных типов** и **абстрактных типов данных** является ключевой в программировании. **Абстракция** подразумевает разделение и независимое рассмотрение **интерфейса** и **реализации**.

Интерфейс и **внутренняя реализация** являются определяющими свойствами объектов окружающего нас мира. Интерфейс – это средство взаимодействия с некоторым объектом. Реализация – это внутреннее свойство объекта.

Наибольший интерес представляет эффективность реализации.

Модульность и **абстракция** дополняют друг друга. Модульность предполагает скрытие деталей реализации, а абстракция позволяет специфицировать каждый модуль перед тем, как будет написана соответствующая программа.

Основная цель абстракции в программировании заключается в отделении интерфейса от реализации. Попытка усовершенствовать объект (его реализацию) пользователю, не являющемуся специалистом в этой области, приводит к отрицательному результату. В программировании запрет таких действий поддерживается механизмом **запрета доступа** или **скрытия** внутренних компонент. Принцип абстракции обязывает использовать механизмы скрытия, которые предотвращают умышленное или случайное изменение внутренней реализации.

Различают процедурную абстракцию и абстракцию данных.

Процедурная абстракция требует раздельного рассмотрения цели процедуры (например функции C/C++) и ее внутренней реализации.

Абстракция данных требует раздельного рассмотрения операций над данными и реализации этих операций. Достаточно знать, какие операции выполняет модуль, но не требуется знать, какие данные он при этом использует (они скрыты) и как в действительности выполняются эти операции. Таким образом, абстракция позволяет отделить внешнее представление модуля от его внутренней структуры. Пользователя не интересует внутренняя структура модуля, а лишь то, что этот модуль может «делать». С точки зрения же разработчика качество модуля определяется его дешевизной и эффективностью.

Абстракция данных предполагает определение и рассмотрение абстрактных типов данных (АТД), или, иначе, новых типов данных, введенных пользователем. АТД – это совокупность данных вместе с множеством операций, которые можно выполнять над этими данными.

1.1.6.2. Три принципа объектно-ориентированного программирования

Объектно-ориентированное программирование основывается на трех основных концепциях: **инкапсуляции**, **полиморфизме** и **наследовании**.

Инкапсуляция (пакетирование) представляет собой механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования. Инкапсуляция позволяет создавать объект, являющийся логическим целым, включающим данные и код для работы с этими данными. Объект обеспечивает защиту против случайной или несанкционированной модификации частных (private) составляющих его членов.

Полиморфизм – принцип (подход), обеспечивающий возможность использования одного и того же кода для решения разных задач. Полиморфизм позволяет уменьшить сложность программы посредством использования одного и того же интерфейса для задания целого класса действий. Задача выбора специфического действия (метода) в зависимости от конкретной ситуации (ко-личества и типа передаваемых аргументов) возлагается на компилятор.

Наследование представляет собой процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта. Объект, используя наследование, нуждается только в определении специфичных только для этого объекта свойств, отличающих его от других объектов этого класса.

1.1.6.3. Функции-шаблоны и классы-шаблоны

Объявление шаблона функции начинается с заголовка, состоящего из ключевого слова template, за которым следует список параметров шаблона.

```
// Описание шаблона функции
template <class X>
X min (X a, X b)
{
 return a<b ? a : b;
}
```

Ключевое слово **class** в описании шаблона означает тип, идентификатор в списке параметров шаблона **X** означает имя любого типа.

В описании заголовка функции этот же идентификатор означает тип возвращаемого функцией значения и типы параметров функции.

```
...
// Использование шаблона функции
int m = min (1, 2);
...
```

Экземпляр шаблона функции породит, сгенерированный компилятором

```
int min (int a, int b)
{
 return a<b ? a : b;
}
```

В списке параметров шаблона слово **class** может также относится к обычному типу данных. Таким образом, список параметров шаблона **<class T>** просто означает, что **T** представляет собой тип, который будет задан позднее. Так как **T** является параметром, обозначающим тип, шаблоны иногда называют параметризованными типами.

Приведем описание шаблона функции

```
template <class T>
T toPower (T base, int exponent)
{
 T result = base;
 if (exponent==0) return (T)1;
 if (exponent<0) return (T)0;
 while (--exponent) result *= base;
 return result;
}
```

Переменная **result** имеет тип **T**, так что, когда передаваемое в программу значение есть 1 или 0, то оно сначала приводится к типу **T**, чтобы соответствовать объявлению шаблона функции.

Типовой аргумент шаблона функции определяется согласно типам данных, используемых в вызове этой функции:

```
int i = toPower (10, 3);
long l = toPower (1000L, 4);
double d = toPower (1e5, 5);
```

В первом примере **T** становится типом **int**, во втором - **long**. Наконец, в третьем примере **T** становится типом **double**. Следующий пример приведет к ошибке компиляции, так как в нем используются разные типы данных:

```
int i = toPower (1000L, 4);
```

1.1.6.4. Классы

Класс - составной тип данных, элементами которого являются функции и переменные. В основу понятия класс положен тот факт, что «над объектами можно совершать различные операции». Свойства объектов описываются с помощью полей классов, а действия над объектами описываются с помощью функций, которые называются методами класса. Класс имеет имя, состоит из полей, называемых членами класса и функций - методов класса.

Описание класса имеет следующий формат:

```
class name          // name – имя класса
{
private:
        // Описание закрытых членов и методов класса
protected:
       // Описание защищенных членов и методов класса
public:
      // Описание открытых членов и методов класса
}
```

Шаблоны классов

Вы можете создавать шаблоны и для классов, что позволяет столь же красиво работать с любыми типами данных. Классическим примером являются контейнерные классы (т.е. классы, содержащие типы), например множества. Используя шаблон, можно создавать родовой класс для множеств, после чего порождать конкретные множества - цветов, строк и т.д.

Давайте сначала рассмотрим вполне тривиальный пример, просто чтобы привыкнуть к используемому синтаксису. Рассматриваемый далее пример - класс, который хранит пару значений. Принадлежащие функции этого класса передают минимальное и максимальное значения, а также позволяют определить, являются ли два значения одинаковыми. Еще раз повторю, что раз перед нами шаблон класса, то тип значения может быть почти любым.

```
template <class T>
class Pair
{
 T a, b;
 public:
 Pair (T t1, T t2);
 T Max();
 T Min ();
 int isEqual ();
};
```

Пока все выглядит также изящно, как и для шаблонов функций. Единственная разница состоит в том, что вместо описания функции используется объявление класса. Шаблоны классов становятся все более сложными, когда вы описываете принадлежащие функции класса. Вот, например, описание принадлежащей функции **Min()** класса **Pair**:

```
template <class T>
T Pair <T>::Min()
{
 return a < b ? a : b;
}
```

Чтобы понять эту запись, давайте вернемся немного назад. Если бы **Pair** был обычным классом (а не шаблоном класса) и **T** был бы некоторым конкретным типом, то функция **Min** класса **Pair** была бы описана таким образом:

```
T Pair::Min()
{
 return a < b ? a : b;
}
```

Для случая шаблонной версии нам необходимо, во-первых, добавить заголовок шаблона **template <class T>**

Затем нужно дать имя классу. Помните, что на самом деле мы описываем множество классов - семейство **Pair**. Повторяя синтаксис префикса (заголовка) шаблона, экземпляр класса **Pair** для целых типов, можно назвать **Pair<int>**, экземпляр для типа **double** - **Pair<double>**, для типа **Vector - Pair<Vector>**. Однако в описании принадлежащей функции нам необходимо использовать имя класса **Pair<T>**. Это имеет смысл, так как заголовок шаблона говорит, что **T** означает имя любого типа.

Приведем текст остальных методов класса **Pair**. Описания методов помещаются в заголовочный файл, так как они должна быть видимы везде, где используется класс **Pair**.

```
        // конструктор
template <class T>
Pair <T>::Pair (T t1, T t2) : a(t1), b(t2)
{}

        // метод Max template <class T>
T Pair <T>::Max()
{
return a>b ? a : b;
}

        // метод isEqual template <class T>
int Pair <T>::isEqual()
{
if (a==b) return 1;
return 0;
}
```

Ранее уже отмечалось, что шаблоны функций могут работать только для тех (встроенных) типов данных или классов, которые поддерживают необходимые операции. То же самое справедливо и для шаблонов классов. Чтобы создать экземпляр класса **Pair** для некоторого классового типа, например для класса **X**, этот класс должен содержать следующие общедоступные функции

X (X &); // конструктор копирования

int operator == (X)

int operator < (X);

Три указанные функции необходимы, так как они реализуют операции, выполняемые над объектами типа **T** в метода шаблона класса **Pair**.

Если вы собираетесь использовать некоторый шаблон класса, как узнать какие операции требуются? Если шаблон класса снабжен документацией, то эти требования должны быть в ней указаны. В противном случае придется читать первичную документацию - исходный текст шаблона. При этом учитывайте, что встроенные типы данных поддерживают все стандартные операции.

1.1.7.ВВОД-ВЫВОД

1.1.7.1. Компененты и функции, используемые для ввода-вывода

В C++ ввод-вывод данных производится потоками байт. Поток (последовательность байт) – это логическое устройство, которое выдает и принимает информацию от пользователя и связано с физическими устройствами ввода-вывода. При операциях ввода байты направляются от устройства в основную память. В операциях вывода – наоборот.

Ввод/вывод данных в языке C++ можно реализовать одним из двух способов: как в C, с помощью функций ввода-вывода (printf и scanf) или с использованием библиотеки iostream.h.

Второй способ удобнее, так как не требует использования шаблонов.

Для использования функций ввода/вывода необходимо подключить файл с описаниями с помощью директивы #include <iostream.h>.

Для того чтобы ввести значение переменной **x** :

```
cin>>x;
```

А если нужно последовательно вывести значения переменных **x**, **y**, **z** :

```
cin>>x>>y>>z;
```

Объект cout позволяет вывести информацию на экран.

Объект cout

Объект cout позволяет выводить информацию на стандартное устройство вывода – экран. Формат записи cout имеет следующий вид:

cout << data [<< data];

data – это переменные, константы, выражения или комбинации всех трех типов.

Простейший пример применения cout – это вывод, например, символьной строки:

cout << ”объектно-ориентированное программирование”;

cout << ”программирование на C++”.

Надо помнить, что cout не выполняет автоматический переход на новую строку после вывода информации. Для перевода курсора на новую строку надо вставлять символ ’\n’ или манипулятор endl.

cout << ”объектно-ориентированное программирование \n”;

cout << ”программирование на C++”<<endl;

Для управления выводом информации используются манипуляторы.

Манипуляторы hex и oct

Манипуляторы hex и oct используются для вывода числовой информации в шестнадцатеричном или восьмеричном представлении. Применение их можно видеть на примере следующей простой программы:

```
#include "iostream.h"
void main()
{ int a=0x11, b=4, // целые числа шестнадцатеричное и десятичное
c=051, d=8, // --//-- восьмеричное и десятичное
i,j;
i=a+b;
j=c+d;
cout << i <<' ' <<hex << i <<' '<<oct << i <<' ' <<dec << i <<endl;
cout <<hex << j <<' ' << j <<' '<<dec << j <<' ' << oct << j <<endl;
}
```

В результате выполнения программы на экран будет выведена следующая

информация:

21 15 25 21

31 31 49 61

Функция printf

Общая форма объявления функции printf:

int printf(const char *format[, argument,...]);

Параметр format является символьным массивом, содержащим выводимый текст. Кроме этого обязательного параметра, могут быть необязательные аргументы. Массив format может содержать специальные форматирующие символы, которые выполняют преобразование необязательных аргументов при выводе.

Функция printf является очень мощной функцией с богатыми возможностями форматирования вывода. В качестве первого шага в освоении ее возможностей рассмотрим Esc-последовательности, позволяющие представлять специальные символы. Esc-последовательность начинается с символа “\” — “обратная косая черта”. Esc-коды представлены в таблице.

Таблица. Esc – последовательности

Последовательность	**Десятичное значение**	**Шестнадцатеричное значение**	**Название**
\a	7	0x07	Звонок
\b	8	0x08	Возврат назад
\f	12	0x0C	Перевод страницы
\n	10	0x0A	Новая строка
\r	13	0x0D	Возврат каретки
\t	9	0x09	Табуляция
\v	11	0x0B	Вертикальная табуляция
\\	92	0x5C	Обратная черта
\'	44	0x2C	Апостроф
\"	34	0x22	Кавычка
\?	63	0x3 F	Знак вопроса
\0			Восьмеричное число, от 1 до 3 цифр
\XHHH и \xhhh		0xhhh	Шестнадцатеричное число

Функция printf имеет специальные форматирующие спецификации (символы) для вывода переменных. Общий вид этих спецификаций таков:

% [flags] [width] [.precision] [F | N | h | l | L] <символ типа>

Опции flags могут определять выравнивание, отображение знака числа при выводе, вывод десятичной точки и символов заполнения. Кроме того, эти флаги определяют префиксы для восьмеричных и шестнадцатеричных чисел. Возможные значения флагов приведены в таблице.

Таблица. Значения флагов строки формата функции printf

Символ	**Назначение**
-	Выравнивать вывод по левому краю поля
+	Всегда выводить знак числа
Пробел	Выводить пробел перед положительным числом и знак минус — перед отрицательным
#	Не влияет на вывод десятичных целых, для шестнадцатеричных чисел выводит префикс 0x или 0X, перед восьмеричными целыми выводит ноль, десятичную точку для вещественных чисел.

Спецификация width определяет минимальное количество выводимых символов. Если необходимо, используются заполнители — пробелы или нули. Когда значение для width начинается с нуля, printf использует в качестве заполнителей нули, а не пробелы. Если в качестве значения для width используется универсальный символ *, а не число, то printf подставляет на место этого символа значение, которое должно содержаться в списке аргументов. Это значение ширины поля должно предшествовать выводимому значению. Ниже приведен пример вывода числа 2, занимающего три позиции, согласно значению второго аргумента printf:

```
printf("%*d", 3, 2);
```

Спецификатор precision определяет максимальное количество выводимых цифр. В случае целого числа он определяет минимальное количество выводимых символов. Для precision также можно применить символ *, вместо которого будет подставлено значение из

списка аргументов. Это значение точности представления должно предшествовать выводимому значению. Ниже приведен пример вывода числа с плавающей точкой 3.3244 с использованием десяти символов, как это задано вторым аргументом printf:

printf("%7.*f", 10, 3.3244);

Символы F, N, h, l и L являются символами размера, переопределяющими размер по умолчанию. Символы F и N применяются с указателями, far и near соответственно. Символы h, l, и L используются для указания соответственно типов short int, long или long double.

Символам типа данных должен предшествовать форматирующий символ %. В таблице 2 мы показали возможные значения флагов форматирующей строки printf.

Функция стандартного ввода scanf()

Функция scanf() - функция форматированного ввода. С её помощью вы можете вводить данные со стандартного устройства ввода (клавиатуры). Вводимыми данными могут быть целые числа, числа с плавающей запятой, символы, строки и указатели.

Функция scanf() имеет следующий прототип в файле stdio.h:
int scanf(char *управляющая строка);

Функция возвращает число переменных которым было присвоено значение. Управляющая строка содержит три вида символов: спецификаторы формата, пробелы и другие символы. Спецификаторы формата начинаются с символа %.
Спецификаторы формата:

%c	чтение символа
%d	чтение десятичного целого
%i	чтение десятичного целого
%e	чтение числа типа float (плавающая запятая)

%h	чтение short int
%o	чтение восьмеричного числа
%s	чтение строки
%x	чтение шестнадцатеричного числа
%p	чтение указателя
%n	чтение указателя в увеличенном формате

Разделителями между двумя вводимыми числами являются символы пробела, табуляции или новой строки. Знак * после % и перед кодом формата (спецификатором формата) дает команду прочитать данные указанного типа, но не присваивать это значение.

Например:
scanf("%d%*c%d",&i,&j);

при вводе 50+20 присвоит переменной i значение 50, переменной j - значение 20, а символ + будет прочитан и проигнорирован.

В команде формата может быть указана наибольшая ширина поля, которая подлежит считыванию.
Например:
scanf("%5s",str);
указывает необходимость прочитать из потока ввода первые 5 символов. При вводе 1234567890ABC массив str будет содержать только 12345, остальные символы будут проигнорированы. Разделители: пробел, символ табуляции и символ новой строки - при вводе символа воспринимаются, как и все другие символы.

Одной из мощных особенностей функции scanf() является возможность задания множества поиска (scanset). Множество поиска определяет набор символов, с которыми будут сравниваться читаемые функцией scanf() символы. Функция scanf() читает символы

до тех пор, пока они встречаются в множестве поиска. Как только символ, который введен, не встретился в множестве поиска, функция scanf() переходит к следующему спецификатору формата. Множество поиска определяется списком символов, заключённых в квадратные скобки. Перед открывающей скобкой ставиться знак %. Давайте рассмотрим это на примере.

```
#include <stdio.h>

void main(void)
{
   char str1[10], str2[10];
   scanf("%[0123456789]%s", str1, str2);
   printf("\n%s\n%s",str1,str2);
}
```

Введём набор символов:

12345abcdefg456

На экране программа выдаст:

12345

abcdefg456

При задании множества поиска можно также использовать символ "дефис" для задания промежутков, а также максимальную ширину поля ввода.

```
scanf("%10[A-Z1-5]", str1);
```

Можно также определить символы, которые не входят в множество поиска. Перед первым из этих символов ставиться знак ^. Множество символов различает строчные и прописные буквы.

1.1.8. УПРАВЛЯЮЩИЕ КОМПОНЕНТЫ, МЕНЮ

C++ Builder представляет собой SDI-приложение, главное окно которого содержит настраиваемую инструментальную панель (слева) и палитру компонентов (справа). Помимо этого, по умолчанию при запуске C++ Builder появляются окно инспектора объектов (слева)

и форма нового приложения (справа). Под окном формы приложения находится окно редактора кода.

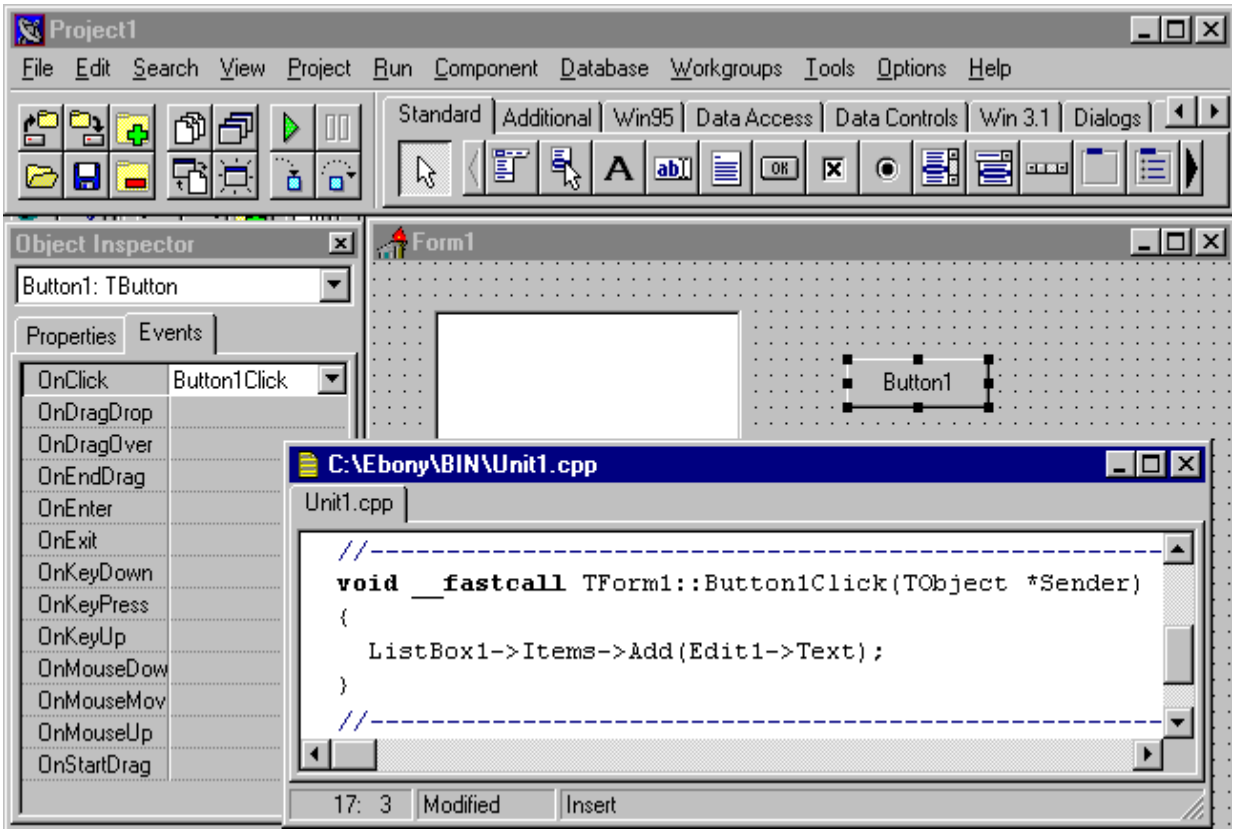


Рисунок 1.1.8.1. Среда разработки C++ Builder

Формы являются основой приложений C++ Builder. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы элементов объектов C++ Builder, называемых компонентами. Компоненты C++ Builder располагаются на палитре компонентов, выполненной в виде многостраничного блокнота. Важная особенность C++ Builder состоит в том, что он позволяет создавать собственные компоненты и настраивать палитру компонентов, а также создавать различные версии палитры компонентов для разных проектов.

Компоненты C++ Builder

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются во время выполнения точно так же, как и во время проектирования. Примерами являются кнопки и редактируемые поля. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они никогда не видны во время выполнения, но обладают определенной функциональностью (например, обеспечивают доступ к данным, вызывают стандартные диалоги Windows и др.)

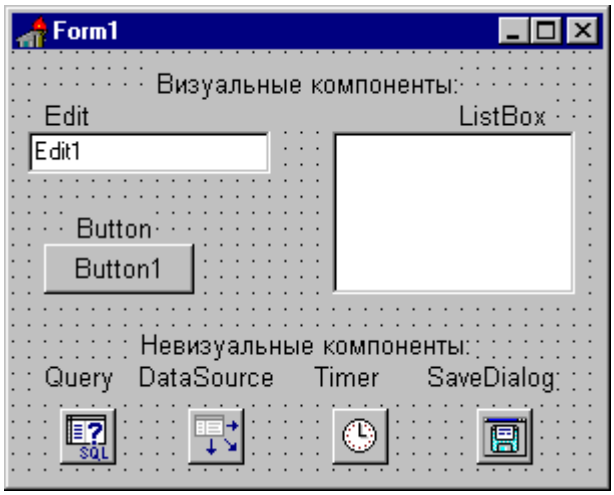


Рисунок 1.1.8.2. Пример использования видимых и невидимых компонентов

Для добавления компонента в форму можно выбрать мышью нужный компонент в палитре и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать, менять размеры и другие характеристики.

Каждый компонент C++ Builder имеет три разновидности характеристик: свойства, события и методы.

Если выбрать компонент из палитры и добавить его к форме, инспектор объектов автоматически покажет свойства и события, которые могут быть использованы с этим компонентом. В верхней части инспектора объектов имеется выпадающий список, позволяющий выбирать нужный объект из имеющихся на форме.

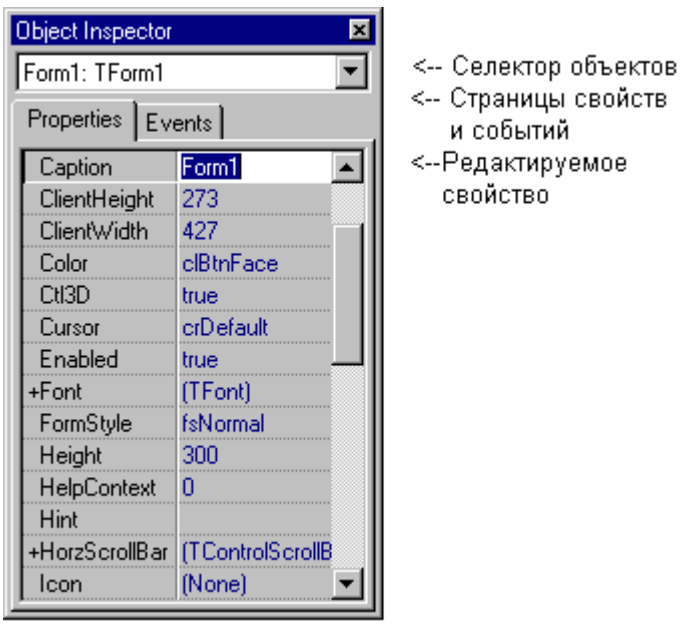


Рисунок 1.1.8.3. Инспектор объектов

Свойства компонентов

Свойства являются атрибутами компонента, определяющими его внешний вид и поведение. Многие свойства компонента в колонке свойств имеют значение, устанав иваемое по умолчанию (например, высота кнопок). Свойства компонента отображаются а странице свойств (Properties). Инспектор объектов отображает опубликованные (published) свойства компонентов. Помимо published-свойств, компоненты могут и чаще всего имеют общие (public), опубликованные свойства, которые доступны только во

время выполнения приложения. Инспектор объектов используется для установки свойств во время проектирования. Список свойств располагается на странице свойств инспектора объектов. Можно определить свойства во время проектирования или написать код для видоизменения свойств компонента во время выполнения приложения.

При определении свойств компонента во время проектирования нужно выбрать компонент на форме, открыть страницу свойств в инспекторе объектов, выбрать определяемое свойство и изменить его с помощью редактора свойств (это может быть поле для ввода текста или числа, выпадающий список, раскрывающийся список, диалоговая панель и т.д.).

События

Страница событий (Events) инспектора объектов показывает список событий, распознаваемых компонентом (программирование для операционных систем с графическим пользовательским интерфейсом, в частности, для Windows предполагает описание реакции приложения на те или иные события, а сама операционная система занимается постоянным опросом компьютера с целью выявления наступления какого-либо события). Каждый компонент имеет свой собственный набор обработчиков событий. В C++ Builder следует писать функции, называемые обработчиками событий, и связывать события с этими функциями. Создавая, обработчик того или и ого события, вы поручаете программе выполнить написанную функцию, если это событие произойдет.

Для того, чтобы добавить обработчик событий, нужно выбрать на форме с помощью мыши компонент, которому необходим обработчик событий, затем открыть страницу событий инспектора объектов и дважды щелкнуть левой клавишей мыши на колонке значений рядом с событием, чтобы заставить C++ Builder сгенерировать прототип обработчика событий и показать его в редакторе кода. При этом автоматически генерируется текст пустой функции, и редактор открывается в том месте, где следует

вводить код. Курсор позиционируется внутри операторных скобок { ... }. Далее нужно ввести код, который должен выполняться при наступлении

события. Обработчик событий может иметь параметры, которые указываются после имени функции в круглых скобках.

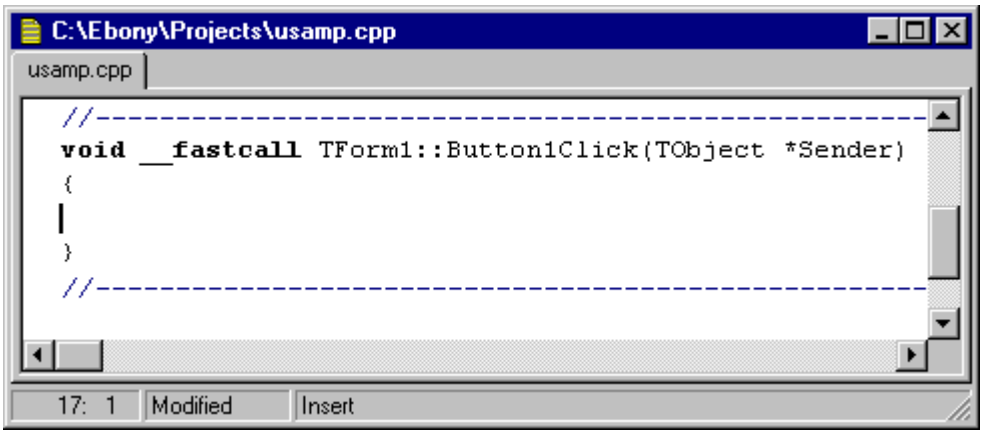


Рисунок 1.1.8.4. Прототип обработчика событий.

Методы

Метод является функцией, которая связана с компонентом, и которая объявляется как часть объекта. Создавая обработчики событий, можно вызывать методы, используя следующую нотацию: ->, например:

Edit1->Show();

Отметим, что при создании формы связанные с ней модуль и заголовочный файл с расширением *.h генерируются обязательно, тогда как при создании нового модуля он не обязан быть связан с формой (например, если в нем содержатся процедуры расчетов). Имена формы и модуля можно изменить, причем желательно сделать это сразу после создания, пока на них не появилось много ссылок в других формах и модулях.

Менеджер проектов

Файлы, образующие приложение - формы и модули - собраны в проект. Менеджер проектов показывает списки файлов и модулей приложения и позволяет осуществлять навигацию между ними. Можно вызвать менеджер проектов , выбрав пункт меню View/Project Manager. По умолчанию вновь созданный проект получает имя Project1.cpp.

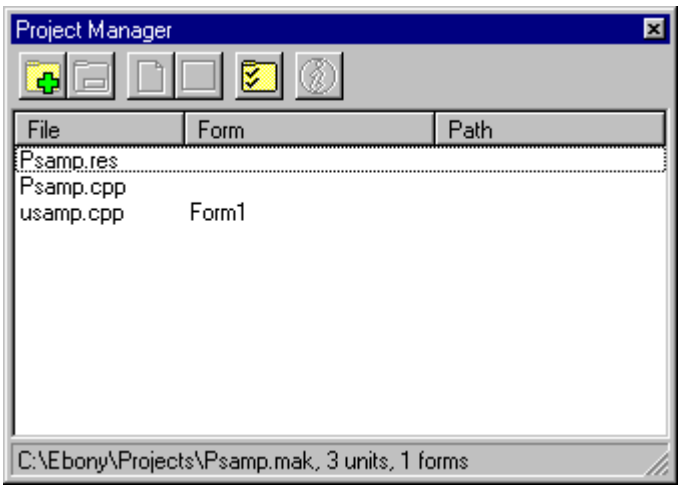


Рисунок 1.1.8.5. Менеджер проектов

По умолчанию проект первоначально содержит файлы для одной формы и исходного кода одного модуля. Однако большинство проектов содержат несколько форм и модулей.

Чтобы добавить модуль или форму к проекту, нужно щелкнуть правой кнопкой мыши и выбрать пункт New Form из контекстного меню. Можно также добавлять существующие формы и модули к проекту, используя кнопку Add контекстного меню менеджера проектов и выбирая модуль или форму, которую нужно добавить. Формы и модули можно удалить в любой момент в течение разработки проекта. Однако, из-за того, что форма связаны всегда с модулем, нельзя удалить одно без удаления другого, за исключением случая, когда модуль не имеет связи с формой. Удалить модуль из проекта можно, используя кнопку Remove менеджера проектов.

Если выбрать кнопку Options в менеджере проектов, откроется диалоговая панель опций проекта, в которой можно выбрать главную форму приложения, определить, какие формы будут создаваться динамически, каковы параметры компиляции модулей (в том числе созданных в Delphi 2.0, так как C++ Builder может включать их в проекты) и компоновки.

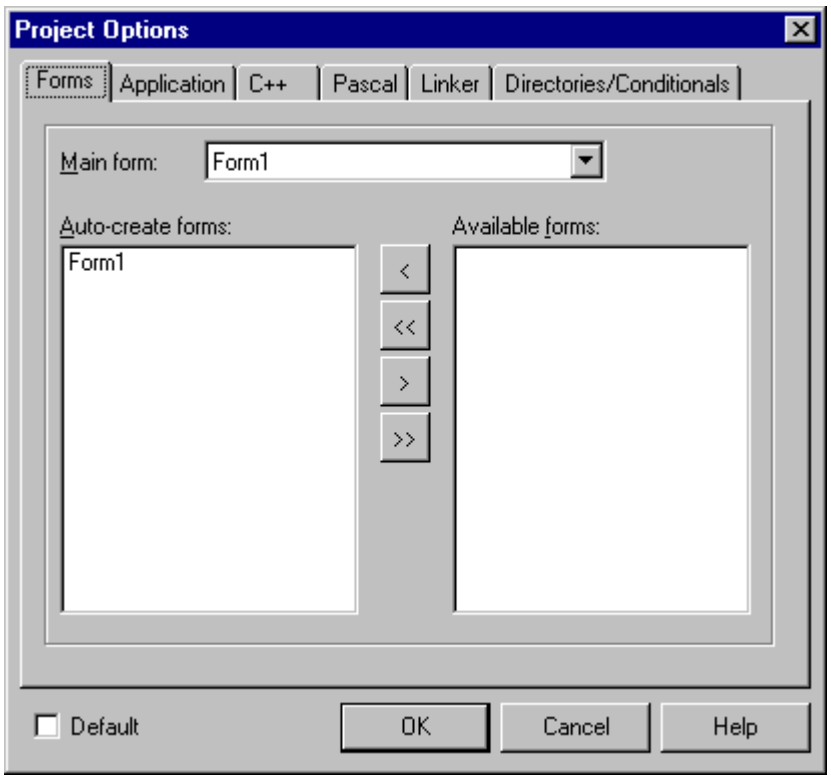


Рисунок 1.1.8.6. Установка опций проекта

Важным элементом среды разработки C++ Builder является контекстное меню, появляющееся при нажатии на правую клавишу мыши и предлагающее быстрый доступ к наиболее часто используемым командам.

Разумеется, C++ Builder обладает встроенной системой контекстно-зависимой помощи, доступной для любого элемента интерфейса и являющейся обширным источником справочной информации о C++ Builder.

Создание приложений в C++ Builder

Первым шагом в разработке приложения C++ Builder является создание проекта. Файлы проекта содержат сгенерированный автоматически исходный текст, который становится частью приложения, когда оно скомпилировано и подготовлено к выполнению. Чтобы создать новый проект, нужно выбрать пункт меню File/New Application.

C++ Builder создает файл проекта с именем по умолчанию Project1.cpp, а также make-файл с именем по умолчанию Project1.mak. При внесении изменений в проект, таких, как добавление новой формы, C++ Builder обновляет файл проекта.

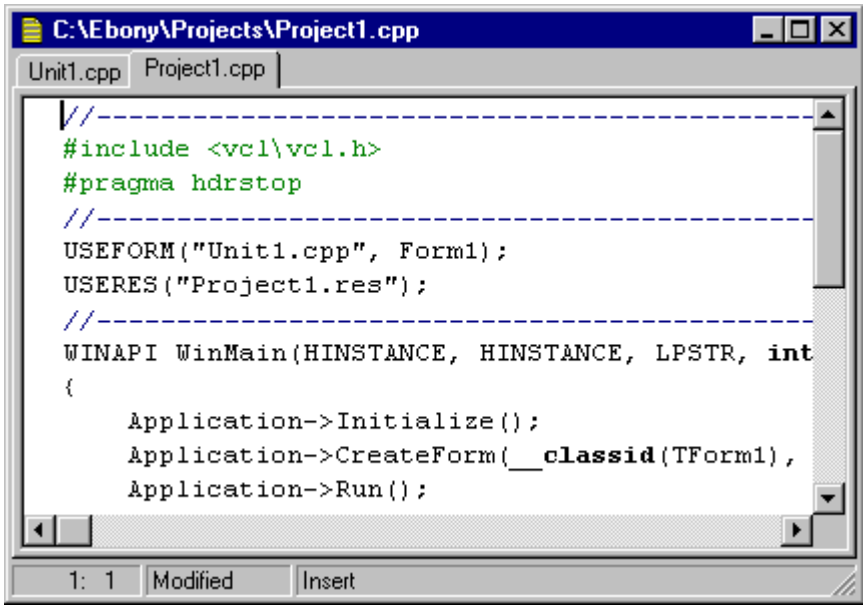


Рисунок 1.1.8.7. Файл проекта

Проект или приложение обычно имеют несколько форм. Добавление формы к проекту создает следующие дополнительные файлы:

Файл формы с расширением.DFM, содержащий информацию о ресурсах окон для конструирования формы

Файл модуля с расширением.CPP, содержащий код на C++.

Заголовочный файл с расширением .H, содержащий описание класса формы.

Когда вы добавляете новую форму, файл проекта автоматически обновляется.

Для того чтобы добавить одну или более форм к проекту, выберите пункт меню File/New Form. Появится пустая форма, которая будет добавлена к проекту. Можно воспользоваться пунктом меню File/New, выбрать страницу Forms и выбрать подходящий шаблон из репозитория объектов.

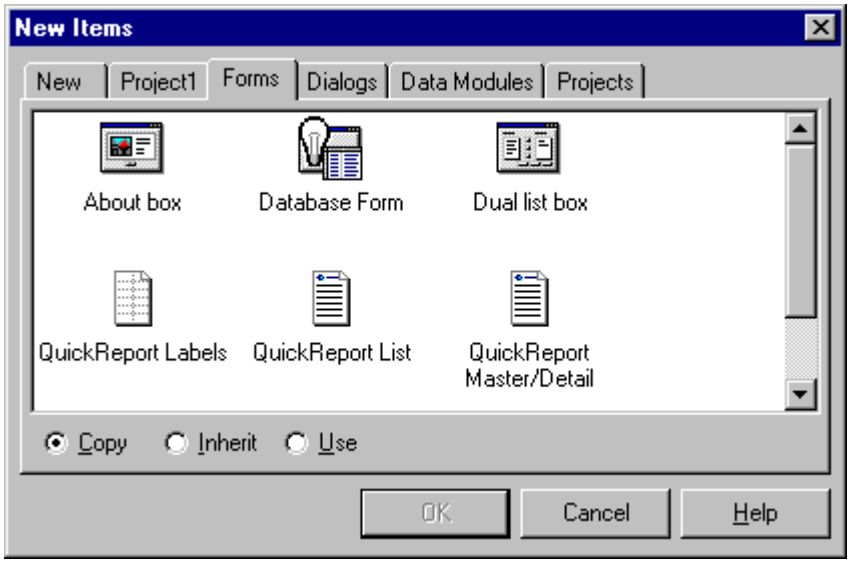


Рисунок 1.1.8.8. Шаблоны форм

Для того, чтобы просто откомпилировать текущий проект, из меню Compile нужно выбрать пункт меню Compile. Для того, чтобы откомпилировать проект и создать исполняемый файл для текущего проекта, из меню Run нужно выбрать пункт меню Run. Компоновка проекта является инкрементной (перекомпилируются только изменившиеся модули).

Если при выполнении приложения возникает ошибка времени выполнения, C++ Builder делает паузу в выполнении программы и показывает редактор кода с курсором, установленным на операторе, являющемся источником ошибки. Прежде чем делать необходимую коррекцию, следует перезапустить приложение, выбирая пункт меню Run из контекстного меню или из меню Run, закрыть приложение и лишь, затем вносить изменения в проект. В этом случае уменьшится вероятность потери ресурсов Windows.

Компонент TForm

Как и любой другой визуальный компонент, форма имеет свойства, методы и события, общие для всех визуальных компонентов.

Свойства формы

Свойства формы могут встречаться у многих компонентов. В инспекторе объекта, отображаются не все свойства объекта (только те, что попадают в секцию published). Все свойства объекта, в том числе, отображаемые в инспекторе объекта, можно посмотреть в разделе справочной системы для данного объекта (надо при выделенном объекте нажать к вишу <F1>).

Caption— сюда помещается название формы. По умолчанию первая форма проекта получает имя Form1, вторая — Form2 и т.

ActiveControl — это свойство определяет, какой компонент, помещеный в форму, в данный момент является активным (или, как говорят, имеет фокус).

AutoScroll — это свойство определяет, будут ли автоматически появляться полосы прокрутки формы, если в ней не будут помещаться компоненты (т. е. чтобы их увидеть, надо будет форму "прокрутить"). Если значение этого свойства true, то полосы прокрутки будут появляться, а если false — то не будут.

Autosize — если значение свойства true, то форма автоматически принимает прежние размеры, как бы вы ее не растягивали. При свойстве, равном false, форма "позволяет" себя растягивать.

BorderStyle— свойство задает появление и поведение границ формы; можно ли мышью менять размеры формы, когда приложение находится в режиме исполнения.

BorderWidth — здесь задается в пикселях величина отступа координатной сетки формы от границ окна формы, т. е. фактически размеры формы можно изменить за счет изменения координатной сетки, задав ее отступ от границ окна. По умолчанию величина отступа равна нулю, т. е. форма, занимает все пространство окна.

Canvas — обеспечивает пользователю возможность рисования в форме. Задает на плоскости формы битовый холст, на котором и можно рисовать Это свойство само является классом и имеет свои свойства и методы обеспечивающие рисование различных фигур (точек, линий, прямоугольников и т. п.).

Color— это свойство задает цвет поля формы. Цвет можно выбрать из раскрывающегося списка, который появится, если нажать кнопку в поле этого свойства. В этом списке следует установить стрелками или мышью строку подсветки на нужный цвет и щелкнуть мышью или нажать клавишу <Enter>. Кроме того, цвет можно выбрать из палитры цветов, которая появится, если дважды щелкнуть кнопкой мыши в поле.

Constraints — здесь указываются ограничительные величины на размеры компонента (в данном случае — формы). Это свойство рекомендуется не изменять, т. к. его значения связаны со свойствами Align (выравнивание объекта относительно контейнера его содержащего) и Anchors (якоря), которые задают

привязку компонента к родителю. Компонент поддерживает свое текущее месторасположение относительно угла родительского окна, даже тогда, когда родительское окно изменяет размеры. В этом случае компонент удерживает свою позицию по отношению к тому углу, к которому он привязан. Якоря и задают эту привязку.

Cursor — в раскрывающемся списке можно выбрать форму курсора. Форма курсора будет действовать над областью всего компонента. Выбор курсора аналогичен выбору цвета, но при двойном щелчке будут появляться новые формы курсора.

Docksite — свойство задает способность компонента стать участком стыковки для других компонентов.

DragKind — этим свойством обладают только некоторые компоненты. Оно задает способ перетаскивания компонента: если это свойство имеет значение dkDrag, то компонент участвует в операции перетаскивания, если — dkDock, то в операции стыковки. Если задать значение свойства DragKind равным dkDock, а значение свойства DragMode— dmAutomatic, то *в режиме исполнения программы* компонент приобретает способность перемещаться по форме: он становится потомком участка стыковки (в нашем случае — формы), и его можно перетаскивать мышью. По умолчанию DragMode = dmManual. Это означает, что компонент можно заставить двигаться, применяя обработчики событий щелчка кнопкой мыши. Если свойство DragKind имеет значение dkDock, то с компонентом могут происходить

следующие метаморфозы: он может быть захвачен участком стыковки, в этом случае компонент займет весь участок. Уже измененный в размере компонент можно вытащить из участка стыковки и двигать по форме. Е ли на компоненте щелкнуть, он может принимать вид обычного окна Windows с линейкой захвата и кнопкой закрытия окна на ней.

Enabled (группа **Action)**— задает право доступа к компоненту: значение true означает, что доступ разрешен, false — запрещен. В случае с формой значение свойства, равное false, приведет к блокированию формы: после компиляции ничто в ней не будет реагировать на мышь, даже закрыть форму будет невозможно.

Font — задает характеристики шрифта формы. Все компоненты, расположенные в форме, унаследуют ее шрифт. Чтобы задать значение свойства Font, нужно два раза щелкнутьна свойстве, после чего откроется диалоговое окно выбора характеристик шрифта.

FormStyle — определяет характеристики формы: является ли она одной из форм так называемого MDI-приложения. Мы сейчас говорим о создании приложений со стандартным документным интерфейсом (SDI) и поэтому свойства FormStyle и DefaultMonitor должны оставаться заданными средой по умолчанию.

Hint — подсказка. Она появляется, как только мышь зависает над компонентом, но при условии, что значение свойства ShowHint (показать подсказку) установлено в true.

HorzScrollBar, VertScrollBar — это составные свойства, которые позволяют задать характеристики вертикальной и горизонтальной полос прокрутки формы.

KeyPreview — это свойство определяет, может ли форма получить событие с клавиатуры раньше, чем активный компонент в ней. Если значение свойства KeyPreview установлено равным true, то события с клавиатуры в форме возникают раньше, чем в активном компоненте (активный компонент выбирается из списка свойства ActiveControl). Если значение свойства KeyPreview установлено равным false, события с клавиатуры возникают только в активном компоненте. Навигационные клавиши (<Tab>, клавиши со стрелками и т. д.) не действуют на свойст KeyPreview, потому что они не генерируют событий клавиатуры. По умолчанию свойство KeyPreview имеет значение false. Это означает, что форма и другой компонент, который может в данный момент обрабатывать события, возникающие при вводе с клавиатуры, имеют одинаковые события. Но при задании свойству KeyPreview значения true эти события можно перехватывать в форме и выполнять определенные действия до того, как будут выполнены действия по этим же событиям в активном компоненте

Menu— если в форму поместить компонент MainMenu, то его имя попадет в это свойство, и при запуске формы главное меню будет готово к выполнению своих команд.

ModalResult — это свойство используется для закрытия формы, когда она открыта в модальном режиме. По умолчанию ModalResult имеет значение mrNone.

Name — задается имя формы.

PopupMenu — если в форму поместить компонент PopupMenu (всплывающее меню), то его имя попадет в это свойство, и при запуске формы меню появится, если нажать правую кнопку мыши. Меню будет готово к выполнению своих команд.

Position — определяет размер и размещение формы.

Tag— сюда помещается некоторое целое число, которое можно потом доставать из формы во время исполнения программы.

Visible — если это свойство имеет значение false, то форма становится невидимой при исполнении программы.

События формы

onActivate — возникает, когда форма активизируется.

onClick — возникает при щелчке мышью в форме.

onClose — возникает, когда форма закрывается.

onCreate — возникает, когда форма создается.

onDeactivate — возникает, когда форма перестаетбыть активной.

onKeyDown — возникает, когда пользователь нажимает некоторую клавишу. Это событие может возникать в ответ на нажатие любых клавиш, включая функциональные (<F1>—<F12>) и комбинации с клавишами <Shift>, <Alt> и <Ctrl>.

onKeyPress — возникает, когда пользователь нажимает некоторую (только одну) клавишу, кроме функциональной.

onKeyUp—- возникает, когда пользователь отпускает клавишу, которая до этого была нажата. Это событие возникает и тогда, когда до этого были нажаты и комбинации клавиш с <Shift>, <Alt> и <Ctrl>, а также функциональные клавиши.

onMouseDown— возникает при нажатии любой кнопки мыши, а также при комбинации клавиш <Shift>, <Ctrl> и <Alt> и кнопок мыши. В обработчике этого события X и Y — это координаты в пикселах указателя мыши.

onMouseMove— возникает, когда пользователь двигает указатель мыши, пока он находится над объектом (в нашем случае — над формой). В обработчике этого события X и Y— это координаты в пикселах указателя мыши.

onMouseUp — возникает, когда пользователь отпускает кнопку мыши, которая была нажата на компоненте. В обработчике этого события X и Y — это координаты в пикселах указателя мыши.

onPaint — возникает, когда начинается прорисовка формы.

onShow — возникает, когда форма появляется на экране.

Методы формы

Форма имеет большое количество методов, которое можно посмотреть справочной системе, нажав клавишу <F1> при активной форме.

Close() — закрывает форму. Если закрывается главная форма, приложение закрывается.

Hide() — свойство visible устанавливается в false и форма становится невидимой.

Print() — форма печатается.

Release() — форма разрушается и память, занятая ею, освобождается.

SetFocus() — делает форму активной: свойства visible и Enabled становятся равными true: форма становится видимой и доступной.

Show() — показать форму: в этом случае свойство visible устанавливается в true, и форма перемещается поверх всех форм на экране.

ShowModal() — показать форму в модальном режиме. Когда форма показана в модальном режиме, приложение не может выполняться, пока форма не будет закрыта. Чтобы закрыть такую форму, надо установить ее свойство ModaiResult в ненулевое значение.

Компонент TButton

TButton (Кнопка) позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. Поместив TButton на форму, по двойному щелчку можно создать заготовку обработчика события нажатия кнопки.

Свойства TButton

Cancel — если значение свойства true, то, когда пользователь нажимает клавишу <Esc>, включается обработчик события OnClick. Так как приложение может иметь более одной кнопки Отмена (Cancel), форма вызывает событие OnClick только для первой по порядку (в смысле TabOrder) видимой кнопки.

Caption — сюда помещается название кнопки (только в одну строку).

Cursor — это свойство аналогично одноименному свойству формы.

Enabled — это свойство аналогично одноименному свойству формы.

Font — аналогично одноименному свойству формы.

ModalResult— определяет, как кнопка закрывает модальную форму, на которой она находится. Установка этого компонента— легкий путь к закрытию модальной формы. Когда кнопку нажимают, свойство ModalResult формы, на которой находится кнопка,

устанавливается в то значение, которое имеет это свойство в кнопке. Поэтому перед закрытием модальной формы с помощью кнопки нет необходимости предварительно присваивать свойству ModaiResuit формы ненулевое значение: достаточно выбрать такое значение в свойстве ModalResult кнопки, и оно передастся одноименному свойству формы, в которой расположена эта кнопка. Поэтому достаточно выполнить Form1->Close() и модальная форма закроется, как ей положено.

PopupMenu— это свойство обеспечивает кнопке дополнительные возможности. Если в форму поместить всплывающее меню, то его имя можно включить в свойство PopupMenu. Тогда в режиме исполнения достаточно щелкнуть на кнопке правой кнопкой мыши (после щелчка левой кнопкой мыши будет запускаться обработчик кнопки), как появится всплывающее меню рядом с кнопкой, команды которого можно выполнять.

TabOrder — аналогично одноименному свойству формы.

TabStop — с помощью этого свойства можно позволить или запретить использовать клавишу <Tab>. Если свойство TabStop имеет значение true, переход будет происходить в порядке, определенном свойством TabOrder. Если значение TabStop— false, пользователь не сможет перейти к следующему компоненту, нажав клавишу <TaЬ>.

Visible— это свойство обеспечивает видимость или невидимость компонента в режиме исполнения приложения (аналогично одноименному свойству формы).

WordWrap— если это свойство установить в true, то название кнопки (свойство caption) может быть многострочным.

События *TButton*

onClick — возникает, когда на кнопке щелкают мышью.

onEnter — возникает, когда кнопка получает фокус ввода, т. е. становится активной: ее можно нажимать.

onExit — возникает, когда кнопка теряет фокус ввода.

Методы *TButton*

Кнопка имеет большое число методов, главным образом унаследованных от своих классов-предков. Рассмотрим только некоторые из методов.

Click() — имитирует нажатие кнопки. Выполнение этой функции назначает свойству ModalResult формы значение свойства ModalResult кнопки вызывает событие OnClick.

Focused() — когда значение этой функции true, кнопка имеет входной фокус (т. е. активна: ее можно нажимать). В противном случае пользовате не может применять кнопку.

Hide () — прячет кнопку: делает ее невидимой.

SetFocus () — делает кнопку активной: ее можно нажимать.

Show() — показывает кнопку: присваивает ее свойству visible значе true.

Компонент *TPanel*

Компонент находится в разделе **Standard** палитры компонентов. Панель — это компонент, как и форма, являющийся контейнером, в который помещаются другие компоненты. Панели обеспечивают общее (родовое) поведение, введенное в классе-родителе TCustomPanel.

TCustomPanel — это базовый класс для всех панельных компонентов. Он используется в качестве базового класса для определения объектов, которые включают в себя другие объекты. Панельные компоненты могут содержать в себе другие компоненты, объединяя их в единое целое. При перемещении панели такие компоненты перемещаются вместе с ней. Когда панель выравнивается в форме с помощью свойства Align, она сохраняет те же относительные позиции по отношению к форме, даже если форма перерисовывается. Например, панель может быть выровнена так, что всегда останется в вершине формы, даже когда пользователь меняет фигуру и размер формы. Это свойство панели делает ее полезной для расположения на ней компонентов, которые действуют как линейки инструментов или линейки состояний.

Свойства *TPanel*

У панели имеются свойства, определяющие ее обрамление.

BevelInner— определяет стиль внутренней кромки. Если свойство имеет значение bvLowred, то внутренняя кромка будет опущена вниз, если — bvNone, то внутренней кромки у панели не будет, а если значение этого свойства — bvRaised, то внутренняя кромка панели будет поднята.

BevelOuter — определяет стиль внешней кромки. Может принимать те же значения, что и BevelInner.

BevelWidth — ширина кромок в пикселах

BorderWidth — расстояние в пикселах между внутренней и внешней кромками.

BorderStyle — стиль внешнего обвода. Обвод одиночной линией —bsSingle, нет внешнего обвода — bsNone.

Align— размещает панель в форме в соответствии со значениями этого свойства: alTop— привязывает панель к верхней кромке формы, alBottom— к нижней, alLeft — к

левой, alRight — к правой, alNone — панель не привязана ни к какой кромке формы, и ее можно двигать мышью, alClient — панель принимает размеры формы.

Alignment — определяет, как выравнивается свойство caption, в котором задается название панели, относительно самой панели: по центру, по левому краю или по правому краю. Например, если использовать панель как линейку состояния для вывода подсказки (подсказку надо помещать в свойство Caption панели), можно присвоить свойству Alignment значение taLeftJustify, и текст подсказки разместится с левой стороны панели.

События *TPanel*

onEnter (возникает, когда панель становится активной — получает, как говорят, фокус ввода),

onExit (когда панель теряет фокус ввода— перестает быть активной),

onClick (когда на панели происходит щелчок мышью).

Во всех этих ситуациях управление в приложении передается на обработчик соответствующего события, и можно написать в обработчике необходимые команды, отражающие реакцию на произошедшее событие.

Методы *TPanel*

Собственных методов у компонента нет, а только унаследованные. Все методы можно посмотреть в справочной системе, нажав клавишу <F1> при активном компоненте.

Компонент *TLabel*

Компонент находится в разделе **Standard** палитры компонентов. Он выводит в форму текст, который пользователь в режиме исполнения приложения может редактировать. Этот текст может использоваться как метка к другому компоненту и устанавливать фокус этого компонента, когда пользователь наажимает "горячую" клавишу.

Свойства *TLabel*

Alignment — задает способ расположения (выравнивания) текста, записываемого в поле свойства Caption: будет ли текст выравниваться по левой или правой границе поля, или же по центру поля.

FocusControl — это свойство имеет раскрывающийся список, содержащий компоненты, которые могут быть связаны с меткой и получать от нее фокус ввода (например, TEdit, TButton и др.). Выбрав из списка один из таких компонентов, который

должна помечать метка, мы в ее тексте в свойстве Caption в некотором месте пишем символ амперсанд (&) перед символом, который и станет "горячей" клавишей, когда приложение начнет исполняться. Если в момент исполнения приложения нажать клавишу с этим символом, то связанный с меткой по свойству FocusControl компонент станет активным (но только при условии, что свойство ShowAccelChar имеет значение true).

Align— свойство задает способ размещения метки в окне формы. Из выпадающего списка можно выбрать конкретное значение свойства: метка может быть размещена в нижней части формы, слева, справа и т. д.

Layout — размещение текста, заданного в свойстве Caption, в поле метки. Из раскрывающегося списка можно выбрать, как будет расположен текст в метке: в верхней части ее поля, в центре или в нижней части поля (метку можно растягивать за маркеры по ее краям, когда она активна).

Transparent — если некоторый компонент будет расположен под меткой, то он может быть невидим. Чтобы этого не происходило, надо сделать метку прозрачной, т. е. установить это свойство в true.

1.1. 9. РАБОТА С ТЕКСТОМ

1.1.9.1. Стандартные процедуры и функции для работы со строками

Ввод строк

В программах, которые мы рассматривали, операторы потокового вывода выводили строковые константы; C++ поддерживает потоковый вывод для строк как специального не-предопределенного типа данных. (Можно сказать, что это было сделано по требованию масс.) Операции и синтаксис для вывода строковых переменных остаются прежними. При вводе строк операция извлечения из потока ” не всегда будет работать так, как вы ожидаете, поскольку строки часто содержат пробелы, которые игнорируются оператором ввода; поэтому вместо оператора ввода вам нужно использовать функцию getline. Эта функция вводит заданное количество символов.

Функция getline

Перегруженная функция getline объявляется следующим образом:

```
istreams getline( signed char *buffer,
int size,
char delimiter = '\n') ;
istreams getline( unsigned char *buffer,
int size,
char delimiter = '\n') ;
istream& getline( char *buffer,
int size,
char delimiter = '\n') ;
```

Параметр buffer является указателем на строку, в которую помещаются вводимые символы. Параметр size задает максимальное количество вводимых символов. Параметр delimeter определяет символ-ограничитель, при появлении которого ввод символов прекращается прежде, чем будут введены все size символов. Параметр delimeter имеет аргумент по умолчанию, равный '\n'. В случае ввода символов с клавиатуры этот символ появляется в потоке ввода при нажатии клавиши

Пример:

```
#include <iostream.h> //см. файл Ex01.cpp
int main()
{
char name[80] ;
cout “ "Enter your name: ";
cin.getline(name, sizeof(name) - 1);
cout “ "Hello " “ name “ ", how are you?";
return 0;
}
```

Функции, объявленные в STRING. H

Стандартная библиотека для работы со строками содержит много полезных функций (объявляемых в STRING.H), разработанных коллективными усилиями многих программистов на C. В файлах заголовка STDIO.H и IOS-TREAM.H также имеются прототипы строковых функций. Комитетом ANSI/ISO C++ предложен класс для работы со строками. Строки этого класса больше похожи на строки в языках Pascal и BASIC. (Мы познакомимся с классами в День 8, а со строковым классом в День 11.) Этот раздел будет посвящен рассмотрению некоторых (ни в коей мере не всех) функций, объявленных в STRING.H.

Некоторые функции из STRING.H имеют несколько версий. Дополнительные версии этих функций, имеющих в имени префиксы _f, f или _ работают с указателями типа far. Этих версий вы не встретите в плоской, 32-битной модели памяти компилятора Borland.

Присвоение значений строкам

C++ поддерживает два способа присвоения значений строкам. Вы можете присвоить строковой переменной строковую константу, произведя инициализацию при объявлении строки. Этот метод прост: требуется операция присваивания и строковая константа.

Инициализация строки

Общий метод инициализации строки:

char stringVar[stringSize] = stringLiteral;

Пример:

```
char a3tring[81] = "Borland C++ 5 in 21 days";
char Named = "Rene Kinner";
```

Второй способ присвоить значение строке — это вызвать функцию, которая копирует содержимое одной строки в другую, — не забывая при этом и нуль-символ. Эта функция называется strcpy. Она предполагает, что копируемая строка оканчивается символом NUL и прекращает копирование, как только встретит этот символ.

Функция strcpy

Прототип функции strcpy таков:

char* strcpy(char *target, const char *source);

Функция копирует строку source в строку target. Функция предполагает, что целевая строка имеет размер, достаточный для того, чтобы вместить содержимое строки-источника.

Пример:

```
char name[41] ;
strcpy(name, "Borland C++ 5");
```

Переменная name содержит строку "Borland C++ 5".

Функция strdup

Функция strdup копирует одну строку в другую, при этом отводит необходимое количество памяти для целевой строки

.

Прототип функции strdup таков:

char* strdup(const char *source);

Функция копирует строку source и возвращает указатель на строку-копию.

Пример:

```
char *string1 = "            ";
char *string2;
string2 = strdup(string1);
```

После того, как будет отведено необходимое количество памяти для строки string2, строка string1 будет скопирована в строку string2.

Функция strncpy

Библиотека строковых функций предлагает также функцию strncpy, копирующую заданное количество символов из одной строки в другую.

Прототип функции strncpy таков:

char * strncpy(char *target, const char *source, size_t num);

Функция копирует num символов из строки source в строку target. Функция не выполняет ни усечение, ни заполнение строки.

Пример:

```
char str1[] = "Pascal";
char str2[] = "Hello there";
strcnpy(strl, str2, 5);
```

Переменная strl содержит строку "Hellol". Заметьте, что символ ‘l’ строки-приемника, следующий за скопированной частью строки, сохранился.

Определение длины строки

При работе со строками часто бывает нужно знать длину строки.

Функция strlen

Функция strlen возвращает количество символов в строке, в которое не включается нуль-терминатор.

Прототип функции strncpy таков:

size_t strlen (const char *string) ,

Функция strlen возвращает длину строки string. size_t — это имя, приписанное типу unsigned int оператором typedef.

Пример:

```
char str[] = "1234567890";
size_t i;
i = strlen(str),
```

Переменной i будет присвоено значение 10.

Конкатенация строк

Операция конкатенации используется достаточно часто, когда новая строка получается объединением двух или более строк.

Присоединить одну строку к другой можно функцией strcat.

Функция strcat

Конкатенация строк означает их последовательное присоединение друг к другу.

Прототип функции strcat таков:

char *strcat(char *target, const char *source) ;

Функция добавляет к содержимому целевой строки содержимое строки-источника и возвращает указатель на целевую строку. Функция предполагает, что целевая строка может вместить содержимое объединенной строки.

Пример:

```
char string[81] ;
strcpy(string, "Turbo");
strcat (string, " C++");
```

Переменная string содержит строку "Turbo C++".

Функция strncat

Функция strncat добавляет к содержимому целевой строки указанное количество символов из строки-источника.

Прототип функции strcat :

char *strncat(char *target, const char *source, size_t num);

Функция добавляет к содержимому целевой строки num символов из строки-источника и возвращает указатель на целевую строку.

Функция strcmp

Функция strcmp выполняет сравнение двух строк с учетом регистра символов.

Прототип функции strcmp:

int strcmp(const char *strl, const char *str2);

Функция сравнивает строки strl и str2. Возвращает в качестве результата сравнения целую величину:

< 0 когда strl меньше, чем str2;

= 0 когда strl равна str2;

> 0 когда strl больше, чем str2.

Пример

```
char stringl[] = "Borland C++";
char string2[] = "BORLAND C++";
i = strcmp(string1, string2);
```

В последнем операторе переменной i присваивается положительное значение, так как string1 больше string2 (ASCII-коды символов в нижнем регистре больше ASCII-кодов символов в верхнем.)

Функция stricmp

Функция stricmp выполняет сравнение двух строк, не учитывая регистра символов.

Прототип функции stricmp:

int stricmp(const char *strl, const char *str2);

Функция сравнивает строки strl и str2, не делая различия между символами в нижнем и верхнем регистре. Возвращает в качестве результата сравнения целую величину:

< 0 когда strl меньше, чем str24

= 0 когда strl равна str24

- 0 когда strl больше, чем str2.

Пример:

```
char string1[] = "Borland C++";
char string2[] = "BORLAND C++";
int i = strcmp(string1, string2);
```

В последнем операторе переменной i присваивается значение 0, так как string1 и string2 отличаются друг от друга только регистром символов.

Функция strncmp выполняет сравнение заданного количества символов двух строк с учетом регистра символов.

Функция strncmp

Прототип функции strncmp:

int strncmp(const char *strl, const char *str2, size_t num);

Функция сравнивает первые num символов строк strl и str2. Возвращает в качестве результата сравнения целую величину:

< 0 когда strl меньше, чем str2;

= 0 когда strl равна str2;

- 0 когда strl больше, чем str2.

Функция strnicmp

Функция strnicmp выполняет сравнение заданного количества символов двух строк без учета регистра символов.

Прототип функции strnicmp :

int strnicmp(const char *strl, const char *str2, size_t num);

Функция сравнивает первые num символов строк strl и str2, не делая различия в регистре символов. Возвращает в качестве результата сравнения целую величину:

< 0 когда strl меньше, чем str2;
= 0 когда strl равна str2;
> 0 когда strl больше, чем str2.

Преобразование строк

Функция strlwr

Прототип функции strlwr:

char* strlwr (char *source)

Функция преобразует символы верхнего регистра в символы нижнего регистра в строке source. Другие символы не затрагиваются. Функция возвращает указатель на строку source.

1.1.10. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ

1.1.10.1. Рисование элементарных фигур

Графические объекты - это объекты, с помощью которых осуществляется вывод на экран изображений. Их использование в приложениях для создания графических изображений возможно на различных уровнях - в этом раздели будет рассмотрена работа с объектами с использованием функций GDI.

Графических объектов не так уж и много:

- Точка. Изображение можно рисовать точками. Наиболее простой метод, так как для него достаточно вызова лиш одной функции SetPixel() (ровно как и для чтения GetPixel()). В принципе, точка не является графическим объектом, так как она не существует как объект и рассматривается здесь лишь как одна из возможностей получения изображения.
- Перо. Изображение можно рисовать перьями (как корандашом на листе бумаги). Перо обладает толщиной, цветом и типом линии (сплошная, прерывистая, точечная и т. п.). Перед использованием перо создается как объект.
- Кисть. Предопределенный системой или созданный программистом набор пикселов, который, как единое целое, может быть отображен на экране монитора.

Кисти используются в функциях для заполнения внутренних областей замкнутых фигур и фонов окон (аналогично сплошной закраске). Перед использованием кисть создается как объект.

- Растровые изображения. Набор байт, содержащий значения цветов и информацию о координатах для отображения на экране пикселов, в совокупности составляющих изображение. Это картинки, фоны, графические элементы (кнопки, меню, иконки) и т. п. Перед использованием растровые изображения могут быть созданы программно или использоваться как заранее созданные и хранящиеся в файлах, ресурсах и т.д.
- Шрифты. Это либо набор пикселов (растровое изображение) для матричных шрифтов, либо набор кривых, описывающих контур отображаемых букв для векторных шрифтов (например шрифты True Type). Перед использованием

шрифты могут быть созданы программно или используются заранее подготовленные и установленные в системе шрифты.

Получение изображения перерисовкой пикселей

Изменить один контекста устройства дает возможность вызов функции SetPixel():

WINGDIAPI COLORREF WINAPI SetPixel(HDC hdc,int x,int y,COLORREF color);

Первый аргумент хэндл контекста устройства (как и во всех последующих примерах - поэтому далее этот параметр не будет поясняться).

x и y - координаты прорисовываемого пикселя.

color - цвет пикселя, как совокупность трех цветов (red - красного, зеленого - green и синего - blue, или RGB -значение цвета). Каждая из этих микроточек может иметь значение, соответствующее интенсивности свечения от 0 до 255 (максимальная яркость). Таким образом может быть определено почти 17 миллионов цветов. Число отображаемых цветов может быть меньше и зависит от видиокарты компьютера и установок Windows.

Можно задать цвет как 16 ричное число:

0xx00bbggrr

Макрос RGB, возвращающий цвет пикселя, определен как:

```
#define RGB(r, g, b) ((COLORREF) (((BYTE)(r)
                | ((WORD) ((BYTE)(g)) <8 ))
                | (((DWORD) (BYTE) (P)) <16 )))
```

Пример использования функции SetPixel(). В примере точками рисуется зеленая линия длиной 100 пикселей.

```
HDC  hDc = GetDC(Handle);
for(int i=0; i < 100;i++)
 SetPixel(hDc,100,100,RGB(0,255,0)):
ReleaseDC(Handle,hDc);
```

В принципе для рисования линий более подходят функции работы с перьями - о чем речь пойдет ниже.

Перо и его использование для рисования графических примитивов

Перо перед использованием создается с помощью функции CreatePen. Эта функция создает логическое перо, которое задает указанный стиль, ширину, и цвет пера. Перо перед использованием выбирается в контекст устройства.

```
HPEN CreatePen
(
 int      PenStyle,     // стиль пера
 int      nWidth,       // ширина
 COLORREF  crColor          // цвет
);
```

Стиль пера задают константы:

PS_SOLID - сплошная линия.
PS_DASH - штриховая линия.
PS_DOT - пунктирная линия.
PS_DASHDOT - штрих пунктирная линия.
PS_DASHDOTDOT - чередующиеся черточки и двойные точки.
PS_NULL - невидимое перо.
PS_INSIDEFRAME - перо для линий, выводимых внутри рамки закрытых форм (например, Ellipse, Rectangle, RoundRect, Pie, и Chord функции).

Перо может быть создано также функцией CreatePenIndirect().

```
HPEN CreatePenIndirect
(
 CONST LOGPEN *lplgpn  // указатель на структуру LOGPEN
);
```

Функция SelectObject выбирает объукт (в данном случае перо) в указанный контекст устройства. Новый объект заменяет предыдущий объект того же самого типа и возвращает HDC заменяемого объекта.

```
HGDIOBJ SelectObject
(
 HDC hdc,        // дескриптор контекста устройства
 HGDIOBJ hgdiobj  // дескриптор объекта который вибирается
);
```

Для рисования перьями используются функции:

- bool MoveToEx(HDC hdc,int x1,int y1,LPPOINT lpPoint). Перемещает точку начала рисования линии в указанные координаты. LPPOINT lpPoint - адрес старой текущей позиции
- bool LineTo(int x2,int y2). Рисует линию начиная с текущей позиции, заданной функцией MoveTo до указанных координат.
- bool Rectangle(HDC hdc,int x1,int y1,int x2,int y2). Рисует прямоугольник, размер которого определяется координатами верхнего (x1,y1) и нижнего (x2,y2) угла. Используется текущее перо, а для заполнения текущая кисть.
- bool Ellipse(HDC hdc,int x1,int y1,int x2,int y2).Рисует элипс, вписанный в прямоугольник, размер которого определяется координатами верхнего (x1,y1) и нижнего (x2,y2) угла. элипс заполнен белым цветом и обведен линией пера контекста устройства. Используется текущее перо, а для заполнения текущая кисть.
- bool RoundRect(HDC hdc,int x1,int y1,int x2,int y2,int x3,int y3). Рисует прямоугольник с закругленными краями, размер которого определяется координатами верхнего (x1,y1), нижнего (x2,y2) угла и координатами скругления (x3,y3). Используя текущее перо, а для заполнения текущая кисть.
- bool Arc(HDC hdc,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4); Рисует элиптическую дугу, логически ограниченную прямоугольником, размер которого определяется координатами верхнего (x1,y1) и нижнего (x2,y2) угла. Непосредственно дуга определяется дополнительными двумя точками (x3,y3,x4,y4). Первая - начало дуги - находится на пересечении эллипса, частью которого является дуга, и прямой, проходящей через центр прямоугольника и точку начала дуги. Вторая - конец дуги - определяется аналогично. Дуга прорисовывается против часовой стрелки. Ограничивающий прямоугольник должен быть не длиннее и не шире 32767.
- bool ArcTo(HDC hdc,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4). Полностью аналогична функции ArcTo, за исключением того, что запоминается как текущая позиция пера последняя точка дуги.

- bool Pie(HDC hdc,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4); Рисует сектор элипса, логически вписываемый в прямоугольник, размер которого определяется координатами верхнего (x1,y1) и нижнего (x2,y2) угла, а координаты начальной и конечной точек (x3,y3,x4,y4), аналогично функции Arc(). Ограничивающий прямоугольник должен быть не длиннее и не шире 32767. Используется текущее перо, а для заполнения текущая кисть.
- bool Chord(HDC hdc,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4). Рисует сегмент элипса (область, ограниченную пересечением эллипса и линии), логически вписываемый в прямоугольник, размер которого определяется координатами верхнего (x1,y1) и нижнего (x2,y2) угла, а координаты ограничительной линии (x3,y3,x4,y4). Используется текущее перо, а для заполнения текущая кисть.
- bool Polyline(HDC hdc,CONST POINT *lppt,int cPoints); Функция рисует ломаную линию по массиву точек, на который указывает lppt и число точек из этого массива равно cPoints.

Отрезки прямых рисуются текущим пером. Фигуры, образованные сегментами, не закрашиваются.

- bool Polygon(HDC hdc,CONST POINT *lppt,int cPoints); Функция рисует многоугольник, состоящий из двух или больше вершины, связанных прямыми линиями по массиву точек, на который указывает lppt и число точек из этого массива равно cPoints.. Используется текущее перо и заполнение текущей кистью.

Эти функции работают только в Windows NT:

- bool AngleArc(HDC hdc,int x1,int y1,int r,float fStartAngle, float fSweepAngle); Рисует линию сегмента и дугу с центром радиуса дуги в точке x1,y1 и радиусом r (радиус круга в логических модулях, всегда положителен). fStartAngle - стартовый угол в градусах относительно оси X, fSweepAngle - определяет конечный угол в градусах относительно стартового угла. Фигура не заполнена.
- bool PolyPolyline(HDC hdc,CONST POINT *lppt, CONST DWORD *lpdwPolyPoints,DWORD cCount); lppt - указатель на массив структур типа POINT. Каждая структура в массиве идентифицирует точку в логическом пространстве. lpdwPolyPoints - указывает на массив переменных, определяющих число точек в массиве lppt для соответствующей полилинии. Значение каждого элемента должно быть больше или равно двум. cCount - определяет количество элементов в массиве

lpdwPolyPo. Отрезки прямых рисуются текущим пером. Фигуры, образованные сегментами, не закрашиваются.

- bool PolyPolylineTo(HDC hdc,CONST POINT *lppt, CONST DWORD *lpdwPolyPoints,DWORD cCount); Полностью аналогична функции PolyPolyline, за исключением того, что запоминается как текущая позиция пера последняя точка линии.
- bool PolyPolygon(HDC hdc,CONST POINT *lppt, CONST DWORD *lpdwPolyPoints,DWORD cCount); Параметры аналогичны параметрам функции PolyPolyline. Функция рисует ряд замкнутых многоугольников. Каждый многоугольник рисуются текущим пером и закрашен текущей кистью. Многоугольники могут накладываться друг на друга.
- bool PolyBlezier(HDC hdc,CONST POINT *lppt,DWORD cPoints); Выводит одну или большее Bezier сплайнов (кривых Блейзера). Эти кривые задаются началом, концом линии и промежуточными точками, определяющие изгиб. В массиве точек первая и четвертая используются как конечные, вторая и третья как промежуточные. Для следущей линии необходимо еще три точки - четвертая точка первой линии является начальной для второй. Рисунок не заполнен. Эта функция выводит линии, используя текущее перо
- bool PolyBlezierTo(HDC hdc,CONST POINT *lppt,DWORD cPoints); Полностью аналогична функции PolyBezier, за исключением того, что запоминается как текущая позиция пера последняя точка линии.
- bool PolyDraw(const POINT* lpPoints,const BYTE* lpTypes,int nCount); Функция составляет множество сегментов строки и Кривых Безье. lpPoints - указатель на массив структур данных POINT, который содержит оконечные точки для каждого сегмента линии и контрольных точек для каждого Bezier сплайна.lpTypes - указатель на массив, который определяет, как каждая точка в lpPoints массиве используется. Значения могут быть одним из следующего:
    - PT_MOVETO - Определяет, что эта точка начинает новый рисунок. Эта точка становится новой текущей позицией.
    - PT_LINETO - Определяет, что линия должна быть выведена от текущей позиции до этой точки, которая затем становится новой текущей позицией.
    - PT_BEZIERTO - Определяет, что эта точка - контрольная точка или последняя точка для Bezier сплайна.

- PT_BEZIERTO - точек всегда должно быть три. Текущая позиция определяет отправную точку для Bezier сплайна. Первые две PT_BEZIERTO точки - контрольные точки, а третья PT_BEZIERTO точка - точка окончания. Точка окончания становится новой текущей позицией. Если не имеется трех последовательных точек PT_BEZIERTO, результат ошибка. PT_LINETO или тип PT_BEZIERTO может быть объединен с другими константами, используя поразрядный оператор OR указывая, что соответствующая точка - последняя точка в рисунке, и рисунок закрыт.
- PT_CLOSEFIGURE - Определяет, что рисунок автоматически закрыт после того, как PT_LINETO или тип PT_BEZIERTO для этой точки выполнен. Линия выведена от этой точки до самой последней точки MoveTo или PT_MOVETO. Этот флажок объединен с типом PT_LINETO для строки, или с типом PT_BEZIERTO для Bezier сплайна, используя поразрядный OR оператора. Текущая позиция установлена к отметке окончания заключительной строки.

- nCount - Определяет общее число точек в lpPoints массиве и число байтов в lpTypes массиве.

Функция модифицирует текущую позицию. Нарисованные замкнутые фигуры не заполняются.

Следующие примеры показывают использование некоторых описанных функций.

```
void __fastcall
TForm1::Button1Click(TObject *Sender)
{
 HPEN      hPen,hPenOld;
 //Получаем контекст окна приложения
 HDC       hDc = GetDC(Handle);
 //Создаем перо сплошное, толщиной 2,красное
 hPen=CreatePen(PS_SOLID,2,RGB(255,0,0));
 //Выбираем перо в контекст устройства и запоминаем старое
 hPenOld = SelectObject(hDc,hPen);
 //Рисуем линию
 MoveToEx(hDc,0,0,NULL);
 LineTo(hDc,Width,Height);
```

```
//Рисуем  прямоугольник
Rectangle(hDc,50,50,100,100);
//Рисуем  прямоугольник со скругленными углами
RoundRect(hDc,150,150,400,400,50,50);
//Рисуем элипс
Ellipse(hDc,100,100,300,300);
//Рисуем  сектор элипса
Pie(hDc,0,0,200,200,0,0,0,100);
//Рисуем дугу
Arc(hDc,0,0,250,250,125,0,50,0);
//Рисуем  сегмент, замкнутый дугой
AngleArc(hDc,100,150,50,0,180);
//Рисуем  сегмент элипса
Chord(hDc,0,0,200,200,0,0,10,200);
//Рисуем ломаную линию
TPoint      tPoint[3];
tPoint[0] = Point(100,500);
tPoint[1] = Point(150,400);
tPoint[2] = Point(100,300);
Polyline(hDc,tPoint,3);
//Рисуем  кривые Блейзера
TPoint tPoints[7];
tPoints[0]=TPoint(0,0);
tPoints[1]=TPoint(800,30);
tPoints[2]=TPoint(0,40);
tPoints[3]=TPoint(550,400);
tPoints[4]=TPoint(350,200);
tPoints[5]=TPoint(550,400);
tPoints[6]=TPoint(0,500);
PolyBezier(hDc,tPoints,6);
//Возвращаем все на свои места
SelectObject(hDc,hPenOld);
DeleteObject(hPen);
DeleteObject(hPenOld);
ReleaseDC(Handle,hDc);
```

```
 //Или
 DeleteDC(hDc);
}
```

Для того, чтобы рисовать не в форме приложения,а на экране дисплея, достаточно в предыдущем примере получить контекст экрана монитора.

```
HDC   hDc=CreateDC("DISPLAY",NULL,NULL,NULL);
```

Несмотря на то, что многие функции работают только в Windows NT, но в BorlandC++ Builder доступ к ним имеется через свойство Canvas компонентов.

Кисти, их создание и применение

Кисти используются в Windows в основном для заливки внутренних областей.

Чтобы использовать кисть ее необходимо создать с помощью функций CreateSolidBrush, CreateDIBPatternBrush, CreateHatchBrush, CreatePatternBrush... и затем, как и для пера, выбрать их в контекст отображения, используя функцию SelectObject. После применения средств рисования их можно удалить с помощью функции DeleteObject.

Основные функции для создания кисти, следующие:

Функция CreateSolidBrush создает логическую кисть, которая имеет указанный цвет.

```
HBRUSH CreateSolidBrush
(
 COLORREF crColor // цвет кисти
);
```

Функция CreateHatchBrush создает логическую кисть, которая имеет указанный образец штриховки и цвет.

```
HBRUSH CreateHatchBrush
(
 Int fnStyle,    // стиль штриховки
 COLORREF clrref // цвет
);
```

FnStyle определяет стиль штриховки кисти. Этот параметр может иметь одно из следующих значений:

HS_BDIAGONAL штриховка 45 градусов слева направо;
HS_CROSS горизонтальная и вертикальная штриховка(квадратиками);
HS_DIAGCROSS Штриховка сеточкой 45 градусов;
HS_FDIAGONAL С 45 градусами справа на лево;
HS_HORIZONTAL горизонтальная штриховка;
HS_VERTICAL вертикальная штриховка.

Функция CreatePatternBrush создает логическую кисть с указанным растровым изображением.

```
HBRUSH CreatePatternBrush
(
 HBITMAP hbmp // Хэндл изображения
);
```

В Win9x кисти, созданные как точечные рисунки в формате растрового изображения больше чем 8x8 пикселы не поддерживаются. Если точечный рисунок больше, то используется только его часть.

Цвет однобитных кистей при значении бита 1 соответствует цвету текста и фона контекста устройства, 0 текущему цвету контекста.

1.1.11. ЯЗЫК JAVA

Создание языка Java в **1995 году** — это действительно один из самых значительных шагов вперед в области разработки сред программирования за последние 20 лет. Язык HTML (Hypertext Markup Language — язык разметки гипертекста) был необходим для статического размещения страниц во “Всемирной паутине” WWW (World Wide Web). Язык Java потребовался для качественного скачка в создании интерактивных продуктов для сети Internet.

Три ключевых элемента объединились в технологии языка Java и сделали ее в корне отличной от всего, существующего на сегодняшний день.

- Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.
- Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.
- Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

Давайте поближе познакомимся со всеми этими тремя аспектами, но сначала – история создания.

1.1.11.1. Апплеты Java

Апплеты — это маленькие приложения, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются на месте, как часть документа HTML. Когда апплет прибывает к клиенту, его доступ к ресурсам ограничен.

Ниже приведен исходный код канонической программы HelloWorld, оформленной в виде апплета:

```
import java.awt.*;
import java.applet.*;
public class HelloWorldApplet extends Applet {
public void paint(Graphics g) {
g.drawString("Hello World!", 20, 20);
} }
```

Этот апплет начинается двумя строками, которые импортируют все пакеты иерархий java.applet и java.awt. Дальше в нашем примере присутствует метод paint, замещающий одноименный метод класса Applet. При вызове этого метода ему передается

аргумент, содержащий ссылку на объект класса Graphics. Последний используется для прорисовки нашего апплета. С помощью метода drawString, вызываемого с этим объектом типа Graphics, в позиции экрана (20,20) выводится строка “Hello World”.

Для того, чтобы с помощью браузера запустить этот апплет, нам придется написать несколько строк html-текста.

<applet code="HelloWorldApplet" width=200 height=40>
</applet>

Вы можете поместить эти строки в отдельный html-файл, либо вставить их в текст этой программы в виде комментария и запустить программу appletviewer с его исходным текстом в качестве аргумента.

Тег HTML <Applet>

Тег <applet> используется для запуска апплета как из HTML-документа, так и из программы appletviewer. Программа appletviewer выполняет каждый найденный ей тег <applet> в отдельном окне, в то время как браузеры позволяют разместить на одной странице несколько апплетов. Синтаксис тэга <APPLET> в настоящее время таков :

<APPLET

CODE = appletFile
OBJECT = appletSerialFile
WIDTH = pixels
HEIGHT = pixels
[ARCHIVE = jarFiles]
[CODEBASE = codebaseURL]
[ALT = alternateText]
[NAME = appletInstanceName]
[ALIGN = alignment]
[VSPACE = pixels]
[HSPACE = pixels]
>
[< PARAM NAME = AttributeNamel VALUE = AttributeValuel >]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue2 >]

[HTML-текст, отображаемый при отсутствии поддержки Java]
</APPLET>

CODE = appletClassFile

CODE — *обязательный* атрибут, задающий имя файла, в котором содержится оттранслированный код апплета. Имя файла задается относительно codebase, то есть либо от текущего каталога, либо от каталога, указанного в атрибуте CODEBASE. В Java 1.1 вместо этого атрибута может использоваться атрибут OBJECT.

OBJECT = appletClassSerialFile

Указывает имя файла, содержащего сериализованный апплет, из которого последний будет восстановлен. При запуске определяемого таким образом апплета должен вызываться не метод init(), а метод start(). Для апплета необходимо задать либо атрибут CODE, либо атрибут OBJECT, но задавать эти атрибуты одновременно нельзя.

WIDTH = pixels

HEIGHT = pixels

WIDTH и HEIGHT — *обязательные* атрибуты, задающие начальный размер видимой области апплета.

ARCHIVE = jarFiles

Задает список jar-файлов (разделяется запятыми), которые предварительно загружаются в Web-броузер. В этих архивных файлах могут содержаться файлы классов, изображения, звуки и любые другие ресурсы, необходимые апплету. Для создания архивов используется утилита JAR, синтаксис вызова которой напоминает вызов команды TAR :

```
c:\> jar cf soundmap.jar *.class image.gif sound.wav
```

Очевидно, что передача сжатых jar-файлов повышает эффективность работы. Поэтому многие средства разработки (Lotus JavaBeans, Borland JBuilder) уже имеют средства для публикации апплетов в виде jar-файлов.

CODEBASE = codebaseURL

CODEBASE — *необязательный* атрибут, задающий базовый URL кода апплета, являющийся каталогом, в котором будет выполняться поиск исполняемого файла апплета (задаваемого в признаке CODE). Если этот атрибут не задан, по умолчанию используется каталог данного HTML-документа. CODEBASE не обязательно должен указывать на тот же узел, с которого был загружен HTML-документ.

ALT = alternateAppletText

Признак ALT — *необязательный* атрибут, задающий короткое текстовое сообщение, которое должно быть выведено в том случае, если используемый браузер распознает синтаксис тега <applet>, но выполнять апплеты не умеет. Это не то же самое, что HTML-текст, который можно вставлять между <applet> и </applet> для браузеров, вообще не поддерживающих апплетов.

NAME = appletInstanceName

NAME — *необязательный* атрибут, используемый для задания имени для данного экземпляра апплета. Присвоение апплетам имен необходимо для того, чтобы другие апплеты на этой же странице могли находить их и общаться с ними. Для того, чтобы получить доступ к подклассу MyApplet класса Applet с именем “Duke”, нужно написать:

MyApplet a = getAppletContext().getApplet("Duke");

После того, как вы получили таким образом дескриптор именованного экземпляра апплета, вы можете вызывать его методы точно так же, как это делается с любым другим объектом.

ALIGN = alignment

ALIGN — *необязательный* атрибут, задающий стиль выравнивания апплета. Этот атрибут трактуется так же, как в теге IMG, возможные его значения — LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

VSPACE = pixels

HSPACE = PIXELS

Эти *необязательные* атрибуты задают ширину свободного пространства в пикселях сверху и снизу апплета (VSPACE), и слева и справа от него (HSPACE). Они трактуются точно так же, как одноименные атрибуты тега IMG.

PARAM NAME = appletAttribute1 VALUE = value1

Этот тег дает возможность передавать из HTML-страницы апплету необходимые ему аргументы. Апплеты получают эти атрибуты, вызывая метод getParameter(), описываемый ниже.

Передача параметров

getParameter(String)

Метод getParameter возвращает значение типа String, соответствующее указанному имени параметра. Если вам в качестве параметра требуется значение какого-либо другого типа, вы должны преобразовать строку-параметр самостоятельно. Вы сейчас увидите некоторые примеры использования метода getParameter для извлечения параметров из приведенного ниже примера:

```
<applet code=Testing width=40 height=40>
<param name=fontName value=Univers>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
```

Ниже показано, как извлекается каждый из этих параметров:

```
String FontName = getParameter("fontName");
String FontSize = Integer.parseInt(getParameter("fontSize"));
String Leading = Float.valueOf(getParameter("leading"));
String PaidUp = Boolean.valueOf(getParameter("accountEnabled"));
```

Контекст апплета

getDocumentBase и getCodeBase

Возможно, Вы будете писать апплеты, которым понадобится явно загружать данные и текст. Java позволяет апплету загружать данные из каталога, в котором

располагается HTML-документ, запустивший апплет (база документа - getDocumentBase), и из каталога, из которого был загружен class-файл с кодом апплета (база кода - getCodeBase).

AppletContext и showDocument

AppletContext представляет собой средства, позволяющие получать информацию об окружении работающего апплета. Метод showDocument приводит к тому, что заданный его параметром документ отображается в главном окне браузера или фрейме.

Отладочная печать

Отладочную печать можно выводить в два места: на консоль и в статусную строку программы просмотра апплетов. Для того, чтобы вывести сообщение на консоль, надо написать:

System.out.println("Hello there, welcome to Java");

Сообщения на консоли очень удобны, поскольку консоль обычно не видна пользователям апплета, и в ней достаточно места для нескольких сообщений. В браузере Netscape консоль Java доступна из меню Options, пункт "Show Java Console".

Метод showStatus выводит текст в статусной области программы appletviewer или браузера с поддержкой Java. В статусной области можно вывести только одну строку сообщения.

Порядок инициализации апплета

Ниже приведен порядок, в котором вызываются методы класса Applet, с пояснениями, нужно или нет переопределять данный метод.

init

Метод init вызывается первым. В нем вы должны инициализировать свои переменные.

start

Метод start вызывается сразу же после метода init. Он также используется в качестве стартовой точки для возобновления работы после того, как апплет был остановлен. В то время, как метод init вызывается только однажды — при загрузке апплета, start

вызывается каждый раз при выводе HTML-документа, содержащего апплет, на экран. Так, например, если пользователь перейдет к новой WWW-странице, а затем вернется назад к странице с апплетом, апплет продолжит работу с метода start.

paint

Метод paint вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод paint.

update

Используемый по умолчанию метод update класса Applet сначала закрашивает апплет цветом фона по умолчанию, после чего вызывает метод paint. Если вы в методе paint заполняете фон другим цветом, пользователь будет видеть вспышку цвета по умолчанию при каждом вызове метода update — то есть, всякий раз, когда вы перерисовываете апплет. Чтобы избежать этого, нужно заместить метод update. В общем случае нужно выполнять операции рисования в методе update, а в методе paint, к которому будет обращаться AWT, просто вызвать update.

stop

Метод stop вызывается в тот момент, когда браузер покидает HTML-документ, содержащий апплет. При вызове метода stop апплет еще работает. Вы должны использовать этот метод для приостановки тех подпроцессов, работа которых необязательна при невидимом апплете. После того, как пользователь снова обратится к этой странице, вы должны будете возобновить их работу в методе start.

destroy

Метод destroy вызывается тогда, когда среда (например, браузер Netscape) решает, что апплет нужно полностью удалить из памяти. В этом методе нужно освободить все ресурсы, которые использовал апплет.

Перерисовка

Возвратимся к апплету HelloWorldApplet. В нем мы заместили метод paint, что позволило апплету выполнить отрисовку. В классе Applet предусмотрены дополнительные

методы рисования, позволяющие эффективно закрашивать части экрана. При разработке первых апплетов порой непросто понять, почему метод update никогда не вызывается. Для инициации update предусмотрены три варианта метода repaint.

repaint

Метод repaint используется для принудительного перерисовывания апплета. Этот метод, в свою очередь, вызывает метод update. Однако, если ваша система медленная или сильно загружена, метод update может и не вызваться. Близкие по времени запросы на перерисовку могут объединяться AWT, так что метод update может вызываться спорадически. Если вы хотите добиться ритмичной смены кадров изображения, воспользуйтесь методом repaint(time) — это позволит уменьшить количество кадров, нарисованных не вовремя.

repaint(time)

Вы можете вызывать метод repaint, устанавливая крайний срок для перерисовки (этот период задается в миллисекундах относительно времени вызова repaint).

repaint(x, y, w, h)

Эта версия ограничивает обновление экрана заданным прямоугольником, изменены будут только те части экрана, которые в нем находятся.

repaint(time, x, y, w, h)

Этот метод — комбинация двух предыдущих.

Задание размеров графических изображений.

Графические изображения вычерчиваются в стандартной для компьютерной графики системе координат, в которой координаты могут принимать только целые значения, а оси направлены слева направо и сверху вниз. У апплетов и изображений есть метод size, который возвращает объект Dimension. Получив объект Dimension, вы можете получить и значения его переменных width и height:

Dimension d = size();
System.out.println(d. width + "," + d.height);

Графические средства

У объектов класса Graphics есть несколько простых функций рисования. Каждую из фигур можно нарисовать заполненной, либо прорисовать только ее границы. Каждый из методов drawRect, drawOval, fillRect и fillOval вызывается с четырьмя параметрами: int x, int y, int width и int height. Координаты x и y задают положение верхнего левого угла фигуры, параметры width и height определяют ее границы.

drawLine

drawline(int x1, int y1, int x2, int y2)

Этот метод вычерчивает отрезок прямой между точками с координатами (x1,y1) и (x2,y2). Эти линии представляют собой простые прямые толщиной в 1 пиксель. Поддержка разных перьев и разных толщин линий не предусмотрена.

drawArc и fillArc

Форма методов drawArc и fillArc следующая:

drawArc(int x, int y, int width, int height, int startAngle, int sweepAngle)

Эти методы вычерчивают (fillArc заполняет) дугу, ограниченную прямоугольником (x,y,width, height), начинающуюся с угла startAngle и имеющую угловой размер sweepAngle. Ноль градусов соответствует положению часовой стрелки на 3 часа, угол отсчитывается против часовой стрелки (например, 90 градусов соответствуют 12 часам, 180 — 9 часам, и так далее).

drawPolyson и fillPolyson

Прототипы для этих методов:

drawPolygon(int[], int[], int)

fillPolygon(int[], int[], int)

Метод drawPolygon рисует контур многоугольника (ломаную линию), задаваемого двумя массивами, содержащими x и y координаты вершин, третий параметр метода — число пар координат. Метод drawPolygon не замыкает автоматически вычерчиваемый

контур. Для того, чтобы прямоугольник получился замкнутым, координаты первой и последней точек должны совпадать.

Цвет

Цветовая система AWT разрабатывалась так, чтобы была возможность работы со всеми цветами. После того, как цвет задан, Java отыскивает в диапазоне цветов дисплея тот, который ему больше всего соответствует. Вы можете запрашивать цвета в той семантике, к которой привыкли — как смесь красного, зеленого и голубого, либо как комбинацию оттенка, насыщенности и яркости. Вы можете использовать статические переменные класса Color.black для задания какого-либо из общеупотребительных цветов - black, white, red, green, blue, cyan, yellow, magenta, orange, pink, gray, darkGray и lightGray.

Для создания нового цвета используется один из трех описанных ниже конструкторов.

Color(int, int, int)

Параметрами для этого конструктора являются три целых числа в диапазоне от 0 до 255 для красного, зеленого и голубого компонентов цвета.

Color(int)

У этого конструктора — один целочисленный аргумент, в котором в упакованном виде заданы красный, зеленый и голубой компоненты цвета. Красный занимает биты 16-23, зеленый — 8-15, голубой — 0-7.

Color(float, float, float)

Последний из конструкторов цвета, Color(float, float, float), принимает в качестве параметров три значения типа float (в диапазоне от 0.0 до 1.0) для красного, зеленого и голубого базовых цветов.

Методы класса Color

HSBtoRGB(float, float, float)
RGBtoHSB(int, int, int, float[1)

HSBtoRGB преобразует цвет, заданный оттенком, насыщенностью и яркостью (HSB), в целое число в формате RGB, готовое для использования в качестве параметра конструктора Color(int). RGBtoHSB преобразует цвет, заданный тремя базовыми компонентами, в массив типа float со значениями HSB, соответствующими данному цвету.

Цветовая модель HSB (Hue-Saturation-Brightness, оттенок-насыщенность-яркость) является альтернативой модели Red-Green-Blue для задания цветов. В этой модели оттенки можно представить как круг с различными цветами (оттенок может принимать значения от 0.0 до 1.0, цвета на этом круге идут в том же порядке, что и в радуге — красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый). Насыщенность (значение в диапазоне от 0.0 до 1.0) - это шкала глубины цвета, от легкой пастели до сочных цветов. Яркость - это также число в диапазоне от 0.0 до 1.0, причем меньшие значения соответствуют более темным цветам, а большие - более ярким.

getRedQ, getGreenO, setBlue()

Каждый из этих методов возвращает в младших восьми битах результата значение соответствующего базового компонента цвета.

getRGB()

Этот метод возвращает целое число, в котором упакованы значения базовых компонентов цвета, причем

red = Oxff & (getRGB() >> 16);
green = Oxff & (getRGB() >> 8);
blue = Oxff & getRGB();

setPaintMode() и setXORMode(Color)

Режим отрисовки paint — используемый по умолчанию метод заполнения графических изображений, при котором цвет пикселей изменяется на заданный. XOR устанавливает режим рисования, когда результирующий цвет получается выполнением операции XOR (исключающее или) для текущего и указанного цветов (особенно полезно для анимации).

Шрифты

Библиотека AWT обеспечивает большую гибкость при работе со шрифтами благодаря предоставлению соответствующих абстракций и возможности динамического выбора шрифтов. Вот очень короткая программа, которая печатает на консоли Java имена всех имеющихся в системе шрифтов.

```
/*
* <applet code="WhatFontsAreHere" width=100 height=40>
* </applet>
*
*/
import java.applet.*;
import java.awt.*;
public class WhatFontsAreHere extends Applet {
public void init() {
String FontList[];
FontList = getToolkit().getFontList();
for (int i=0; i < FontList.length; i++) {
System.out.println(i + ": " + FontList[i]);
}
} }
```

drawString

В предыдущих примерах использовался метод drawString(String, x, y). Этот метод выводит строку с использованием текущих шрифта и цвета. Точка с координатами (x,y) соответствует левой границе базовой линии символов, а не левому верхнему углу, как это принято в других методах рисования. Для того, чтобы понять, как при этом располагается описывающий строку прямоугольник, прочтите раздел о метрике шрифта в конце этой главы.

Использование шрифтов

Конструктор класса Font создает новый шрифт с указанным именем, стилем и размером в пунктах:

```
Font StrongFont = new Font("Helvetica", Font.BOLD|Font.ITALIC, 24);
```

В настоящее время доступны следующие имена шрифтов: Dialog, Helvetica, TimesRoman, Courier и Symbol. Для указания стиля шрифта внутри данного семейства предусмотрены три статические переменные. — Font.PLAIN, Font.BOLD и Font.ITALIC, что соответствует обычному стилю, курсиву и полужирному.

Теперь давайте посмотрим на несколько дополнительных методов.

getFamily и getName

Метод getFamily возвращает строку с именем семейства шрифтов. С помощью метода getName можно получить логическое имя шрифта.

getSize

Этот метод возвращает целое число, представляющее собой размер шрифта в пунктах.

getStyle

Этот метод возвращает целое число, соответствующее стилю шрифта. Полученный результат можно побитово сравнить со статическими переменными класса Font: — PLAIN, BOLD и ITALIC.

isBold, isItalic, isPlain

Эти методы возвращают true в том случае, если стиль шрифта — полужирный (bold), курсив (italic) или обычный (plain), соответственно.

Позиционирование и шрифты: FontMetrics

В Java используются различные шрифты, а класс FontMetrics позволяет программисту точно задавать положение выводимого в апплете текста. Прежде всего нам нужно понять кое-что из обычной терминологии, употребляемой при работе со шрифтами:

- *Высота* (height) — размер от верхней до нижней точки самого высокого символа в шрифте.
- *Базовая линия* (baseline) — линия, по которой выравниваются нижние границы символов (не считая снижения (descent)).

- *Подъем* (ascent) — расстояние от базовой линии до верхней точки символа.
- *Снижение* (descent) — расстояние от базовой линии до нижней точки символа.

Использование FontMetrics

Ниже приведены некоторые методы класса FontMetrics:

stringWidth

Этот метод возвращает длину заданной строки для данного шрифта.

bytesWidth, charsWidth

Эти методы возвращают ширину указанного массива байтов для текущего шрифта.

getAscent, getDescent, getHeight

Эти методы возвращают подъем, снижение и ширину шрифта. Сумма подъема и снижения дают полную высоту шрифта. Высота шрифта — это не просто расстояние от самой нижней точки букв g и y до самой верхней точки заглавной буквы T и символов вроде скобок. Высота включает подчеркивания и т.п.

getMaxAscent и getMaxDescent

Эти методы служат для получения максимальных подъема и снижения всех символов в шрифте.

1.1.11.2. Базовые типы

Простые типы в Java не являются объектно-ориентированными, они аналогичны простым типам большинства традиционных языков программирования. В Java имеется восемь простых типов: — byte, short, int, long, char, float, double и boolean. Их можно разделить на четыре группы:

1. Целые. К ним относятся типы byte, short, int и long. Эти типы предназначены для целых чисел со знаком.

2. Типы с плавающей точкой — float и double. Они служат для представления чисел, имеющих дробную часть.
3. Символьный тип char. Этот тип предназначен для представления элементов из таблицы символов, например, букв или цифр.
4. Логический тип boolean. Это специальный тип, используемый для представления логических величин.

В Java, в отличие от некоторых других языков, отсутствует автоматическое приведение типов. Несовпадение типов приводит не к предупреждению при трансляции, а к сообщению об ошибке. Для каждого типа строго определены наборы допустимых значений и разрешенных операций.

Целые числа

В языке Java понятие беззнаковых чисел отсутствует. Все числовые типы этого языка — знаковые. Например, если значение переменной типа byte равно в шестнадцатиричном виде 0x80, то это — число -1.

ЗАМЕЧАНИЕ

Единственная реальная причина использования беззнаковых чисел — это использование иных, по сравнению со знаковыми числами, правил манипуляций с битами при выполнении операций сдвига. Пусть, например, требуется сдвинуть вправо битовый массив mask, хранящийся в целой переменной и избежать при этом расширения знакового разряда, заполняющего старшие биты единицами. Стандартный способ выполнения этой задачи в C — ((unsigned) mask) >> 2. В Java для этой цели введен новый оператор беззнакового сдвига вправо. Приведенная выше операция записывается с его помощью в виде mask>>>2. Детально мы обсудим все операторы в следующей главе.

Отсутствие в Java беззнаковых чисел вдвое сокращает количество целых типов. В языке имеется 4 целых типа, занимающих 1, 2, 4 и 8 байтов в памяти. Для каждого типа — byte, short, int и long, есть свои естественные области применения.

byte

Тип byte — это знаковый 8-битовый тип. Его диапазон — от -128 до 127. Он лучше всего подходит для хранения произвольного потока байтов, загружаемого из сети или из файла.

byte b;

byte c = 0x55;

Если речь не идет о манипуляциях с битами, использования типа byte, как правило, следует избегать. Для нормальных целых чисел, используемых в качестве счетчиков и в арифметических выражениях, гораздо лучше подходит тип int.

short

short — это знаковый 16-битовый тип. Его диапазон — от -32768 до 32767. Это, вероятно, наиболее редко используемый в Java тип, поскольку он определен, как тип, в котором старший байт стоит первым.

short s;
short t = Ox55aa;

ЗАМЕЧАНИЕ

Случилось так, что на ЭВМ различных архитектур порядок байтов в слове различается, например, старший байт в двухбайтовом целом short может храниться первым, а может и последним. Первый случай имеет место в архитектурах SPARC и Power PC, второй — для микропроцессоров Intel x86. Переносимость программ Java требует, чтобы целые значения одинаково были представлены на ЭВМ разных архитектур.

int

Тип int служит для представления 32-битных целых чисел со знаком. Диапазон допустимых для этого типа значений — от -2147483648 до 2147483647. Чаще всего этот тип данных используется для хранения обычных целых чисел со значениями, достигающими двух миллиардов. Этот тип прекрасно подходит для использования при обработке массивов и для счетчиков. В ближайшие годы этот тип будет прекрасно соответствовать машинным словам не только 32-битовых процессоров, но и 64-битовых с поддержкой быстрой конвейеризации для выполнения 32-битного кода в режиме совместимости. Всякий раз, когда в одном выражении фигурируют переменные типов byte, short, int и целые литералы, тип всего выражения перед завершением вычислений приводится к int.

int i;

int j = 0x55aa0000;

long

Тип long предназначен для представления 64-битовых чисел со знаком. Его диапазон допустимых значений достаточно велик даже для таких задач, как подсчет числа атомов во вселенной.

long m;

long n = Ox55aa000055aa0000;

Не надо отождествлять *разрядность* целочисленного типа с занимаемым им количеством памяти. Исполняющий код Java может использовать для ваших переменных то количество памяти, которое сочтет нужным, лишь бы только их поведение соответствовало *поведению* типов, заданных вами. Фактически, нынешняя реализация Java из соображений эффективности хранит переменные типа byte и short в виде 32-битовых значений, поскольку этот размер соответствует машинному слову большинства современных компьютеров (СМ – 8 бит, 8086 – 16 бит, 80386/486 – 32 бит, Pentium – 64 бит).

Ниже приведена таблица разрядностей и допустимых диапазонов для различных типов целых чисел.

Имя	Разрядность	Диапазон
long	64	-9, 223, 372, 036, 854, 775, 808.. 9, 223, 372, 036, 854, 775, 807
Int	32	-2, 147, 483, 648.. 2, 147, 483, 647
Short	16	-32, 768.. 32, 767
byte	8	-128.. 127

Числа с плавающей точкой

Числа с плавающей точкой, часто называемые в других языках вещественными числами, используются при вычислениях, в которых требуется использование дробной части. В Java реализован стандартный (IEEE-754) набор типов для чисел с плавающей точкой — float и double и операторов для работы с ними. Характеристики этих типов приведены в таблице.

Имя	Разрядность	Диапазон
double	64	1. 7e-308.. 1. 7e+ 308
float	32	3. 4e-038.. 3. 4e+ 038

float

В переменных с обычной, или *одинарной точностью,* объявляемых с помощью ключевого слова float, для хранения вещественного значения используется 32 бита.

float f;

float f2 = 3. 14F; // обратите внимание на F, т.к. по умолчанию все литералы double

double

В случае *двойной точности,* задаваемой с помощью ключевого слова double, для хранения значений используется 64 бита. Все *трансцендентные* математические функции, такие, как sin, cos, sqrt, возвращают результат типа double.

double d;

double pi = 3. 14159265358979323846;

Символы

Поскольку в Java для представления символов в строках используется кодировка Unicode, разрядность типа char в этом языке — 16 бит. В нем можно хранить десятки тысяч символов интернационального набора символов Unicode. Диапазон типа char — 0..65536. Unicode — это объединение десятков кодировок символов, он включает в себя латинский, греческий, арабский алфавиты, кириллицу и многие другие наборы символов.

char c;

char c2 = Oxf132;

char c3 = ' a';

char c4 = '\n';

Хотя величины типа char и не используются, как целые числа, вы можете оперировать с ними так, как если бы они были целыми. Это дает вам возможность сложить два символа вместе, или инкрементировать значение символьной переменной. В приведенном ниже фрагменте кода мы, располагая базовым символом, прибавляем к нему целое число, чтобы получить символьное представление нужной нам цифры.

int three = 3;

char one = '1';

char four = (char) (three+ one);

В результате выполнения этого кода в переменную four заносится символьное представление нужной нам цифры — '4'. Обратите внимание — тип переменной one в приведенном выше выражении повышается до типа int, так что перед занесением результата в переменную four приходится использовать оператор явного приведения типа.

Тип boolean

В языке Java имеется простой тип boolean, используемый для хранения логических значений. Переменные этого типа могут принимать всего два значения — true (истина) ифalse (ложь). Значения типа boolean возвращаются в качестве результата всеми операторами сравнения, например (a < b) — об этом разговор пойдет в следующей главе. Кроме того, в главе 6 вы узнаете, что boolean — это тип, *требуемый* всеми условными операторами управления — такими, как if, while, do.

boolean done = false;

Массивы

Для объявления типа массива используются квадратные скобки. В приведенной ниже строке объявляется переменная month_days, тип которой — “массив целых чисел типа int”.

int month_days [];

Для того, чтобы зарезервировать память под массив, используется специальный оператор new. В приведенной ниже строке кода с помощью оператора new массиву month_days выделяется память для хранения двенадцати целых чисел.

```
month_days = new int [12];
```

Итак, теперь month_days — это ссылка на двенадцать целых чисел. Ниже приведен пример, в котором создается массив, элементы которого содержат число дней в месяцах года (невисокосного).

```
class Array {
public static void main (String args []) {
int month_days[];
month_days = new int[12];
month_days[0] = 31;
month_days[1] = 28;
month_days[2] = 31;
month_days[3] = 30;
month_days[4] = 31;
month_days[5] = 30;
month_days[6] = 31;
month_days[7] = 31;
month_days[8] = 30;
month_days[9] = 31;
month_days[10] = 30;
month_days[11] = 31;
System.out.println("April has " + month_days[3] + " days.");
} }
```

При запуске эта программа печатает количество дней в апреле, как это показано ниже. Нумерация элементов массива в Java начинается с нуля, так что число дней в апреле — это month_days [3].

```
C: \> java Array;
```

Многомерные массивы

На самом деле, настоящих многомерных массивов в Java не существует. Зато имеются массивы массивов, которые ведут себя подобно многомерным массивам, за исключением нескольких незначительных отличий. Приведенный ниже код создает традиционную

матрицу из шестнадцати элементов типа double, каждый из которых инициализируется нулем. Внутренняя реализация этой матрицы — массив массивов double.

```
double matrix [][] = new double [4][4];
```

Следующий фрагмент кода инициализирует такое же количество памяти, но память под вторую размерность отводится вручную. Это сделано для того, чтобы наглядно показать, что матрица на самом деле представляет собой вложенные массивы.

```
double matrix [][] = new double [4][];
matrix [0] = new double[4];
matrix[1] = new double[4];
matrix[2] = new double[4], matrix[3] = { 0, 1, 2, 3 };
```

В следующем примере создается матрица размером 4 на 4 с элементами типа double, причем ее диагональные элементы (те, для которых x==y) заполняются единицами, а все остальные элементы остаются равными нулю.

```
class Matrix {
public static void main(String args[]) { double m[][];
m = new double[4][4];
m[0][0] = 1;
m[1][1] = 1;
m[2][2] = 1;
m[3][3] = 1;
System.out.println(m[0][0] +" "+ m[0][1] +" "+ m[0][2] +""+ m[0][3]);
System.out.println(m[1][0] +" "+ m[1][1] +" "+ m[1][2] +""+ m[1][3]);
System.out.println(m[2][0] +" "+ m[2][1] +" "+ m[2][2] +""+ m[2][3]);
System.out.println(m[3][0] +" "+ m[3][1] +" "+ m[3][2] +""+ m[3][3]);
}
}
```

Запустив эту программу, вы получите следующий результат:

```
C : \> Java Matrix
1 0 0 0
0 1 0 0
0 0 1 0
```

0 0 0 1

Классы Java.

Базовым элементом объектно-ориентированного программирования в языке Java является класс. В этой главе Вы научитесь создавать и расширять свои собственные классы, работать с экземплярами этих классов и начнете использовать мощь объектно-ориентированного подхода. Напомним, что классы в Java не обязательно должны содержать метод main. Единственное назначение этого метода — указать интерпретатору Java, откуда надо начинать выполнение программы. Для того, чтобы создать класс, достаточно иметь исходный файл, в котором будет присутствовать ключевое слово class, и вслед за ним — допустимый идентификатор и пара фигурных скобок для его тела.

class Point {

}

К*ласс* — это шаблон для создания объекта. Класс определяет структуру объекта и его *методы,* образующие функциональный интерфейс. В процессе выполнения Java-программы система использует определения классов для создания представителей классов. Представители являются реальными *объектами.*Термины «представитель», «экземпляр» и «объект» взаимозаменяемы. Ниже приведена общая форма определения класса.

class имя_класса extends имя_суперкласса { type переменная1_объекта:

type переменная2_объекта:

type переменнаяN_объекта:

type имяметода1(список_параметров) { тело метода;

}

type имяметода2(список_параметров) { тело метода;

}

type имя методаМ(список_параметров) { тело метода;

}

}

1.1.11.5. Элементы управления

Управление в Java почти идентично средствам, используемым в C и C++.

Условные операторы

Они хорошо Вам знакомы, давайте познакомимся с каждым из них в Java.

if-else

В обобщенной форме этот оператор записывается следующим образом:

if (логическое выражение) оператор1; [else оператор2;]

Раздел else необязателен. На месте любого из *операторов* может стоять *составной оператор,* заключенный в фигурные скобки. *Логическое выражение* — это любое выражение, возвращающее значение типа boolean.

int bytesAvailable;

// ...

if (bytesAvailable > 0) {

ProcessData();

bytesAvailable -= n;

} else

waitForMoreData();

А вот полная программа, в которой для определения, к какому времени года относится тот или иной месяц, используются операторы if-else.

```
class IfElse {

public static void main(String args[]) { int month = 4;

String season;

if (month == 12 || month == 1 || month == 2) {

season = "Winter";

} else if (month ==3 || month == 4 || month == 5) {

season = "Spring";

} else if (month == 6 || month == 7 || month == 8) {

season = "Summer";

} else if (month == 9 || month == 10 || month == 11) {

season = "Autumn";

} else {

season = "Bogus Month";

}

System.out.println( "April is in the " + season  + ".");

} }
```

После выполнения программы вы должны получить следующий результат:

```
C: \> java IfElse
```

April is in the Spring.

break

В *языке* Java отсутствует оператор goto. Для того, чтобы в некоторых случаях заменять goto, в Java предусмотрен оператор break. Этот оператор сообщает исполняющей среде, что следует прекратить выполнение именованного блока и передать управление оператору, следующему за данным блоком. Для именования блоков в языке Java используются метки. Оператор break при работе с циклами и в операторах switch может использоваться без метки. В таком случае подразумевается выход из текущего блока.

Например, в следующей программе имеется три вложенных блока, и у каждого своя уникальная метка. Оператор break, стоящий во внутреннем блоке, вызывает переход на оператор, следующий за блоком b. При этом пропускаются два оператора println.

```
class Break {

public static void main(String args[]) { boolean t = true;

a:   { b:   { c:   {

System.out.println("Before the break"); // Перед break

  if (t)

  break b;

  System.out.println("This won't execute");  // Не будет  выполнено    }

  System.out.println("This won't execute"); // Не будет выполнено}

  System.out.println("This is after b"); //После b

} } }
```

В результате исполнения программы вы получите следующий результат:

```
C:\> Java Break
```

Before the break

This is after b

switch

Оператор switch обеспечивает ясный способ переключения между различными частями программного кода в зависимости от значения одной переменной или выражения. Общая форма этого оператора такова:

switch (выражение) { case значение1:

break;

case значение2:

break;

case значением:

break;

default:

}

return

В любом месте программного кода метода можно поставить оператор return, который приведет к немедленному завершению работы и передаче управления коду, вызвавшему этот метод. Ниже приведен пример, иллюстрирующий использование оператора return для немедленного возврата управления, в данном случае — исполняющей среде Java.

class ReturnDemo {

public static void main(String args[]) {

```
boolean t = true;

System.out.println("Before the return"); //Перед оператором return

if (t) return;

System.out.println("This won't execute"); //Это не будет выполнено

} }
```

Циклы

Любой цикл можно разделить на 4 части — *инициализацию, тело, итерацию и условие завершения.* В Java есть три циклические конструкции: while (с пред-условием), do-while (с пост-условием) и for (с параметром).

while

Этот цикл многократно выполняется до тех пор, пока значение логического выражения равно true. Ниже приведена общая форма оператора while:

[инициализация;]

while (завершение) {

тело;

[итерация;] }

do-while

Иногда возникает потребность выполнить тело цикла по крайней мере один раз — даже в том случае, когда логическое выражение с самого начала принимает значение false. Для таких случаев в Java используется циклическая конструкция do-while. Ее общая форма записи такова:

[инициализация;] do { тело; [итерация;] } while (завершение);

В следующем примере тело цикла выполняется до первой проверки условия завершения. Это позволяет совместить код итерации с условием завершения:

```
class DoWhile {

public static void main(String args[]) {

int n = 10;

do {

   System.out.println("tick " + n);

   } while (--n > 0);

} }
```

for

В этом операторе предусмотрены места для всех четырех частей цикла. Ниже приведена общая форма оператора записи for.

for (инициализация; завершение; итерация) тело;

Любой цикл, записанный с помощью оператора for, можно записать в виде цикла while, и наоборот. Если начальные условия таковы, что при входе в цикл условие завершения не выполнено, то операторы тела и итерации не выполняются ни одного раза. В каноническая форме цикла for происходит увеличение целого значения счетчика с минимального значения до определенного предела.

```
class ForDemo {

public static void main(String args[]) {

for (int i = 1; i <= 10; i++)

System.out.println("i = " + i);

} }
```

continue

В некоторых ситуациях возникает потребность досрочно перейти к выполнению следующей итерации, проигнорировав часть операторов тела цикла, еще не выполненных в текущей итерации. Для этой цели в Java предусмотрен оператор continue. Ниже приведен пример, в котором оператор continue используется для того, чтобы в каждой строке печатались два числа.

```
class ContinueDemo {

public static void main(String args[]) {

for (int i=0; i < 10; i++) {

   System.out.print(i + " ");

   if (i % 2 == 0) continue;

   System.out.println("");

   }

}}
```

Если индекс четный, цикл продолжается без вывода символа новой строки. Результат выполнения этой программы таков:

```
C: \> java ContinueDemo

0 1

2 3

4 5

5 7

8 9
```

1.1.11.6. Сети

Java поддерживает протокол TCP/IP, во-первых, расширяя свой интерфейс потоков ввода-вывода, описанного в предыдущей главе, и во вторых, добавляя возможности, необходимые для построения объектов ввода-вывода при работе в сети.

InetAddress

Java поддерживает адреса абонентов, принятые в Internet, с помощью класса InetAddress. Для адресации в Internet используются служебные функции, работающие с обычными, легко запоминающимися символическими именами, эти функции преобразуют символические имена в 32-битные адреса.

Фабричные методы

В классе InetAddress нет доступных пользователю конструкторов. Для создания объектов этого класса нужно воспользоваться одним из его фабричных методов. Фабричные методы — это обычные статические методы, которые возвращают ссылку на объект класса, которому они принадлежат. В данном случае, у класса InetAddress есть три метода, которые можно использовать для создания представителей. Это методы getLocalHost, getByName и getAllByName.

В приведенном ниже примере выводятся адреса и имена локальной машины, локального почтового узла и WWW-узла компании, в которой работает автор.

```
InetAddress Address = InetAddress.getLocalHost();

System.out.println(Address);

Address = InetAddress.getByName("mailhost");

System.out.println(Address);

InetAddress SW[] = InetAddress.getAllByNarne(“www.starwave.com");

System.out.println(SW);
```

У класса InetAddress также есть несколько нестатических методов, которые можно использовать с объектами, возвращаемыми только что названными фабричными методами:

- getHostName() возвращает строку, содержащую символическое имя узла, соответствующее хранящемуся в данном объекте адресу Internet.
- getAddress() возвращает байтовый массив из четырех элементов, в котором в порядке, используемом в сети, записан адрес Internet, хранящийся в данном объекте.
- toString() возвращает строку, в которой записано имя узла и его адрес.

РАЗДЕЛ II. ЧИСЛЕННЫЕ МЕТОДЫ

1.2.1. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Пусть задана непрерывная функция *f(x)* и требуется найти корни уравнения

$$f(x) = 0$$

на всей числовой оси или на некотором интервале $a < x < b$.

Всякое значение $x^* \in (a,b)$, удовлетворяющее условию $f\left(x^*\right) = 0$, называется *корнем уравнения* (1), а способ нахождения этого значения x^* и есть *решение уравнения*.

1.2.1.1. Метод бисекции

Математическая модель:

1. $f(a) \cdot f(b) < 0$;
2. $c = \frac{a+b}{2}$;
3. Если $f(a) \cdot f(c) < 0$, то $b = c$

   Если $f(b) \cdot f(c) < 0$, то $a = c$;
4. Критерий окончания счета:

$$\begin{cases} |f(c)| \le \varepsilon \\ |a-b| \le \varepsilon \end{cases}$$

1.2.1.2. Метод хорд

Математическая модель:

1. Если $f(a)\cdot f(b)<0$, то на отрезке $[a;b]$ существует корень;

2. 2.1. Если $f(a)<0$, то $x_0=a$ и $x_{n+1}=x_n-\frac{b-x_n}{f(b)-f(x_n)}\cdot f(x_n),\ n=0,1,...$

   2.2. Если $f(a)>0$, то $x_0=b$ и $x_{n+1}=x_n-\frac{a-x_n}{f(a)-f(x_n)}\cdot f(x_n),\ n=0,1,...$

3. Критерий окончания счета:

$$\begin{cases}\left|f(x_{n+1})\right|\le\varepsilon\\ \left|x_{n+1}-x_n\right|\le\varepsilon\end{cases}$$

1.2.1.3. Метод простой итерации

Математическая модель:

Дано нелинейное уравнение

$$f(x)=0 \qquad (1.2.1.3.1)$$

Корень отделен $x^*\in$ [a;b]. Требуется уточнить корень с точностью ε.

Уравнение (1) преобразуем к эквивалентному виду

$$x=\varphi(x) \qquad (1.2.1.3.2)$$

Выберем начальное приближение $x_0\in$ [a;b].

Вычислим новые приближения:

$$x_1=\varphi(x_0)$$

$$x_2=\varphi(x_1) \qquad (1.2.1.3.3)$$

$$x_i=\varphi(x_{i-1}) ,$$

i=1,2,… где i − номер итерации.

Последовательное вычисление значений x_i по формуле (1.2.3.3) называется итерационным процессом метода простых итераций, а сама формула - формулой итерационного процесса метода.

Если $\lim\limits_{i \to \infty} x_i = x^*$, то итерационный процесс сходящийся.

Условие сходимости

$$\left|\varphi'_{(x)}\right| < 1, \qquad \forall x \in [a,b] \tag{1.2.1.3.4}$$

Точное решение x^* получить невозможно, так как требуется бесконечный итерационный процесс.

Итерационный процесс продолжается при достижении условия

$$\left|x_i - x_{i-1}\right| \le \varepsilon , \tag{1.2.1.3.5}$$

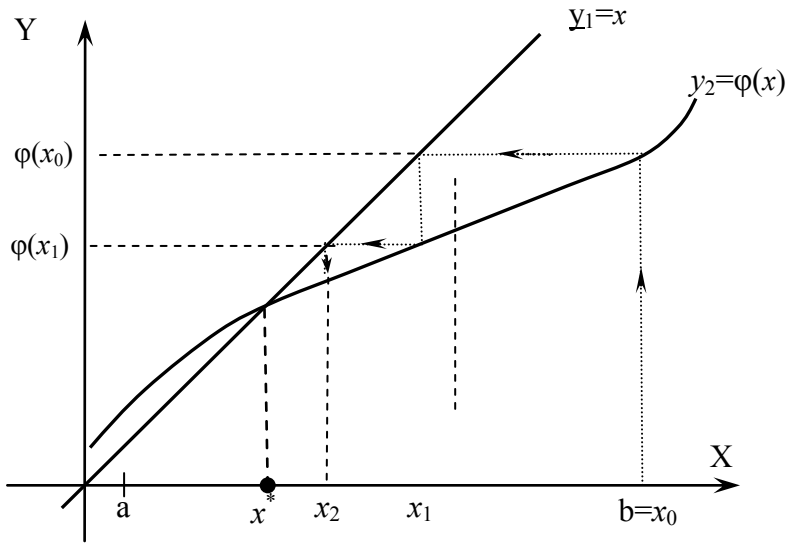
где ε - заданная точность; i - номер последней итерации.

Условие завершения итерационного процесса (1.2.1.3.5) обеспечивает близость значения x_i к точному решению:

$$\left|x^* - x_i\right| \le \varepsilon$$

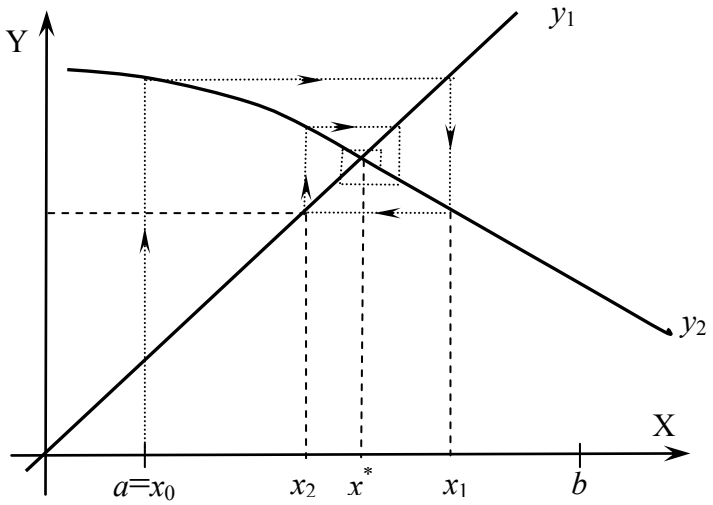
Рассмотрим геометрическую иллюстрацию метода простых итераций. Уравнение (1.2.3.2) представим на графике в виде двух функций*:* $y_1 = x$ и $y_2 = \varphi(x)$.

Возможные случаи взаимного расположения графиков функций, и соответственно, видов итерационного процесса показаны на рисунках 1.2.1.3.1 – 1.2.1.3.4.



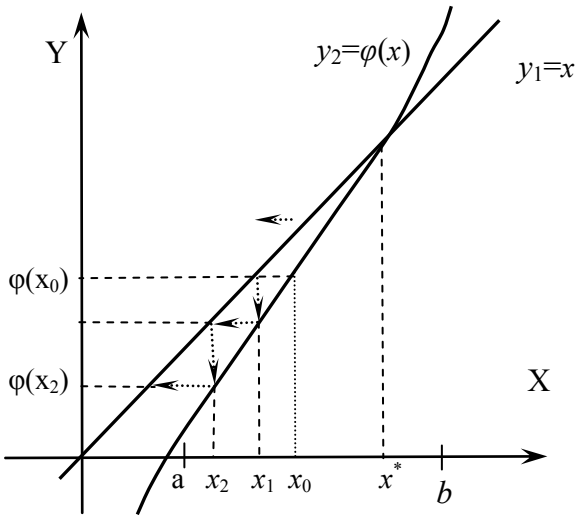
Итерационный процесс монотонно сходится из любой точки [a,b]

Рисунок 1.2.1.3.1. Итерационный процесс для случая $0<\varphi'_x<1 \quad \forall \quad x\in[a,b]$.



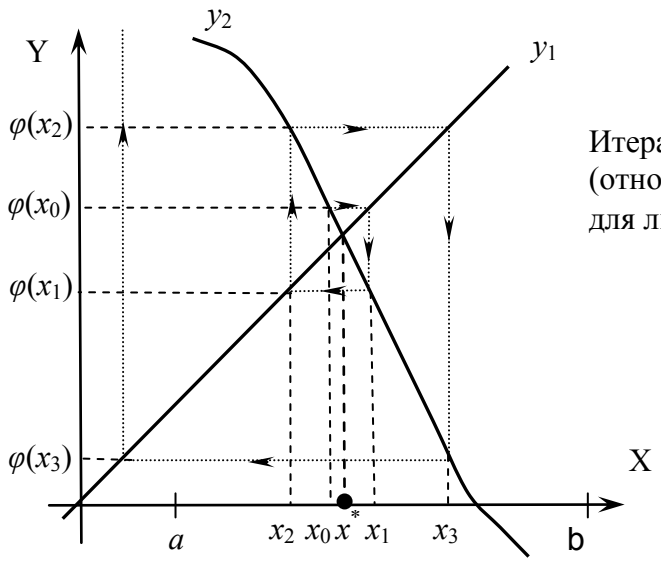
Итерационный процесс колебательно (около корня x^*) сходится из любой точки [a,b]

Рисунок 1.2.1.3.2. Итерационный процесс для случая $-1<\varphi'_x<1 \quad \forall \quad x\in[a,b]$.



Итерационный процесс монотонно расходится для любого $x_0 \in$[a,b]

Рисунок 1.2.1.3.3. Итерационный процесс для случая $\varphi'_x > 1 \quad \forall \quad x \in$[a,b].



Итерационный процесс колебательно (относительно корня x^*) расходится для любого $x_0 \in$[a,b]

Рисунок 1.2.1.3.4. Итерационный процесс для случая $\varphi'_x \le -1 \quad \forall \quad x \in$[a,b].

Из анализа графиков следует, что скорость сходимости растет при уменьшении значения $|\varphi'_x|$

Математическая модель:

1. Если $\varphi'(x) > 0$, то $x_0 = b$,

   если $\varphi'(x) < 0$, то $x_0 = a$;

2. Уравнение $f(x) = 0$ преобразуем к виду:

   $x_{n+1} = \varphi(x_n)$ или

   $x_{n+1} = x_n + c \cdot f(x_n)$, где $x_n + c \cdot f(x_n) = \varphi(x_n)$,

3. Условие сходимости метода простой итерации:

   $\left|\varphi'(x_0)\right| < 1$

4. Критерий окончания счета:

$$\begin{cases} \left|f(x_{n+1})\right| \le \varepsilon \\ \left|x_{n+1} - x_n\right| \le \varepsilon \end{cases}$$

1.2.1.4. Метод Ньютона (метод касательных)

Математическая модель:

Дано нелинейное уравнение (1.2.1.4.1)

$$f(x) = 0 \qquad (1.2.1.4.1)$$

Корень отделен $x^* \in$ [a;b].

Метод основан на стратегии постепенного уточнения корня. Формулу уточнения можно получить из геометрической иллюстрации идеи метода.

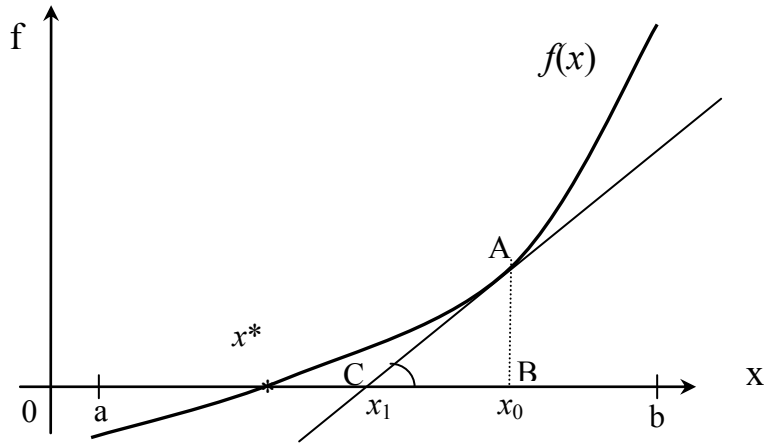


Рисунок 1.2.1.4.1. Геометрическая иллюстрация метода Ньютона.

На отрезке существования корня выбирается начальное приближение x_0. К кривой $f(x)$ в точке А с координатами $(x_0, f(x_0))$ проводится касательная. Абсцисса x_1 точки пересечения этой касательной с осью ОХ является новым приближением корня.

Из рисунка следует, что $x_1 = x_0 - CB$

Из ΔABC: $CD = \dfrac{AB}{tg\angle ACB}$. Но $tg\angle ACB = f'(x_0), \quad AB = f(x_0)$.

Следовательно, $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

Аналогично, для i-го приближения можно записать формулу итерационного процесса метода Ньютона:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}, i = 1,2,\ldots, \quad \text{где} \quad x_0 \in [a;b]. \qquad (1.2.1.4.2)$$

Условие окончания расчета: $|\delta| \le \varepsilon$, (1.2.1.4.3)

где $\delta = x_{i-1} - x_i = \dfrac{f(x_{i-1})}{f'(x_{i-1})}$ –корректирующее приращение или поправка.

Условие сходимости итерационного процесса:

$$\left|f(x)\cdot f''(x)\right| < \left(f'(x)\right)^2 \ \forall x \in [a,b] \qquad (1.2.1.4.4)$$

Если на отрезке существования корня знаки $f'_{(x)}$ и $f''_{(x)}$ не изменяются, то начальное приближение, обеспечивающее сходимость, нужно выбрать из условия

$$f(x_0)\cdot f''(x_0) > 0, \ x_0 \in [a;b]. \qquad (1.2.1.4.5)$$

т.е. в точке начального приближения знаки функций и ее второй производной должны совпадать.

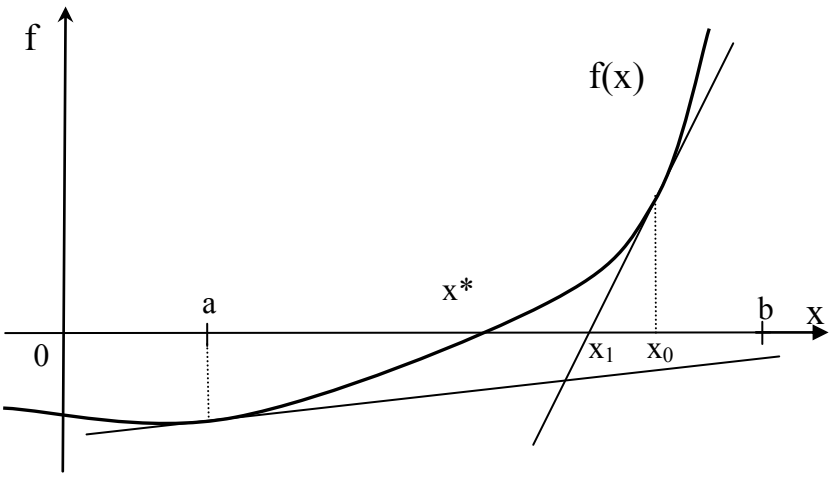


Рисунок 1.2.1.4.2. Геометрическая иллюстрация выбора начального приближения: график f(x) вогнутый, $f''(x) > 0$, тогда $x_0=b$, т.к. $f(b)>0$.

Если же выбрать $x_0=a$, то итерационный процесс будет сходиться медленнее или даже расходиться (см. касательную для $x_0=a$).

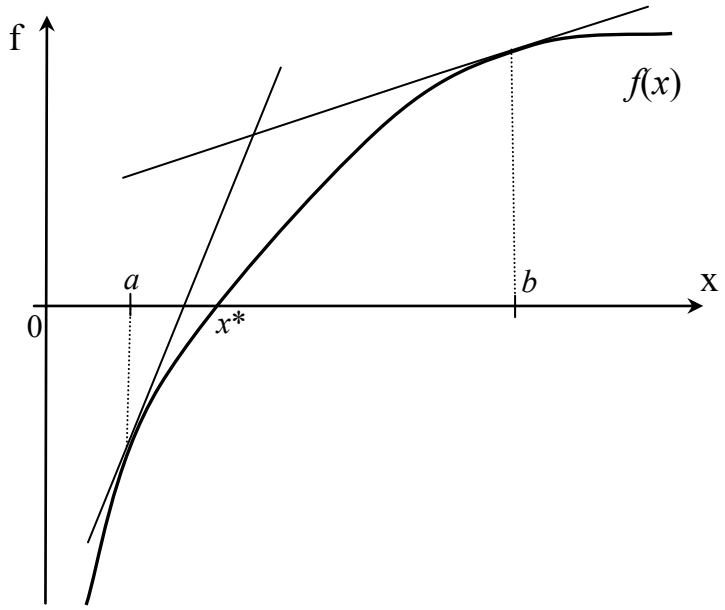


Рисунок 1.2.1.4.3. Геометрическая иллюстрация выбора начального приближения: график *f(x)* выпуклый, *f ''(x)*<0 , тогда $x_0 = a$, т.к. *f(a)*<0.

Метод Ньютона в отличие от ранее рассмотренных методов используют свойства функции в виде значения производной, что значительно ускоряет итерационный процесс. При этом, чем больше значение модуля производной в окрестности корня (чем круче график функции), тем быстрее сходимость.

Математическая модель

1. За начальное приближение принимается такое значение $x_0 \in [a;b]$,

   для которого выполняется условие Фурье: $f(x_0) \cdot f''(x_0) > 0$;

2. $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$,

3. Критерий окончания счета:

$$\begin{cases} \left| f(x_{n+1}) \right| \le \varepsilon \\ \left| x_{n+1} - x_n \right| \le \varepsilon \end{cases}$$

1.2.2. ИНТЕРПОЛЯЦИЯ

Рассматриваемые методы:

1. линейной интерполяции
2. Лагранжа

1.2.2.1.Системы функций Чебышева

Рассмотрим линейное множество R действительных функций, определенных на отрезке $[a,b]$, и некоторую конечную или счетную систему линейно независимых функций $\{\varphi_i(x)\}$ из этого множества.

Линейную комбинацию

$$\tau(x)=\sum_{i=0}^{n} c_i\varphi_i(x)$$

с действительными коэффициентами c_i называют обобщенным многочленом по системе функций $\varphi_k(x)\ \ (k=0,\ 1,\ ...,\ n)$.

Определение. *Совокупность функций* $\varphi_0(x),\ \varphi_1(x),\ ...,\ \varphi_n(x)$ *называется системой Чебышева на отрезке* $[a,b]$*, если любой обобщенный многочлен по этой системе, у которого хотя бы один из коэффициентов отличен от нуля, имеет на* $[a,b]$ *не более* n *корней.*

Пусть отрезке $[a,b]$ в некоторых попарно различных точках $x_0,\ x_1,\ ...,\ x_n$ известны значения функций $f(x)$.

Задача интерполирования функции $f(x)$ состоит в том, чтобы найти значение $f(x),\ x\neq x_i\ \ (i=0,\ 1,\ ...,\ n)$, если известны узлы интерполирования $x_0,\ x_1,\ ...,\ x_n$ и значения функции $f(x)$ в этих узлах. Решается задача интерполирования следующим образом: выбирается система функций $\varphi_i(x)$, строится обобщенный многочлен $\tau(x)=\sum_{i=0}^{n} c_i\varphi_i(x)$, а коэффициенты c_i задаются таким образом, чтобы в узлах

интерполирования значения обобщенного многочлена совпадали со значениями данной функции $f(x)$:

$$\tau(x_k) = \sum_{i=0}^{n} c_i \varphi_i(x_k) = f(x_k), \quad k = 0, 1, \ldots, n.$$

Обобщенный многочлен, обладающий таким свойством, называется обобщенным интерполяционным многочленом. За приближенное значение $f(x)$ принимают значение $\tau(x)$.

Выясним, когда задача интерполирования решается однозначно.

Т е о р е м а 1. *Для того, чтобы для любой функции* $f(x) \in R$, *определенной на отрезке* $[a,b]$, *и любого набора* $n+1$ *узлов* x_0, x_1, ..., x_n ($x_i \neq x^j$ *при* $i \neq j$, $x_i \in [a,b]$) *существовал и был бы единственным обобщенный интерполяционный многочлен* $\tau(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + \ldots + c_n\varphi_n(x)$, *необходимо и достаточно, чтобы система функций* $\varphi_i(x)$ ($i = 0,\ 1,\ \ldots,\ n$) *являлась системой Чебышева на* $[a,b]$.

На практике чаще всего используются следующие системы:

1) $1,\ x,\ x^2,\ \ldots,\ x^n,\ \ldots$;

2) $1,\ \sin x,\ \cos x,\ \ldots,\ \sin nx,\ \cos nx,\ \ldots$;

3) $e^{\alpha_0 x},\ e^{\alpha_1 x},\ \ldots,\ e^{\alpha_n x},\ \ldots$, где α_i – некоторая числовая последовательность попарно различных действительных чисел.

В случае 1) интерполирование называется алгебраическим, в случае 2) – тригонометрическим (применяется для приближения 2π – периодических функций), в случае 3) – экспоненциальным.

1.2.2.2. Формула Лагранжа

Пусть $a = x_0 < x_1 < \ldots < x_n = b$ — набор различных точек (узлов) на отрезке $[a,\ b]$, в котором заданы значения достаточно гладкой функции $f(x)$ так, что $f_i = f(x_i)$, $i = 0,\ 1,\ \ldots,\ n$. Требуется построить многочлен $L_n(x)$ степени не выше n принимающий

в точках x_i значения f_i, и оценить погрешность приближения функции этим многочленом на всём отрезке $[a,\ b]$.

Введём в явном виде вспомогательные многочлены $\Omega_i(x)$ степени $n-1$,

удовлетворяющие условиям $\Omega_i(x_j)=\begin{cases}1, & \text{если } i=j,\\ 0, & \text{если } i\neq j.\end{cases}$ по формулам:

$$\Omega_i(x)=\prod_{\substack{j=0\\ i\neq j}}^{n}\frac{x-x_j}{x_i-x_j}. \qquad (1.2.2.2.1)$$

Тогда интерполяционный многочлен $L_n(x)$ можно задать по формуле Лагранжа

$$L_n(x)=\sum_{i=0}^{n}\Omega_i(x)f_i\ , \qquad (1.2.2.2.2)$$

при этом $\Omega_i(x)$ удобно преобразовать к виду

$$\Omega_i(x)=\frac{\omega(x)}{(x-x_i)\omega'(x_i)}, \qquad (1.2.2.2.3)$$

где $\omega(x)=\prod_{i=0}^{n}(x-x_i)$.

Разность $r_n(x)=f(x)-L_n(x)$ называется погрешностью интерполирования или остаточным членом интерполирования. В узлах интерполирования погрешность $r_n(x)$ обращается в нуль, в остальных точках она отлична от нуля, но если $f(x)$ – многочлен степени k, $k<n+1$, то $r_n(x)\equiv 0$. Если функция $f(x)$ имеет непрерывную $(n+1)$-ю производную, то остаточный член можно представить в виде

$$r_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x), \qquad (1.2.2.2.4)$$

где ξ – некоторая точка, лежащая на отрезке, содержащем узлы $x_0,\ x_1,\ \ldots\ ,\ x_n$ и точку x.

Пример:

Построить многочлен наименьшей степени, принимающий в данных точках следующие значения:

x	-1	0	3	6
y	-14	-5	22	175

По формуле (1.2.2.2.3) вычислим вспомогательные многочлены

$$\Omega_0(x)=\frac{(x-0)(x-3)(x-6)}{(-1-0)(-1-3)(-1-6)}=\frac{x^3-8x^2+9x+18}{18},$$

$$\Omega_1(x)=\frac{(x-(-1))(x-3)(x-6)}{(0-(-1))(0-3)(0-6)}=\frac{x^3-8x^2+9x+18}{18},$$

$$\Omega_2(x)=\frac{(x-(-1))(x-0)(x-6)}{(3-(-1))(3-0)(3-6)}=\frac{x^3-5x^2-6x}{-36},$$

$$\Omega_3(x)=\frac{(x-(-1))(x-0)(x-3)}{(6-(-1))(6-0)(6-3)}=\frac{x^3-2x^2-3x}{126}.$$

Затем по формуле (2) построим интерполяционный многочлен Лагранжа

$$L_3(x)=-14\frac{x^3-9x^2+18}{-28}+(-5)\frac{x^3-8x^2+9x+18}{18}+22\frac{x^3-5x^2-6x}{-36}+$$

$$+175\frac{x^3-2x^2-3x}{126}=x^3-2x^2+6x-5.$$

1.2.2.3. Линейная интерполяция

Линейная интерполяция состоит в том, что заданные точки $M(x_i,y_i)$ $(i=0,1,...,n)$ соединяются прямолинейными отрезками и функция $f(x)$ приближается к ломаной с вершинами в данных точках. Поскольку имеется n интервалов $\left(x_{i-1},x_i\right)$, то для каждого из них в качестве уравнения интерполяционного полинома используется уравнение прямой, проходящей через две точки. В частности для *i*-го интервала можно записать уравнение прямой, проходящей через точки $\left(x_{i-1},y_{i-1}\right)$ и $\left(x_i,y_i\right)$, в виде:

$$\frac{y-y_{i-1}}{y_i-y_{i-1}}=\frac{x-x_{i-1}}{x_i-x_{i-1}}$$

Отсюда $y=a_i x+b_i,\quad x_{i-1}\le x\le x_i,$

$$a_i=\frac{y_i-y_{i-1}}{x_i-x_{i-1}},\ b_i=y_{i-1}-a_i x_{i-1}.$$

Следовательно, при использовании линейной интерполяции сначала нужно определить интервал, в который попадает значение аргумента *x*, а затем подставить его в формулу и найти приближенное значение функции в этой точке.

1.2.3. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ГАУССА

Система *m* линейных уравнений с *m* неизвестными имеет вид:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mm}x_m = b_m, \end{cases}$$

где a_{ij}, b_i , (*i*= 1,2,...,*m*; *j*=1,2,...,*m*)– произвольные числа, называемые соответственно коэффициентами при переменных и свободными членами уравнений.

Система уравнений называется *совместной*, если она имеет хотя бы одно решение, и *несовместной*, если она не имеет решений.

Точные методы

Метод называется ***точным***, если при точном выполнении всех требуемых действий мы получаем точное решение системы. При этом имеется в виду, что все коэффициенты – точные числа.

Методы исключения неизвестных (Метод Гаусса)

Наиболее распространенным методом решения СЛАУ является метод Гаусса, в основе которого лежит идея последовательного исключения неизвестных. С помощью элементарных преобразований система уравнений приводится к равносильной системе ступенчатого (или треугольного) вида, из которой последовательно, начиная с последних (по номеру) переменных, находятся все остальные переменные.

Рассмотрим решение системы (1) *m* линейных уравнений с *m* неизвестными в общем виде.

Предположим, что в системе (1) $a_{11} \neq 0$(этого всегда можно добиться перестановкой уравнений или переменных с изменением их номеров).

Метод Гаусса состоит в последовательном исключении неизвестных из этой системы.

Предположим, что $a_{11} \neq 0$. Последовательно умножая первое уравнение на $-\frac{a_{i1}}{a_{11}}$ и складывая с *i*-м уравнением, исключим x_1 из всех уравнений кроме первого. Получим систему

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1, \\ \qquad a_{22}^{(1)}x_2 + \ldots + a_{2m}^{(1)}x_m = b_2^{(1)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad a_{m2}^{(1)}x_2 + \ldots + a_{mm}^{(1)}x_m = b_m^{(1)}, \end{cases}$$

где $a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}$, $b_i^{(1)} = b_i - \frac{a_{i1}b_1}{a_{11}}$, $i, j = 2,3,\ldots,m$.

Аналогичным образом из полученной системы исключим x_2. Последовательно, исключая все неизвестные, получим систему треугольного вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \quad \ldots \quad + a_{1k}x_k + a_{1m}x_m = b_1, \\ \qquad a_{22}^{(1)}x_2 + \quad \ldots \quad + a_{2k}^{(1)}x_k + a_{2m}^{(1)}x_m = b_2^{(1)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad a_{m-1,m-1}^{(m-1)}x_{m-1} + a_{m-1,m}^{(m-1)}x_m = b_m^{(m-1)}, \\ \qquad\qquad\qquad\qquad a_{m,m}^{(m-1)}x_m = b_m^{(m-1)}. \end{cases}$$

Описанная процедура называется прямым ходом метода Гаусса. Ее выполнение возможно при условии, что все a_{ii}^{l}, $l = 1,2,\ldots,m$ не равны нулю.

Выполняя последовательные подстановки в последней системе, (начиная с последнего уравнения) можно получить все значения неизвестных.

$$x_m = \frac{b_m^{(m-1)}}{a_{m,m}^{(m-1)}}, \; x_i = \frac{1}{a_{i,i}^{(i-1)}}\left(b_{i,i}^{(i-1)} - \sum_{j=i-1}^{m} a_{ij}^{(i-1)}x_j\right).$$

Эта процедура получила название обратный ход метода Гаусса.

1.2.4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДАМИ ПРОСТОЙ ИТЕРАЦИИ И МЕТОДОМ ЗЕЙДЕЛЯ

Метод называется ***приближенным,*** если при точном выполнении всех требуемых действий и при точных коэффициентах мы получаем, как правило, лишь ***приближенный результат***.

Впрочем, практически и при применении точного метода результат оказывается приближенным по двум причинам: во-первых, коэффициенты уравнений обычно бывают приближенными числами; во-вторых, промежуточные вычисления на практике обычно невозможно выполнять с полной точностью. Значит, погрешность окончательного результата складывается из неустранимой погрешности задания исходных данных (коэффициентов) и погрешностей округления.

В случае применения приближенного метода на окончательный результат влияет всегда погрешность метода, также на практике обычно имеет место неустранимая погрешность в задании коэффициентов и погрешность округления в промежуточных действиях.

Одношаговый линейный стационарный итерационный процесс называется методом простой итерации (МПИ).

Пусть дана система ЛАУ $A \cdot x = f$ с неособенной матрицей. В методе простой итерации ее предварительно приводят к виду

$$x = B \cdot x + C$$

Для очень больших систем, когда метод Гаусса становится неэффективным, применяют *итерационные методы*, например, метод простой итерации или метод Зейделя.

Метод простой итерации

Вычисления по *методу простой итерации* начинаются с произвольного вектора $X^0 = \{x_1^0, x_2^0, ..., x_n^0\}$. Итерационный процесс осуществляется по формуле:

$$x_i^{j+1} = x_i^j + \frac{1}{a_{ii}} \cdot \left(b_i - \sum_{k=1}^{n} a_{ik} \cdot x_k^j \right); i = 1,2,\ldots n; j = 0,1,\ldots,$$

т.е. *все* неизвестные на следующей итерации вычисляются *только* через неизвестные на предыдущей итерации.

Условия завершения итерационного процесса:

$$\delta \le \varepsilon \qquad (1.2.4.1)$$

где ε – требуемая точность;

δ – оценка достигнутой точности,

$$\delta = \max_{i=1,n} \left| x_i^{(k)} - x_i^{(k-1)} \right| \qquad (1.2.4.2)$$

или

$$\delta = \frac{1}{n} \sum_{i=1}^{n} \left| x_i^{(k)} - x_i^{(k-1)} \right| \qquad (1.2.4.3)$$

Условие сходимости итерационного процесса (условие преобладания диагональных коэффициентов):

$$|a_{ii}| > \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| \quad \forall \quad i = \overline{1,n} \qquad (1.2.2.4)$$

Метод Зейделя

Метод Зейделя отличается от простой итерации тем, что найдя какое-то приближение для компоненты, мы сразу же используем его для отыскания следующей компоненты.

В *методе Зейделя* используется итерационная формула

$$x_i^{j+1} = x_i^j + \frac{1}{a_{ii}} \cdot \left(b_i - \sum_{k=1}^{i-1} a_{ik} \cdot x_k^{j+1} - \sum_{k=i}^{n} a_{ik} \cdot x_k^j \right); i = 1,2,\ldots n; j = 0,1,\ldots,$$

в которой при вычислении очередного неизвестного используются *последние* найденные значения остальных неизвестных. Вычисления заканчиваются, когда *невязки* системы

становятся достаточно малыми. Итерационные методы сходятся не для всякой матрицы. Достаточным условием сходимости является *положительная определенность* матриц.

1.2.5. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ НЬЮТОНА

Метод Ньютона применяется к решению систем уравнений вида

$$\begin{cases} f(x,y)=0, \\ g(x,y)=0. \end{cases},$$

$J(x,y)=\begin{bmatrix} f'_x(x,y) & f'_y(x,y) \\ g'_x(x,y) & g'_y(x,y) \end{bmatrix}$ - матрица Якоби.

Тогда последовательные приближения по методу Ньютона вычисляются по формуле:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - J^{-1}(x_n,y_n) \begin{pmatrix} f(x_n,y_n) \\ g(x_n,y_n) \end{pmatrix},$$

$$\text{где } J^{-1}(x_n,y_n) = \frac{1}{\det\left(J\left(x_n,y_n\right)\right)} \begin{vmatrix} g'_y(x_n,y_n) & -f'_y(x_n,y_n) \\ -g'_x(x_n,y_n) & f'_x(x_n,y_n) \end{vmatrix}$$

или

$$\begin{cases} x_{n+1} = x_n - \dfrac{1}{\det\left(J\left(x_n,y_n\right)\right)} (g'_y(x_n,y_n)\cdot f(x_n,y_n) - f'_y(x_n,y_n)\cdot g(x_n,y_n)), \\ y_{n+1} = y_n - \dfrac{1}{\det\left(J\left(x_n,y_n\right)\right)} (-g'_x(x_n,y_n)\cdot f(x_n,y_n) + f'_x(x_n,y_n)\cdot g(x_n,y_n)). \end{cases}$$

1.2.6. МЕТОД ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

Рассматриваемые методы:

1. трапеций
2. прямоугольников
3. Симпсона

Необходимость вычисления значений определенных интегралов при моделировании возникает достаточно часто.

Формула Ньютона-Лейбница

$$I^* = \int_a^b f(x)dx = F(b) - F(a) \qquad (1.2.6.1)$$

имеет ограниченное применение:

- во-первых, не позволяет вычислить интегралы от таблично заданной подынтегральной функции *f(x)*;
- во-вторых, не всякая подынтегральная функция имеет первообразную *F(x)*.

Численные методы интегрирования универсальны: позволяют вычислить значение определенного интеграла непосредственно по значениям подынтегральной функции *f(*x*)*, независимо от способа ее задания или вида аналитического выражения.

Геометрический смысл определенного интеграла – площадь криволинейной трапеции, ограниченной осью OX, кривой *f(x)*, и прямыми *x=a* и *x=b* (Рис.6.1.).

Численные методы интегрирования основаны на различных способах оценки этой площади, поэтому полученные формулы численного интегрирования называются *квадратурными* (формулами вычисления площади).

Рассмотрим получение и применение простейших формул.

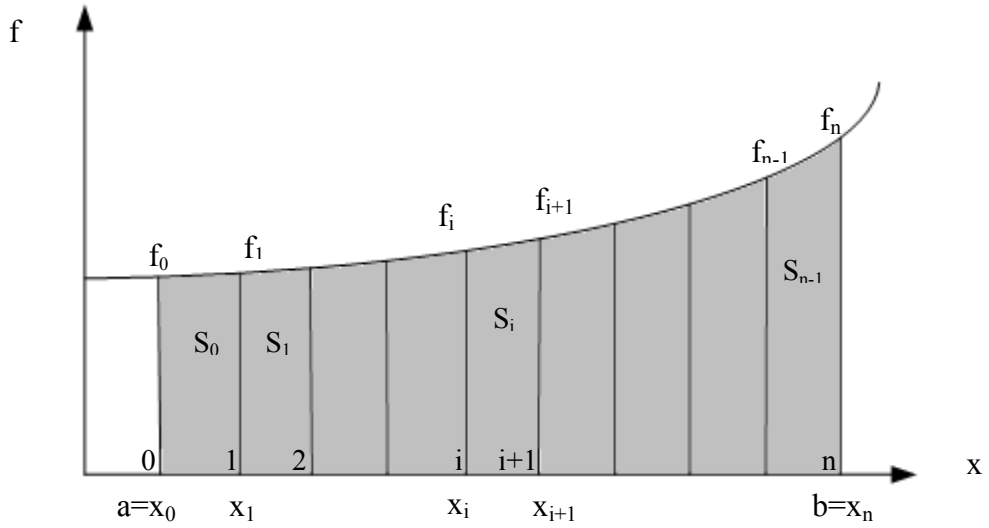


Рисунок 1.2.6.1. Геометрический смысл определённого интеграла

Отрезок [*a,b*] делят на *n* равных частей – *элементарных отрезков*. Принято такое деление отрезка называть сеткой, а точки $x_0, x_1, \ldots, x_n$ – узлами сетки.

Если сетка равномерная, то $h = \frac{b-a}{n}$ – шаг сетки, при интегрировании – шаг интегрирования, а координата i-го узла вычисляется по формуле:

$$x_i = a + i \cdot h, \qquad i = \overline{0,n} \tag{1.2.6.2}$$

Полная площадь криволинейной трапеции состоит из n элементарных криволинейных трапеций – элементарных площадей:

$$I^* = \sum_{i=0}^{n-1} S_i \tag{1.2.6.3}$$

Квадратурные формулы отличаются друг от друга способом оценки значения S_i – площади элементарной криволинейной трапеции.

Рассмотрим получение простейших формул для часто используемой равномерной сетки.

1.2.6.1. Формула трапеций

В данном методе элементарная криволинейная трапеция заменяется трапецией (кривая f(x) заменяется хордой CD).

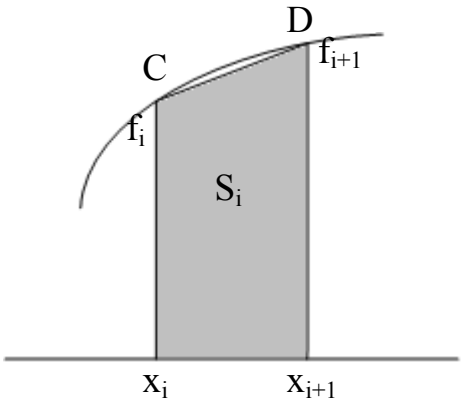


Рисунок 1.2.2.6.1.1. Оценка элементарной площади S_i трапецией.

Из рисунка видно, что $S_i \approx \frac{f_i + f_{i+1}}{2} \cdot h.$

Отсюда:

$$I^* = \sum_{i=0}^{n-1} S_i \approx \sum_{i=0}^{n-1} \frac{f_i + f_{i+1}}{2} \cdot h = h \cdot \left(\frac{f_0 + f_1}{2} + \frac{f_1 + f_2}{2} + \frac{f_2 + f_3}{2} + \ldots + \frac{f_{n-2} + f_{n-1}}{2} + \frac{f_{n-1} + f_n}{2}\right) =$$

$$= h \cdot \left(\frac{f_0 + f_n}{2} + \sum_{i=1}^{n-1} f_i\right) = h \cdot \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(a + i \cdot h)\right) \qquad (1.2.6.1.1)$$

Погрешность формулы трапеций пропорциональная квадрату шаг h $\delta_M \approx O(h^2)$ т.е. *формулы центральных прямоугольников и трапеций имеют близкую точность.*

Формула трапеций имеет такую же точность, как и формула центральных прямоугольников.

Знак погрешности легко объяснить по геометрической иллюстрации применения формулы.

1.2.6.2. Формулы прямоугольников

Площадь *i*-той элементарной трапеции можно оценить (приближенно вычислить) как площадь прямоугольника со сторонами $x_{i+1} - x_i = h$ и f_i. Тогда $S_i \approx f_i \cdot h$ и значение интеграла:

$$I^* = \sum_{i=0}^{n-1} S_i \approx \sum_{i=0}^{n-1} f_i \cdot h = h \sum_{i=0}^{n-1} f_i = h \sum_{i=0}^{n-1} f(x_i) = h \sum_{i=0}^{n-1} f(a + i \cdot h) \qquad (1.2.6.2.1)$$

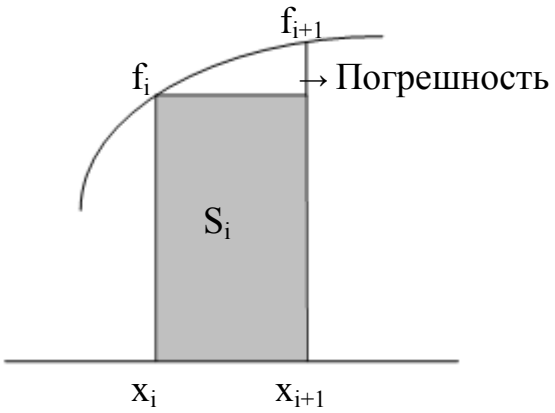


Рисунок 1.2.6.2.1. Оценка элементарной площади S_i левым прямоугольником.

Полученная формула называется формулой *левых прямоугольников*, т.к. для оценки площади использовалось левое основание элементарной криволинейной трапеции.

Аналогично можно получить формулу *правых прямоугольников*:

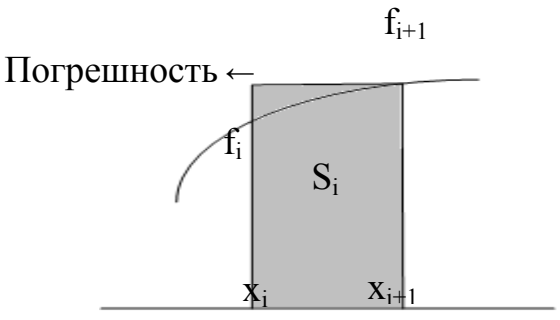


Рисунок 1.2.6.2.2. Оценка элементарной площади S_i правым прямоугольником.

Для данного случая $S_i \approx f_{i+1} \cdot h$ и тогда значение интеграла:

$$I^* = \sum_{i=0}^{n-1} S_i \approx \sum_{i=0}^{n-1} f_{i+1} \cdot h = h \sum_{i=1}^{n} f_i = h \sum_{i=1}^{n} f(x_i) = h \sum_{i=1}^{n} f(a + i \cdot h) \qquad (1.2.6.2.2)$$

Эти формулы не находят широкого применения, т.к. имеют большую погрешность, пропорциональную величине шага $\delta_M \approx O(h)$

Как появляется эта погрешность, видно на рисунках.

Для повышения точности площадь S_i можно оценить, используя прямоугольник со стороной, равной значению подынтегральной функции в середине элементарного отрезка

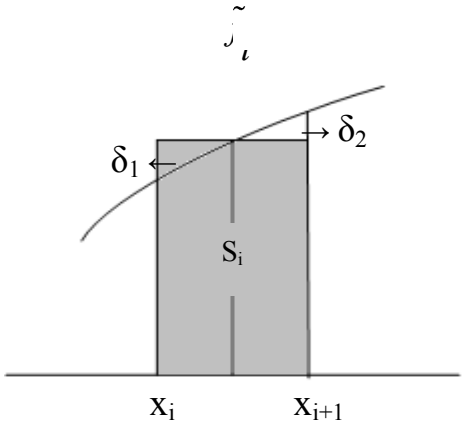


Рисунок 1.2.6.2.3. Оценка элементарной площади S_i центральным прямоугольником.

Для данного случая $S_i \approx$ и формула центральных прямоугольников имеет вид:

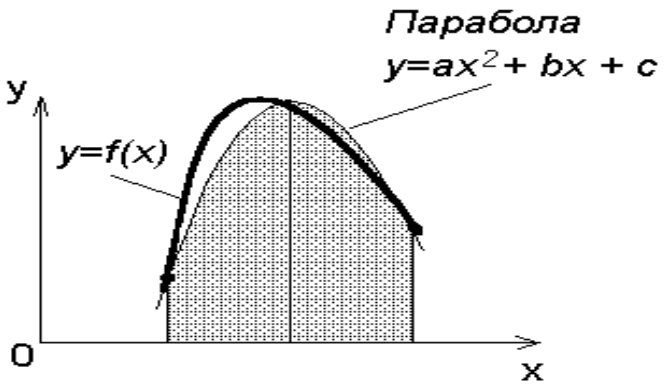
$$I^* = \sum_{i=0}^{n-1} S_i \approx \sum_{i=0}^{n-1} \qquad i=0 \qquad i=0 \qquad i+\frac{h}{2}) = h \sum_{i=0}^{n-1} f(a + i \cdot h + \frac{h}{2}) \qquad (1.2.6.2.3)$$

Как видно из рисунка 1.16.2.3 погрешность в оценке площади S_i в данном случае существенно меньше, чем в двух предыдущих (погрешность оценивается разницей площадей δ_1 и δ_2).

Погрешность метода пропорциональная квадрату величины шага $\delta_M \approx O(h^2)$

Формула центральных прямоугольников на порядок точнее предыдущих формул.

1.2.6.3. Формула Симпсона



На каждом элементарном отрезке подынтегральная функция *f(x)* заменяется квадратичной параболой, построенной по трем точкам: концам элементарного отрезка (x_i, f_i), (x_{i+1}, f_{i+1}) и его середине ($x_i + \frac{h}{2}, \tilde{f}_i$).

Площадь полученной криволинейной трапеции служит оценкой элементарной площади S_i:

$$S_i \approx \frac{h}{6} \cdot \left(f_i + 4\tilde{f}_i \quad f_{i+1} \right) = \frac{h}{6} \cdot \left(f(x_i) + 4f(x_i + \frac{h}{2}) + f(x_{i+1}) \right)$$

Тогда значение интеграла:

$$I^* \approx \sum_{i=0}^{n-1} \frac{h}{6} \left(f_i + 4\tilde{f}_i \quad f_{i+1} \right) =$$

$$\frac{h}{6} \left(f_0 + 4\tilde{f}_0 \quad f_1 \quad f_1 \quad \tilde{f}_1 \quad f_2 \qquad f_{n-2} \quad \tilde{f}_{n-2} \quad f_{n-1} \quad f_{n-1} \quad \tilde{f}_{n-1} \quad f_n \right)$$

Добавим в скобки $-f_0 + f_0$, вынесем общий множитель за скобки:

$$I^* \approx \frac{h}{3} \cdot \left[\frac{f_n - f_0}{2} + \sum_{i=0}^{n-1} \left(f_i + 2\tilde{f}_i \right) \right]$$

$$\frac{h}{3} \cdot \left[\frac{f(b) - f(a)}{2} + \sum_{i=0}^{n-1} \left(f(a+ih) + 2f(a+ih+\frac{h}{2}) \right) \right] \qquad (1.2.6.3.1)$$

Формула Симпсона имеет высокую точность, так как погрешность метода $\delta_м = O(h^3)$

Точность и сходимость методов трапеции, прямоугольников, Симпсона

Формулы для оценки погрешности численного интегрирования методом:

1) прямоугольников

$$\left|R_{np}\right| \le \frac{(b-a)^3}{24n^2} M_2 \qquad (1.2.6.3.2)$$

2) трапеций

$$\left|R_{mp}\right| \le \frac{(b-a)^3}{24n^2} M_2 \qquad (1.2.6.3.3)$$

3) Симпсона

$$\left|R_c\right| \le \frac{(b-a)^5}{2880n^4} M_4 \qquad (1.2.6.3.4)$$

где $M_2 = \max\limits_{x\in[a,b]}\left|f''(x)\right|, \quad M_4 = \max\limits_{x\in[a,b]}\left|f^{(4)}(x)\right|.$

Очевидно, формула Симпсона обладает повышенной точностью:

1) она оказывается точной для $f(x)$, являющихся полиномами до третьей степени включительно, т.к. для этих случаев производная четвертого порядка равна нулю;

2) для достижения той же точности, что и в формуле трапеций, в формуле Симпсона можно брать меньшее число n отрезков разбиения.

1.2.7. АППРОКСИМАЦИЯ

МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Аппроксимация – это процесс определения аналитического вида функции заданной таблично.

Задача аппроксимации сводится к нахождению свободных параметров функции заданного вида, которые обеспечивают наилучшее приближение функции заданной таблично.

Наиболее распространенным методом аппроксимации полинома является аппроксимация методом наименьших квадратов.

Пусть дана функция $y_i = f(x_i)$ $(i = 1, n)$, которая задается таблично.

Пусть

$$P_m = \sum_{i=0}^{m} a_i x^i = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m, \qquad (1.2.7.1.1)$$

который наилучшим образом описывает таблицу (x_i, y_i).

Для нахождения коэффициентов $a_0, a_1, ..., a_m$ используем метод наименьших квадратов.

За меру отклонения полинома $P_m(x)$ от данной функции $f(x)$ на множестве $x_1, x_2, ..., x_n$ принимают величину

$$S_m = \sum_{i=1}^{n} (P_m(x_i) - f(x_i))^2 \qquad (2),$$

равную сумме квадратов отклонений полинома $P_m(x)$ от функции $f(x)$ на заданной системе точек.

Критерий метода наименьших квадратов является минимизация функции многих переменных S_m.

Из условия минимума формируется система линейных уравнений относительно $a_0, a_1, ..., a_m$:

$$\begin{cases} \dfrac{\partial S}{\partial a_0} = 2\sum\limits_{i=1}^{n}(a_0 + a_1 x_i + \ldots + a_m x_i^m - y_i) = 0, \\ \dfrac{\partial S}{\partial a_0} = 2\sum\limits_{i=1}^{n}(a_0 + a_1 x_i + \ldots + a_m x_i^m - y_i)x_i = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{\partial S}{\partial a_0} = 2\sum\limits_{i=1}^{n}(a_0 + a_1 x_i + \ldots + a_m x_i^m - y_i)x_i^m = 0. \end{cases}$$

После преобразований система имеет вид:

$$\begin{cases} \sum\limits_{i=1}^{n}(a_0 + a_1 x_i + \ldots + a_m x_i^m) = \sum\limits_{i=1}^{n} y_i, \\ \sum\limits_{i=1}^{n}(a_0 x_i + a_1 x_i^2 + \ldots + a_m x_i^{m+1}) = \sum\limits_{i=1}^{n} x_i y_i, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \sum\limits_{i=1}^{n}(a_0 x_i^m + a_1 x_i^{m+1} + \ldots + a_m x_i^{2m}) = \sum\limits_{i=1}^{n} x_i^m y_i. \end{cases}$$

После введения обозначений $t_0 = n$; $t_k = \sum\limits_{i=1}^{n} x_i^k$, $k = \overline{0,2m}$; $c_k = \sum\limits_{i=1}^{n} x_i^k \cdot y_i$, $k = \overline{0,m}$;

линейная система имеет вид:

$$\begin{cases} t_0 a_0 + t_1 a_1 + \ldots + t_m a_m = c_0; \\ t_1 a_0 + t_2 a_1 + \ldots + t_{m+1} a_m = c_1; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ t_m a_0 + t_{m+1} a_1 + \ldots + t_{2m} a_m = c_m; \end{cases}$$

Эта линейная система может быть решена любым методом (методом Гаусса или итерационными методами).

1.2.8. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПЕРВОГО ПОРЯДКА.

Общий вид дифференциального уравнения 1-го порядка:

$$F(x, y, y') = 0 . \qquad (1.2.8.1)$$

Если это уравнение разрешимо относительно y', то

$$y' = f(x, y) \quad \text{или} \quad dy = f(x, y) \qquad (1.2.8.2)$$

Общим решением уравнения (1) называется функция

$$y = \varphi(x, C) \qquad (1.2.8.3)$$

от *x* и произвольной постоянной С, обращающая это уравнение в тождество.

Частным решением уравнения (1.18.1) называется решение, полученное из общего решения (1.2.2.8.3) при фиксированном значении С:

$$y = \varphi(x, C_0) , \qquad (1.2.8.4)$$

C_0 - фиксированное число.

*Задача Коши***.** Найти решение $y = f(x, y)$ дифференциального уравнения (1.2.8.1), удовлетворяющее заданным начальным условиям: $y = y_0$ при $x = x_0$. Другими словами: найти интегральную кривую уравнения (1), проходящую через данную точку $M_0(x_0, y_0)$.

Уравнение, содержащее производные от искомой функции *y = y(x)*, называется *обыкновенным дифференциальным уравнением* (ОДУ).

Общий вид дифференциального уравнения:

$$F(x, y, y', y'', ..., y^{(n)}) = 0 \qquad (1.2.8.5)$$

где n – наивысший порядок производной, определяет *порядок уравнения*.

Решением ОДУ называется функция y = y(x), которая после ее подстановки в уравнение (1.18.5) обращает его в тождество.

Общее решение ОДУ имеет вид:

$$y = y(x, C_1, C_2, ..., C_n) \qquad (1.2.8.6)$$

где $C_1, C_2, ..., C_n$ – постоянные интегрирования.

Частное решение получается из общего при конкретных значениях C_i, $i = \overline{1,n}$. Эти значения определяются из *n дополнительных условий*. В качестве таких условий могут быть заданы значения функции и ее производных при некоторых значениях аргумента *x*, иначе говоря, в некоторых точках.

1.2.8.1. Метод Эйлера (метод Рунге-Кутта 1-го порядка)

Разобьем [a, b] на n равных частей – *элементарных отрезков*, $x_0, x_1, \ldots, x_n$ будем называть *узлами сетки*, $h = (b-a)/n$ - *шаг сетки*.

Очевидно, что $x_i = a + i \cdot h$, $i = \overline{0,1,...,n}$; $x_0 = a$, $x_n = b$.

Заменим в уравнении (7.1) y' в точке x_i её приближенной оценкой – отношением приращений (это следует из определения производной):

$$y'_i \approx \frac{\Delta y_i}{\Delta x_i} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{h}$$

Тогда получаем:

$$\frac{y_{i+1} - y_i}{h} \approx f(x_i, y_i)$$

Отсюда формула Эйлера:

$$y_{i+1} \approx y_i + h \cdot f(x_i, y_i) \qquad (1.2.8.1)$$

$x_i = a + i \cdot h$, $i = \overline{0,1,...,n-1}$ – номер узла

Зная y_0 в точке x_0 (начальное условие) можно найти y_1, затем, используя уже известные значения x_1 и y_1, вычислить x_2 и y_2 и так далее.

Если функция $|f| \le M_1$, $\left|\frac{df}{dx}\right| \le M_2$, $\left|\frac{df}{dy}\right| \le M_3$ для $x \in [a,b]$, $y \in \left[y_0, Y\right]$, то имеет место неравенство

$$\left|y(x_i) - y_i\right| \le \frac{M_4 h}{2M_3} e^{M_3 (x_i - x_0)},$$

где $M_4 = M_2 + M_1 M_3$, $h = \max\limits_{0 \le i \le N-1} h_i$.

Оценка имеет лишь теоретическое значение. На практике чаще всего пользуются двойным пересчетом на ЭВМ: расчет на отрезке $\left[x_i, x_{i+1}\right]$ повторяют с шагом $h_i / 2$ и погрешность более точного решения y^*_{i+1} (при шаге $h_i / 2$) оценивают по формуле

$$\left|y^*_{i+1} - y(x_{i+1})\right| \approx \left|y^*_{i+1} - y_{i+1}\right|.$$

Рассмотрим геометрическую иллюстрацию метода Эйлера. В координатах (x,y) отобразим известные данные: отрезок [a,b] на оси X и начальное условие y_0 – точка A с координатами (a, y_0). Отрезок [a,b] разобьем на n равных частей, получим узлы равномерной сетки a = x_0, x_1, x_2, … , x_n = b. Вычислим значения первой производной искомой функции в точке A, используя координату этой точки и исходное уравнение (1.2.8.1)

$$y'(x_0) = f(x_0, y_0) = \operatorname{tg}\alpha_0$$

Полученное значение позволяет построить касательную к искомой функции в точке A. Эту касательную можно использовать для вычисления приближенного значения искомой функции в новом узле x_1 (кривую y(x) заменяем на отрезком AB на элементарном отрезке [x_0, x_1]).

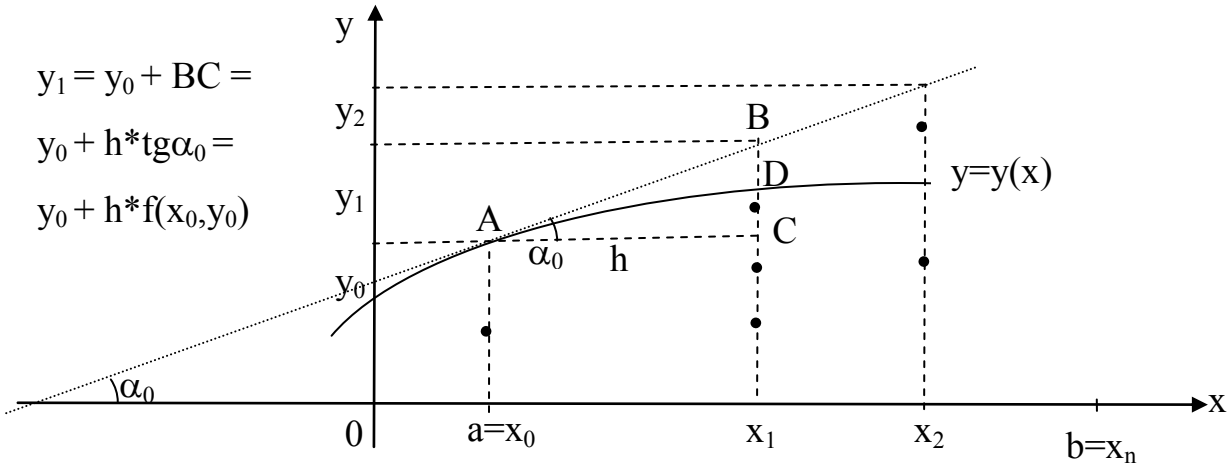


Рисунок 1.2.8.1. Геометрическая иллюстрация метода Эйлера.

Зная (x_1,y_1), можно аналогично получить новую точку (x_2,y_2) и т.д.

Из геометрической иллюстрации следует, что:

1. На каждом шаге есть погрешность (на рисунке это отрезок BD).

Погрешность тем больше, чем больше шаг.

2. Ошибка может накапливаться.

Формула Эйлера (3) имеет погрешность метода $\delta_M = O(h^2)$

Для практического выбора *h* с целью обеспечения заданной точности решения задачи ε применяется следующий прием.

Выполняются 2 расчета: с *n* и 2*n* узлами. Если полученные значения функции в во всех узлах отличаются не более чем на ε, задача считается решенной. Если нет, число узлов вновь удваивают и опять сравнивают полученные значения функций.

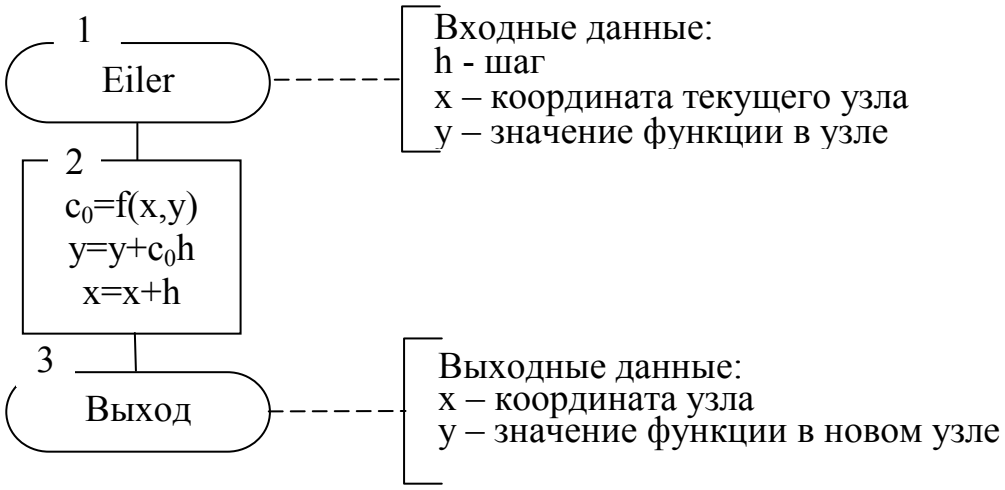
Таким образом, расчет продолжается до достижения условия

$$\delta = \max_{i=1,n} | y_i^n - y_i^{2n} | \le \varepsilon \qquad (1.2.8.2)$$

Значение *n* может достигать большой величины – более 1000. Чтобы не печатать столько значений функции, в алгоритме решения ОДУ методом Эйлера нужно

предусмотреть печать не всех рассчитанных значений, а только части их, например, 10-ти значений, распределенных равномерно по всему отрезку.

Блок-схема



Пример

Дано уравнение $y' - 2y + x^2 = 1$

Найти решение для отрезка [0; 1], если y(0) = 1.

Выберем n = 10, тогда шаг h =(1-0)/10 = 0,1.

Запишем уравнение в каноническом виде $y' = f(x, y) = 1 + 2y - x^2$

Начальная точка $x_0 = 0$, $y_0 = 1$.

Вычислим первую точку

$$y_1 = y_0 + h \cdot f(x_0; y_0) = 1 + 0{,}1 \cdot f(0;\ 1) = 1 + 0{,}1 \cdot (1 + 2 \cdot 1 - 0^2) = 1 + 0{,}1 \cdot 3 = 1{,}3$$

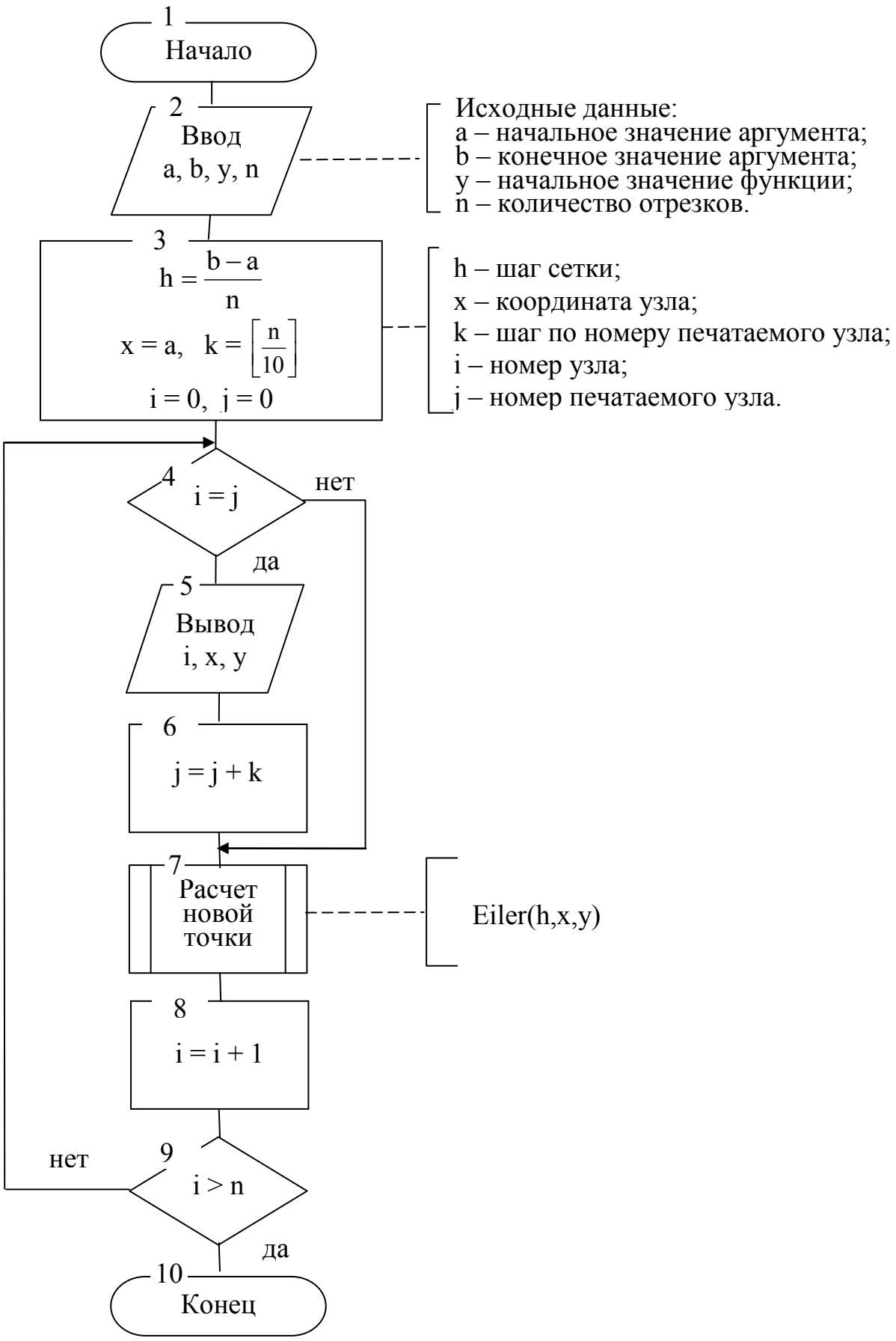
$x_1 = x_0 + h = 0 + 0{,}1 = 0{,}1$

Вычислим вторую точку

$$y_2 = y_1 + h \cdot f(x_1; y_1) = 1{,}3 + 0{,}1 \cdot f(0{,}1; 1{,}3) = 1{,}3 + 0{,}1 \cdot (1 + 2{,}6 - 0{,}01) = 1{,}3 + 0{,}1 \cdot 3{,}59 = 1{,}659$$

$$x_2 = x_1 + h = 0{,}1 + 0{,}1 = 0{,}2$$

Аналогично нужно вычислить еще восемь точек (выбрано n=10).


1
Начало
2
Ввод
a, b, y, n
Исходные данные:
a – начальное значение аргумента;
b – конечное значение аргумента;
y – начальное значение функции;
n – количество отрезков.
3
h = (b – a) / n
x = a, k = [n / 10]
i = 0, j = 0
h – шаг сетки;
x – координата узла;
k – шаг по номеру печатаемого узла;
i – номер узла;
j – номер печатаемого узла.
4
i = j
нет
да
5
Вывод
i, x, y
6
j = j + k
7
Расчет
новой
точки
Eiler(h,x,y)
8
i = i + 1
9
i > n
нет
да
10
Конец

1.2.8.2. Модифицированный метод Эйлера (метод Рунге-Кутта 2-го порядка). Метод Эйлера-Коши

Пусть требуется найти решение задачи Коши:

$$y' = f(x,y)\ ,\ a \le x \le b\ ,\ y(a) = y_0 .$$

Как и в методе Эйлера, на отрезке $[a,b]$ зададим конечное множество точек $\{x_i\}_{i=0}^{N}$, ($a = x_0 < x_1 < ... < x_N = b$). По методу Эйлера-Коши вычисление приближенного решения $y_i \approx y(x_i)$ проводится следующим образом: вначале вычисляется первое приближение

$$\tilde{\ } + h_i f(x_i, y_i),$$

затем находится более точное приближение

$$y_{i+1} = y_i + h_i \frac{f(x_i, y_i) + f(x_{i+1}, \tilde{\ })}{2} .$$

Остаточный член на каждом шаге в методе Эйлера-Коши имеет порядок $O(h^3)$.

Оценка погрешности может быть получена с помощью двойного пересчета на ЭВМ. Расчет повторяют с шагом $h_i/2$, и погрешность более точного решения y_i^* (при шаге $h_i/2$) оценивают приближенно:

$$\left| y_i^* - y(x_i) \right| \approx \frac{1}{3} \left| y_i^* - y_i \right|$$

A- начальная точка.

L_1- касательная к y(x) в точке A.

L_2- касательная к y(x) в середине элементарного отрезка

L_3 параллельно L_2 через т. A

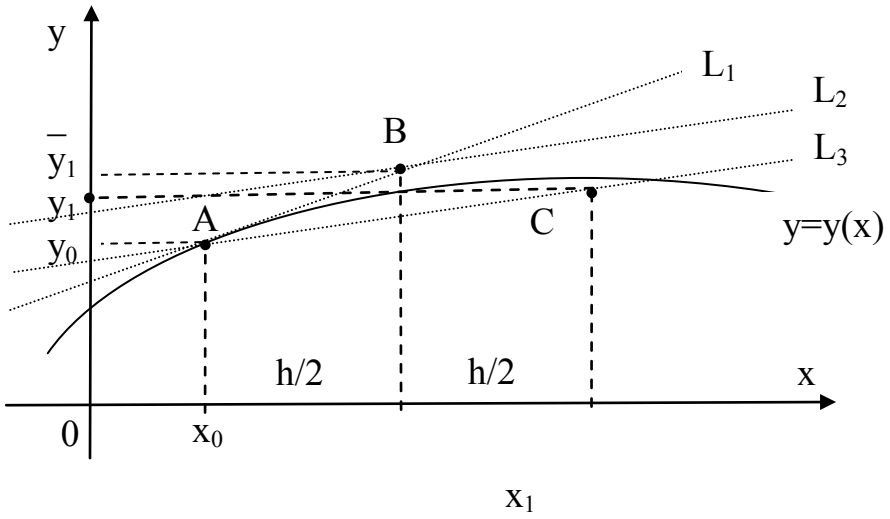


Рисунок 1.2.8.2.1. Геометрическая иллюстрация модифицированного метода Эйлера.

Расчётные формулы:

$\overline{y_1} = y_0 + \frac{h}{2} \cdot f(x_0, y_0)$ - значение функции в середине отрезка $[x_0,x_1]$.

$y_1 = y_0 + h \cdot f(x_0 + \frac{h}{2}, \overline{y_1})$ - значение функции в конце отрезка $[x_0,x_1]$.

Формула модифицированного метода Эйлера:

$$y_{i+1} = y_i + h \cdot f(x_i + \frac{h}{2}, y_i + \frac{h}{2} \cdot f(x_i, y_i)) \qquad (1.2.8.2.1)$$

где $i = 0, 1, ..., n-1$ - номер узла;

$x_i = a + i \cdot h$ - координата узла;

$y_0 = y(x_0)$ - начальное условие.

Алгоритм решения ОДУ отличается от описанного ранее алгоритма метода Эйлера (рис 3) только алгоритмом расчета новой точки.

Погрешность метода $\delta \approx O(h^3)$.

Усовершенствованный метод Эйлера-Коши можно еще более уточнить, применяя *итерационную обработку* каждого значения y_i. Вначале вычисляется

$$y_{i+1}^{(0)} = y_i + h_i f(x_i, y_i),$$

а затем это приближение уточняется по формулам

$$y_{i+1}^{(k+1)} = y_i + h_i / 2(f(x_{i+1}, y_{i+1}^{(k)}) + f(x_i, y_i)).$$

Итерации продолжают до тех пор, пока в пределах требуемой точности два последовательных приближения $y_{i+1}^{(k-1)}$ и $y_{i+1}^{(k)}$ не совпадут. После чего $y_{i+1}^{(k)}$ принимается за приближенное значение $y(x_{i+1})$.

Пример

Решение ранее рассмотренного уравнения модифицированным методом Эйлера.

$y' - 2 \cdot y + x^2 = 1$, $x \in [0;1]$, $y(0) = 1$.

Пусть $n = 10$, $h = (1 - 0)/10 = 0{,}1$.

Начальная точка $x_0 = 0$, $y_0 = 1$.

Расчёт первой точки.

$$y_1 = y_0 + h \cdot f(x_0 + \frac{h}{2}; y_0 + \frac{h}{2} \cdot f(x_0; y_0)) = 1 + 0{,}1 \cdot f(0 + \frac{0{,}1}{2}; 1 + \frac{0{,}1}{2} \cdot f(0; 1)) =$$

$$1 + 0{,}1 \cdot f(0{,}05; 1 + 0{,}05 \cdot (1 + 2 \cdot 1 - 0^2)) = 1 + 0{,}1 \cdot f(0{,}05; 1{,}15) =$$

$$1 + 0{,}1 \cdot (1 + 2 \cdot 1{,}15 - 0{,}05^2) = 1{,}32975$$

$$x_1 = x_0 + h = 0{,}1$$

Аналогично расчёт следующих точек: 2, 3, ... ,10.

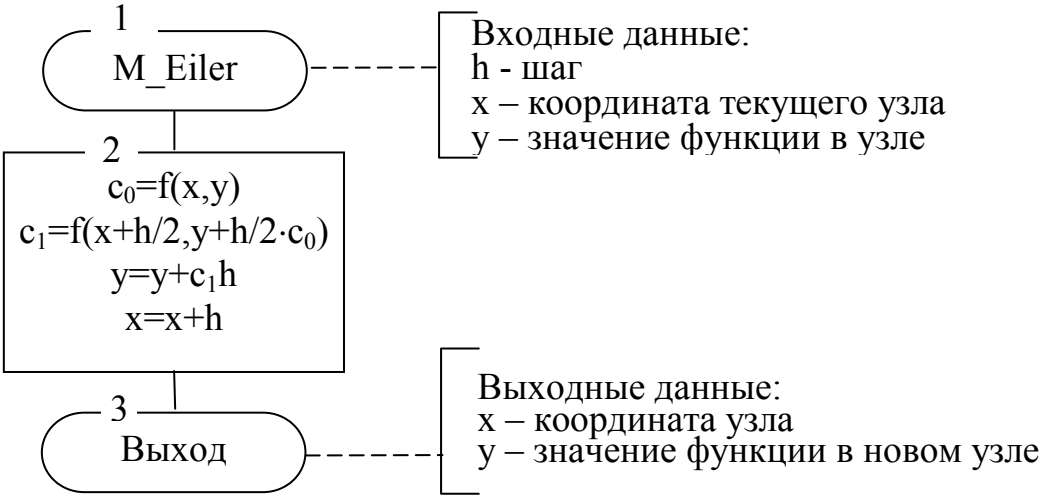


Рисунок 1.2.8.2.2. Алгоритм расчёта новой точки модифицированным методом Эйлера:

1.2.8.3.Метод усредненных точек

Пусть требуется найти решение задачи Коши

$$y' = f(x, y)\ ,\ a \le x \le b\ ,\ y(a) = y_0\ .$$

Как и в методе Эйлера, на отрезке $[a,b]$ зададим конечное множество точек $\{x_i\}_{i=0}^{N}$, ($a = x_0 < x_1 < ... < x_N = b$). По усовершенствованному методу ломаных сначала вычисляются промежуточные значения:

$$x_{i+\frac{1}{2}} = x_i + \frac{h}{2}\ ,\ y_{i+\frac{1}{2}} = y_i + \frac{h}{2} f_i\ ,$$

а затем полагают

$$y_{i+1} = y_i + h f_{i+\frac{1}{2}}\ ,$$

где $f_{i+\frac{1}{2}} = f\left(x_{i+\frac{1}{2}}, y_{i+\frac{1}{2}} \right).$

В этом методе для повышения точности используется усреднённое значение производной на рассматриваемом отрезке:

$$\mathrm{y_{i+1}} = \mathrm{y_i} + \mathrm{h} \cdot \frac{\mathrm{y'_i} + \mathrm{y'_{i+1}}}{2} = \mathrm{y_i} + \frac{\mathrm{h}}{2} \cdot [\mathrm{f(x_i, y_i)} + \mathrm{f(x_{i+1}, y_{i+1})}]$$

В приведённой формуле y_{i+1} входит в обе части уравнения и не может быть выражено явно. Чтобы обойти эту трудность, в правую часть, вместо y_{i+1} подставляется значение, рассчитанное по формуле Эйлера.

$$\overline{\mathrm{y_{i+1}}} = \mathrm{y_i} + \mathrm{h} \cdot \mathrm{f(x_i, y_i)}$$

Получаем формулу исправленного метода Эйлера:

$$y_{i+1} = y_i + \frac{h}{2} \cdot \left[f(x_i, y_i) + f(x_{i+1}, y_i + h \cdot f(x_i, y_i)) \right],\ \mathrm{i} = \overline{0, \mathrm{n} - 1} \qquad (1.2.8.3.1)$$

где i = 0, 1, …., n-1 - номер узла;

$x_i = a + i \cdot h$ - координата узла;

$y_0 = y(x_0)$ - начальное условие.

Погрешность исправленного метода Эйлера $\delta_M = O(h^3)$.

Алгоритм решения ОДУ отличается от описанного ранее алгоритма метода Эйлера только алгоритмом расчета новой точки.

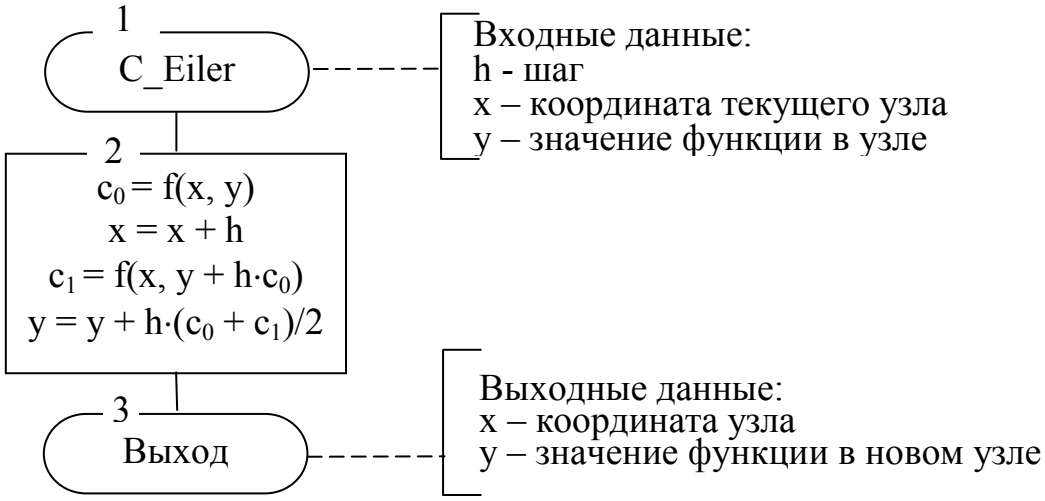
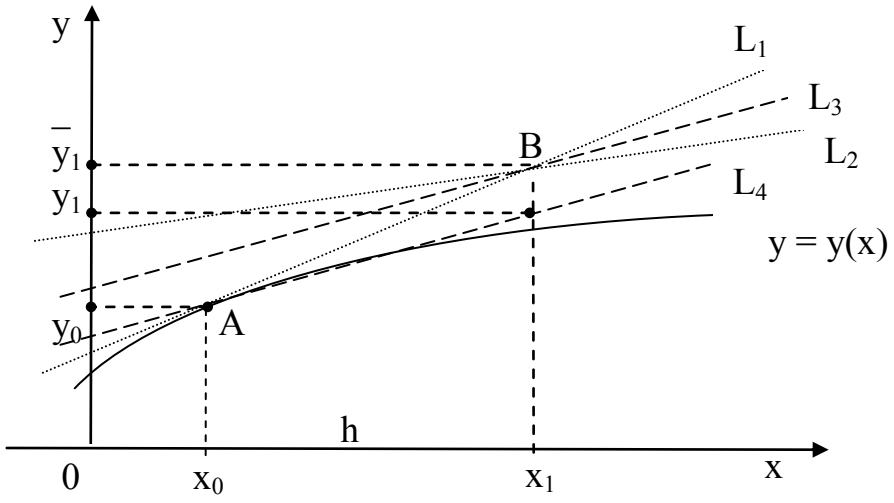


Рисунок 1.2.8.3.1. Алгоритм расчёта новой точки исправленным методом Эйлера:



L_1- касательная к y(x) в начальной точке A, с $tg\alpha_0 = f(x_0, y_0)$.

т. B – значение $\overline{y_1}$ вычисляется по формуле Эйлера.

L_2 – касательная к y(x) в точке B, с $tg\alpha_1 = f(x_1, \overline{y_1})$.

L_3 – прямая через B со среднеарифметическим углом наклона.

L_4 - прямая, паралельная L_3, проведенная через точку A.

Рисунок 1.2.8.3.2. Геометрическая иллюстрация исправленного метода Эйлера.

Пример

Решение ранее рассмотренного уравнения исправленным методом Эйлера.

$y' - 2 \cdot y + x^2 = 1, \quad x \in [0;1], \quad y(0) = 1.$

Пусть $n = 10$, $h = (1 - 0)/10 = 0{,}1$.

Начальная точка $x_0 = 0$, $y_0 = 1$.

Рассчет первой точки.

$$y_1 = y_0 + \frac{h}{2} \cdot \left[f(x_0; y_0) + f(x_0 + h;\ y_0 + h \cdot f(x_0, y_0)) \right] =$$

$$1 + \frac{0{,}1}{2} \cdot \left[f(0;1) + f(0 + 0{,}1;\ 1 + 0{,}1 \cdot f(0, 1)) \right] =$$

$$= 1 + \frac{0{,}1}{2} \cdot [(1 + 2 \cdot 1 - 0^2) + f(0{,}1;\quad 1 + 0{,}1 \cdot (1 + 2 - 0^2))] =$$

$$1 + 0{,}05 \cdot [3 + f(0{,}1;1{,}3)\ ] = 1 + 0{,}05 \cdot [3 + (1 + 2 \cdot 1{,}3 - 0{,}1^2)] =$$

$$= 1 + 0{,}05 \cdot [3 + 3{,}59] = 1 + 0{,}05 \cdot 6{,}59 = 1{,}3295$$

$$x_1 = x_0 + h = 0{,}1$$

Аналогично можно вычислить значения функции во 2, 3, ... , 10 точках.

1.2.8.4. Метод Рунге-Кутта 4 порядка

На практике наибольшее распространение получил метод Рунге-Кутта 4-го порядка, в котором усреднение проводится по трём точкам, формула Эйлера на каждом отрезке используется 4 раза: в начале отрезка, дважды в его середине и в конце отрезка.

Схема численного решения задачи Коши вида

$$y'(x) = f(x, y),\ y(x_0) = y_0$$

по методу Рунге-Кутта четвертого порядка с шагом h имеет вид

$$y_{i+1} = y_i + \Delta y_i \quad (i = 0,1,2,\ldots), \quad \Delta y_i = \frac{1}{6}\left(k_1^i + 2k_2^i + 2k_3^i + k_4^i\right)$$

$$\begin{cases} k_1^i = hf(x_i, y_i) \\ k_2^i = hf(x_i + \frac{h}{2}, y_i + \frac{k_1^i}{2}) \\ k_3^i = hf(x_i + \frac{h}{2}, y_i + \frac{k_2^i}{2}) \\ k_4^i = hf(x_i + h, y_i + k_3^i) \end{cases}$$

Погрешность метода $\delta_M = O(h^5)$.

Схема алгоритма решения ОДУ методом Рунге-Кутта 4-го порядка отличается алгоритмом расчёта новой точки (Рис. 5).

Пример

Решение ранее рассмотренного уравнения (пример 1) методом Рунге-Кутта 4 порядка.

$y' - 2 \cdot y + x^2 = 1$, $x \in [0;1]$, $y(0) = 1$.

Пусть $n = 10$, $h = (1 - 0)/10 = 0,1$.

Начальная точка $x_0 = 0$, $y_0 = 1$.

Рассчет первой точки.

Сначала вычислим значения C_0, C_1, C_2, C_3:

$$k_0 = f(x_0; y_0) = f(0;\ 1) = 1 + 2 \cdot 1 - 0^2 = 3$$

$$k_1 = f(x_0 + \frac{h}{2};\ y_0 + h \cdot \frac{k_0}{2}) = f(0{,}05;\ 1{,}15) = 1 + 2 \cdot 1{,}15 - 0{,}05^2 = 3{,}2975$$

$$k_2 = f(x_0 + \frac{h}{2};\ \ y_0 + h \cdot \frac{k_1}{2}) = f(0{,}05;\ 1{,}164875) = 1 + 2 \cdot 1{,}164875 - 0{,}05^2 = 3{,}32725$$

$$k_3 = f(x_0 + h;\ \ y_0 + h \cdot k_2) = f(0{,}1;\ 1{,}332725) = 1 + 2 \cdot 1{,}332725 - 0{,}1^2 = 3{,}65545$$

Вычислим значение y_1:

$$y_1 = y_0 + \frac{h}{6}\cdot(k_0 + 2\cdot k_1 + 2\cdot k_2 + k_3) =$$

$$1 + \frac{0{,}1}{6}\cdot(3 + 2\cdot 3{,}2975 + 2\cdot 3{,}32725 + 3{,}65545) = 1{,}3317491667$$

Аналогично можно вычислить значения функции во 2, 3, ... , 10 точках.

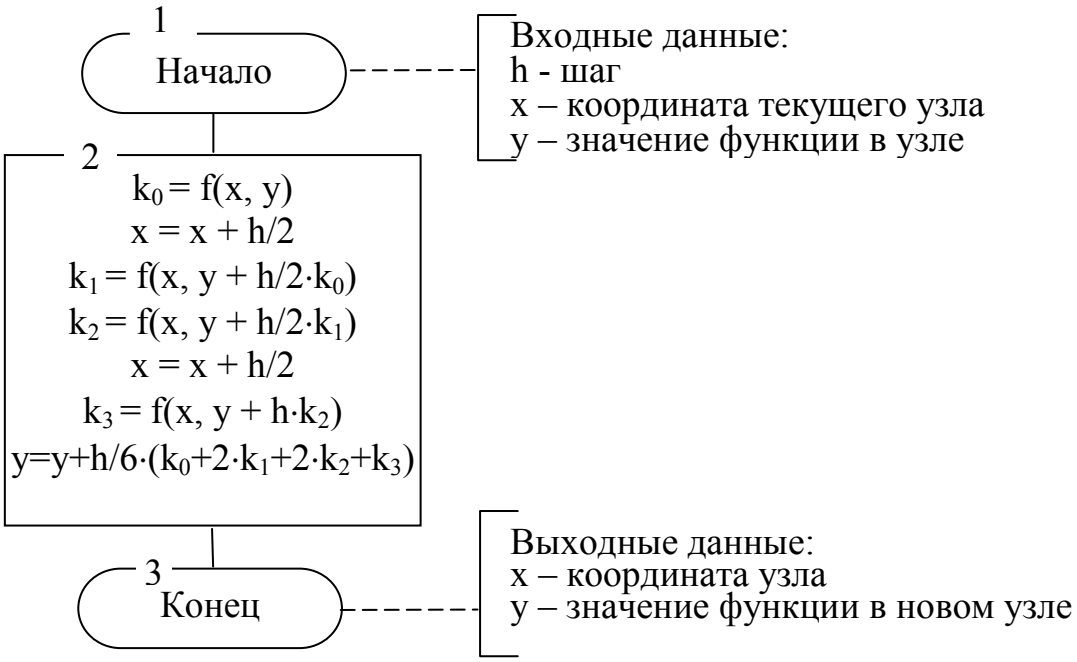


Рисунок 1.2.8.4.1. Схема алгоритма расчета новой точки методом Рунге-Кутта 4-го порядка.

1.2.8.5. Общая характеристика методов

1. Все методы являются *одношаговыми*, то есть для вычисления значения функции в новой точке используется ее значение в предыдущей точке. Это свойство называется *самостартованием*.

Метод Адамса

Экстраполяционный метод Адамса 2 порядка:

$$y_{i+1} = y_i + \frac{h}{2}(3f(t_i, y_i) - f(t_{i-1}, y_{i-1}))$$

Экстраполяционный метод Адамса 3 порядка:

$$y_{i+1} = y_i + \frac{h}{12}(23f(t_i, y_i) - 16f(t_{i-1}, y_{i-1}) + 5f(t_{i-2}, y_{i-2}))$$

Экстраполяционный метод Адамса 4 порядка:

$$y_{i+1} = y_i + \frac{h}{24}\left[55f(t_i, y_i) - 59f(t_{i-1}, y_{i-1}) + 37f(t_{i-2}, y_{i-2}) - \right.$$
$$\left. -9f(t_{i-3}, y_{i-3})\right]$$

1.2.9. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ВЫСШИХ ПОРЯДКОВ

Рассмотрим задачу Коши для ОДУ n-го порядка:

$$y^{(n)} = f(x, y, y', y'', ..., y^{(n-1)}) \qquad (1.2.9.1)$$

с начальными условиями:

$$y(x_0) = y_0,\ y'(x_0) = y'_0,\ y''(x_0) = y''_0, ..., y^{(n-1)}(x_0) = y_0^{(n-1)}.$$

Задача сводится к решению задачи Коши для систем n ОДУ первого порядка.

Обозначим:

$$y' = y_1,\ y'' = y_2, ...,\ y^{(n-1)} = y_{n-1}.$$

Тогда для решения уравнения (1) получаем систему ОДУ:

$$\begin{cases} y' = y_1, \\ y_1' = y_2, \\ \ldots \\ y_{n-2}' = y_{n-1}, \\ y_{n-1}' = f(x, y, y_1, y_2, \ldots, y_{n-1}). \end{cases}$$

с начальными условиями:

$$y(x_0) = y_0,\ y_1(x_0) = y_0',\ y_2(x_0) = y_0'', \ldots,\ y_{n-1}(x_0) = y_0^{(n-1)}.$$

Численное решение обыкновенного дифференциального уравнения 2-го порядка заключается в замене этого уравнения на соответствующую систему из 2 дифференциальных уравнений первого порядка.

Пусть дано уравнение второго порядка

$$y'' = f(x, y, y') \qquad (1.2.9.2)$$

и пусть $y = y(x)$ - точное решение уравнения (2), удовлетворяющее начальным условиям

$$y(x_0) = y_0,\ y'(x_0) = z_0$$

1.2.10. БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ ФУНКЦИЙ

1.2.10.1. Метод Фибоначчи

Требуется найти безусловный минимум функции *f(x)* одной переменной, т.е. такую точку $x^* \in R$, что $f\left(x^*\right) = \min\limits_{x \in R} f(x)$.

Для построения эффективного метода одномерной минимизации, работающего по принципу последовательного сокращения интервала неопределенности, следует задать правило выбора на каждом шаге двух внутренних точек. Желательно, чтобы одна из них всегда использовалась в качестве внутренней и для следующего интервала. Тогда количество вычислений функции сократится вдвое и одна итерация потребует расчета только одного нового значения функции. Метод Фибоначчи реализован на стратегии, обеспечивающая максимальное гарантированное сокращение интервала неопределенности при заданном количестве вычислений функции. Эта стратегия опирается на числа Фибоначчи.

Определение.

Числа Фибоначчи определяются по формуле

$$F_0 = F_1 = 1,\ F_k = F_{k-1} + F_{k-2},\quad k=2,3,4,\ldots$$

Последовательность чисел Фибоначчи имеет вид 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,…

Стратегия поиска

Задается начальный интервал неопределенности и количество N вычислений функции. Алгоритм уменьшения интервала опирается на анализ значений функции в двух точках. Точки вычисления функции находятся с использованием последовательности из N+1 чисел Фибоначчи. Как в методе золотого сечения, на первой итерации требуются два вычисления функции, а на каждой последующей – только по одному. Условия окончания процесса поиска стандартные: поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

Алгоритм

Шаг 1. Задать начальный интервал неопределенности $L_0 = \lfloor a_0, b_0 \rfloor$; $\varepsilon > 0$ - допустимую длину конечного интервала, $\varepsilon > 0$ - константу различимости.

Шаг 2. Найти количество N вычислений функции как наименьшее целое число, при котором удовлетворяется условие $F_N \geq \dfrac{|L_0|}{l}$, и числа Фибоначчи $F_0, F_1, \ldots, F_N$.

Шаг 3. Пусть k=0.

Шаг 4. Вычислить $y_0 = a_0 + \frac{F_{N-2}}{F_N}\left(b_0 - a_0\right);\ z_0 = a_0 + \frac{F_{N-1}}{F_N}\left(b_0 - a_0\right).$

Шаг 5. Вычислить $f(y_k),\ f(z_k)$.

Шаг 6. Сравнить $f(y_k)$ с $f(z_k)$:

a) если $f(y_k) \le f(z_k)$, положить

$$a_{k+1} = a_k;\ b_{k+1} = z_k;\ z_{k+1} = y_k;\ y_{k+1} = a_{k+1} + \frac{F_{N-k-3}}{F_{N-k-1}}\left(b_{k+1} - a_{k+1}\right)$$

b) если $f(y_k) > f(z_k)$, положить

$$a_{k+1} = y_k;\ b_{k+1} = b_k;\ y_{k+1} = z_k;\ z_{k+1} = a_{k+1} + \frac{F_{N-k-2}}{F_{N-k-1}}\left(b_{k+1} - a_{k+1}\right)$$

Шаг 7. Проверить условие окончания и в случае необходимости сделать заключительное N-е вычисление функции для получения решения:

a) если $k \ne N-3$, положить $k = k+1$ и перейти к шагу 5;

b) если $k = N-3$, то всегда $y_{N-2} = z_{N-2} = \frac{\left(a_{N-2} + b_{N-2}\right)}{2}$, т.е. отсутствует точка нового вычисления функции. Следует положить: $y_{N-2} = z_{N-2} = \frac{\left(a_{N-2} + b_{N-2}\right)}{2}$, т.е. отсутствует точка нового вычисления функции. Следует положить: $y_{N-1} = y_{N-2} = z_{N-2}$;

$y_{N-1} = y_{N-1} + \varepsilon$. В точках y_{N-1} и z_{N-1} вычисляются значения функции и находятся границы конечного интервала неопределенности:

- если $f\left(y_{N-1}\right) \le f\left(z_{N-1}\right)$, положить $a_{N-1} = a_{N-2},\ b_{N-1} = z_{N-1}$;
- если $f\left(y_{N-1}\right) > f\left(z_{N-1}\right)$, положить $a_{N-1} = y_{N-1},\ b_{N-1} = b_{N-2}$.

Процесс поиска завершается и $x^* \in \left[a_{N-1}, b_{N-1}\right]$. В качестве приближенного решения можно взять любую точку последнего интервала, например, его середину $x^* \cong \frac{a_{N-1} + b_{N-1}}{2}$.

1.2.10.2. Метод золотого сечения

Требуется найти безусловный минимум функции *f(x)* одной переменной, т.е. такую точку $x^* \in R$, что $f\left(x^*\right) = \min\limits_{x \in R} f(x)$.

Для построения эффективного метода одномерной минимизации, работающего по принципу последовательного сокращения интервала неопределенности, следует задать правило выбора на каждом шаге двух внутренних точек. Желательно, чтобы одна из них всегда использовалась в качестве внутренней и для следующего интервала. Тогда количество вычислений функции сократится вдвое и одна итерация потребует расчета только одного нового значения функции. В методе золотого сечения в качестве двух внутренних точек выбираются точки золотого сечения.

Определение.

Точка производит «золотое сечение» отрезка, если отношение длины всего отрезка к большей части равно отношению большей части к меньшей.

На отрезке $\left[a_0, b_0\right]$ имеются две симметричные относительно его концов точки y_0 и z_0:

$$\frac{b_0 - a_0}{b_0 - y_0} = \frac{b_0 - y_0}{y_0 - a_0} = \frac{b_0 - a_0}{z_0 - a_0} = \frac{z_0 - a_0}{b_0 - z_0} = \frac{1 + \sqrt{5}}{2} \cong 1{,}618 .$$

Кроме того, точка y_0 производит золотое сечение отрезка $\left[a_0, z_0\right]$, а точка z_0 - отрезка $\left[y_0, b_0\right]$.

Алгоритм

Шаг 1. Задать начальный интервал неопределенности $L_0 = \left[a_0, b_0\right]$; точность $l > 0$.

Шаг 2. Пусть *k*=0.

Шаг 3. Вычислить $y_0 = a_0 + \frac{3-\sqrt{5}}{2}\left(b_0 - a_0\right);\ z_0 = a_0 + b_0 - y_0,\ \frac{3-\sqrt{5}}{2} \cong 0{,}38196$.

Шаг 4. Вычислить $f(y_k),\ f(z_k)$.

Шаг 5. Сравнить $f(y_k)$ с $f(z_k)$:

c) если $f(y_k) \le f(z_k)$, положить

$a_{k+1} = a_k;\ b_{k+1} = z_k;\ y_{k+1} = a_{k+1} + b_{k+1} - y_k,\ z_{k+1} = y_k$. Перейти к шагу 6;

d) если $f(y_k) > f(z_k)$, положить

$a_{k+1} = y_k;\ b_{k+1} = b_k;\ y_{k+1} = z_k;\ z_{k+1} = a_{k+1} + b_{k+1} - z_k$.

Шаг 6. Вычислить $\Delta = \left|a_{k+1} - b_{k+1}\right|$ и проверить условие окончания:

a) если $\Delta \le l$, то процесс поиска завершается и $x^* \in \left[a_{k+1}, b_{k+1}\right]$. В качестве приближенного решения можно взять середину последнего интервала: $x^* \cong \frac{a_{k+1} + b_{k+1}}{2}$;

b) если $\Delta > l$, положим *k*=*k*+1 и перейдем к шагу 4.

Электронный учебно-методический комплекс

Практический раздел

ИНФОРМАТИКА

Лабораторные задания

Минск 2016

2. ЛАБОРАТОРНЫЕ ЗАДАНИЯ

Лабораторная работа № 1.ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЯ

Вычислить следующие выражения

Варианты заданий

Номер варианта	**Выражение**	**Исходные данные**
1	$a = \ln(y^{-\sqrt{\lvert x\rvert}}) \cdot \left(\sin(x) + e^{(x+y)}\right)$	x, y
2	$b = \sqrt{c(\sqrt{y} + x^2)} \cdot (\cos(x) - \lvert c - y\rvert)$	c, x, y
3	$c = arctg(x) - \frac{3}{5}e^{xy} + 0{,}5\frac{\lvert x+y\rvert}{(x+y)^b}$	b, x, y
4	$d = \frac{e^{\lvert x-y\rvert} \cdot tg(z)}{arctg(y) + \sqrt{x}} + \ln(x)$	x, y, z
5	$e = \frac{(\cos(x) - \sin(y))^3}{\sqrt{tg(z)}} + \ln^2(x \cdot y \cdot z)$	x, y, z
6	$f = y^x + \sqrt{\lvert x\rvert + e^y} - \frac{z^3 \cdot \sin^2(y)}{y + z^2/(y-x)}$	x, y, z
7	$g = \frac{1 + \cos(x+y)}{\lvert e^x - 2y/(1 + x^2 \cdot y^2)\rvert} \cdot x^3 + \arcsin(y)$	x, y
8	$h = 2 + \frac{x^2}{\sqrt{2}} + \frac{\lvert y^3\rvert}{\sqrt{2}} + \frac{z^4 \cdot (\ln(x) + 1) \cdot \sqrt{2}}{\sqrt{3}}$	x, y, z
9	$j = \left((1+y) \cdot \sqrt{\sin^2(z)} - \frac{\lvert y - x\rvert}{5}\right)^3$	x, y, z
10	$k = \ln\left\lvert(y - \sqrt{\lvert x\rvert}\right\rvert \cdot (x - \frac{y}{z + x^2/4})$	x, y, z
11	$l = 0{,}5x^2 + 3 \cdot \cos(x+y) + e^{-0{,}1y \cdot z} - \sqrt{\lvert x \cdot y\rvert}$	x, y, z

12	$m = \sqrt{e^x + tg(x) + 1} \cdot (\lg(y) + \cos(x \cdot y) + \sqrt[3]{x})$	x, y
13	$p = \dfrac{\lg(x) - e^{x+y}}{\sqrt{2} + y^2 + \lvert x^3 - \ln(y) \rvert}$	x, y
14	$p = \dfrac{\lg(x) - e^{x+y}}{\sqrt{2} + y^2 + \lvert x^3 - \ln(y) \rvert}$	x, y
15	$q = \sqrt{12x^4 - 3x^2 + 4x^2 - 5x + 6} - \lg^2(z)$	x, z
16	$\alpha_1 = \dfrac{ax^2 + \sin^2 z}{\sqrt{1 + e^a}}$	a, x, z
17	$t_2 = \dfrac{\beta^2 + \sqrt{\lvert q \rvert}}{\cos^2 x + \beta \ln x}$	β , z, x
18	$q_3 = \dfrac{\sin^2 (z + a)^3}{t\sqrt[3]{e^{2+3t}}}$	z, a, t

Контрольные вопросы

1. Как описываются константы и переменные?
2. Назвать стандартные типы данных.
3. Описать процедуру стандартного ввода-вывода.
4. Назвать составные части программы на языке **C++**.
5. Изобразить схематически блок-схему линейного алгоритма.

Лабораторная работа № 2. ОПЕРАТОР IF

Вычислить с помощью оператора **if**

Варианты заданий

Номер варианта	Выражение	Исходные данные
1	$a=\begin{cases}(x+y)^2-\sqrt{x\cdot y}, & x\cdot y>0\\ (x+y)^2+\sqrt{\lvert x\cdot y\rvert}, & x\cdot y<0\\ (x+y)^2+1, & x\cdot y=0\end{cases}$	*x, y*
2	$b=\begin{cases}\ln(x/y)+(x^2+y)^3, & x/y>0\\ \ln\lvert x/y\rvert+(x^2+y)^3, & x/y<0\\ (x^2+y), & x=0\\ 0, & y=0\end{cases}$	*x, y*
3	$c=\begin{cases}x^2+y^2+\sin(x), & x-y=0\\ (x-y)^2+\cos(x), & x-y>0\\ (y-x)^2+tg(x), & x-y<0\end{cases}$	*x, y*
4	$d=\begin{cases}(x-y)^3+arctg(x), & x>y\\ (y-x)^3+arctg(x), & y>x\\ (y+x)^3+0{,}5, & y=x\end{cases}$	*x, y*
5	$e=\begin{cases}i\cdot\sqrt{a}, & i-нечетное, a>0\\ \frac{i}{2}\sqrt{\lvert a\rvert}, & i-четное, a<0\\ \sqrt{\lvert i\cdot a\rvert}, & иначе\end{cases}$	*i, a*
6	$g=\begin{cases}e^{\lvert a\rvert-\lvert b\rvert}, & 0{,}5<a\cdot b<10\\ \sqrt{\lvert a+b\rvert}, & 0{,}1<a\cdot b<0{,}5\\ 2x^2, & иначе\end{cases}$	*a, b, x*

7	$h=\begin{cases} arctg(x+\lvert y\rvert), & x<y \\ arctg(\lvert x\rvert+y), & x>y \\ (x+y)^2, & x=y \end{cases}$	*x, y*
8	$j=\begin{cases} \sin(5k+3\cdot m\cdot\lvert k\rvert), & -1<k<m \\ \cos(5k+3\cdot m\cdot\lvert k\rvert, & k>m \\ k^3, & k=m \end{cases}$	*k, m*
9	$l=\begin{cases} 3k^3+3p, & k>\lvert p\rvert \\ \lvert k-p\rvert, & 3<k<\lvert p\rvert \\ (k-p)^2, & k=\lvert p\rvert \end{cases}$	*k, p*
10	$k=\begin{cases} \ln(\lvert f\rvert+\lvert q\rvert), & \lvert f\cdot q\rvert>10 \\ e^{f+q}, & \lvert f\cdot q\rvert<10 \\ f+q, & \lvert f\cdot q\rvert=10 \end{cases}$	*f, q*
11	$y=\begin{cases} ax^3+b\ln\lvert 2x\rvert, & \sqrt{\lvert a-b\rvert}<x \\ \sqrt{\lvert a+\sin 2x\rvert}-b^{3x}, & \sqrt{\lvert a-b\rvert}=x \\ \dfrac{arctg5x}{b\cos x+\lg ax}, & \sqrt{\lvert a-b\rvert}>x \end{cases}$	*a, b, x*
12	$y=\begin{cases} \ln x^2-e^{ax+b}+\lg\lvert a-b\rvert, & 2a+b<0 \\ tg(ax-b^2)-b\lvert x^{-3}\rvert, & 2a+b=0 \\ arctg(2x-0{,}5)+\sqrt{\lvert a+bx\rvert}, & 2a+b>0 \end{cases}$	*a, b, x*
13	$y=\begin{cases} \dfrac{ax-e^x+b^3}{\ln(2x^2+4)}+\cos(4x-0{,}2), & \lvert a^2-b^2\rvert<10x \\ b\sin(x^3-a)-e^{-1,4x}, & \lvert a^2-b^2\rvert=10x \\ tg4x+\dfrac{\lvert x^{-5}\rvert}{\sin 0{,}5x}, & \lvert a^2-b^2\rvert>10x \end{cases}$	*a, b, x*
14	$y=\begin{cases} a\sqrt{\lvert\sin x+\cos^2 x\rvert}+e^{a+bx}, & \sqrt{\lvert bx-65\rvert}<a \\ 1-\ln\sqrt{\lvert ax^3-b\rvert-\lvert x^{-10}\rvert}, & \sqrt{\lvert bx-65\rvert}=a \\ \dfrac{(x^3-b)\cdot\cos(3x-0{,}5)}{\lg x^3-a\sin(a-b)}, & \sqrt{\lvert bx-65\rvert}>a \end{cases}$	*a, b, x*

15	$y=\begin{cases}\ln\left\|\dfrac{x^{-9}}{ax-b}\right\|-e^{2x^2}, & \sin x<\sqrt{a^2+b^2}\\ 4arctg6x+\dfrac{ax+7}{\sqrt{\left\|b^2-x\right\|}}, & \sin x=\sqrt{a^2+b^2}\\ x^3+2{,}3ax+\sqrt{b^2\left\|tgx\right\|}, & \sin x>\sqrt{a^2+b^2}\end{cases}$	*a, b, x*
16	$\alpha_1=\begin{cases}\sqrt{\sin x^3}+\ln^2 z, & при\ \ z<\sqrt[3]{ax}\\ e^{\beta\cdot z}+\left\|x^3\right\|, & при\ \ z=\sqrt[3]{ax}\\ \cos\sqrt{z}+\lg\beta^2 x, & при\ \ z>\sqrt[3]{ax}\end{cases}$	*x, z, a, β*
17	$\beta_2=\begin{cases}3q+\sqrt{\left\|x^3+\sin z\right\|}, & при\ \ \sin z<q\\ \sqrt[5]{\alpha_3+q^3+e^{\alpha\cdot z}}, & при\ \ \sin z=q\\ \left\|\alpha^5\right\|\cdot\ln\sqrt{\cos z}, & при\ \ \sin z>q\end{cases}$	*q, x, z, α*
18	$\gamma_3=\begin{cases}\alpha^3+\sqrt[5]{\cos y^2}, & при\ \ y<\ln\beta\\ \lg(x+w)^2+\left\|y^3\right\|, & при\ \ y=\ln\beta\\ \sqrt{tgy^5}+e^{xw}, & при\ \ y>\ln\beta\end{cases}$	*α , y, x, ω*

Контрольные вопросы

1. Формат записи оператора **IF**.
2. Формат записи с вложенным оператором **IF**.
3. Как работает оператор **IF**?
4. В чем отличие короткой и полной формы записи оператора **IF**?
5. Изобразить схематически блок-схему разветвляющего алгоритма.

Лабораторная работа № 3. ОПЕРАТОР SWITCH

Вычислить с помощью оператора **SWITCH**.

Варианты заданий

Номер варианта	**Выражение**	**Исходные данные**
1	$a=\begin{cases}(x+y)^2-\sqrt{x\cdot y}, & \kappa=1\\ (x+y)^2+\sqrt{\lvert x\cdot y\rvert}, & \kappa=12\\ (x+y)^2+1, & \kappa=100\end{cases}$	x, y, к
2	$b=\begin{cases}\ln(x/y)+(x^2+y)^3, & \alpha=a\div d\\ \ln\lvert x/y\rvert+(x^2+y)^3, & \alpha=k\div p\\ (x^2+y), & \alpha=r\div y\\ 0, & \textit{иначе}\end{cases}$	x, y, α
3	$c=\begin{cases}x^2+y^2+\sin(x), & \beta=10,12,23\\ (x-y)^2+\cos(x), & \beta=14,18,24\\ (y-x)^2+tg(x), & \beta=100,120,235\end{cases}$	x, y, β
4	$d=\begin{cases}(x-y)^3+arctg(x), & z=1\div 10\\ (y-x)^3+arctg(x), & z=12\div 34\\ (y+x)^3+0{,}5, & \textit{иначе}\end{cases}$	x, y, z
5	$e=\begin{cases}i\cdot\sqrt{a}, & \lambda=1,2,3\\ \frac{i}{2}\sqrt{\lvert a\rvert}, & \lambda=5,6,7\\ \sqrt{\lvert i\cdot a\rvert}, & \textit{иначе}\end{cases}$	i, a, λ
6	$g=\begin{cases}e^{\lvert a\rvert-\lvert b\rvert}, & s=1\\ \sqrt{\lvert a+b\rvert}, & s=12\\ 2x^2, & \textit{иначе}\end{cases}$	a, b, x, s

7	$h=\begin{cases} arctg(x+\lvert y\rvert), & \eta=1 \\ arctg(\lvert x\rvert+y), & \eta=12,\ 23,56 \\ (x+y)^2, & \textit{иначе} \end{cases}$	x, y, η
8	$j=\begin{cases} \sin(5k+3\cdot m\cdot\lvert k\rvert), & \alpha=a\div d \\ \cos(5k+3\cdot m\cdot\lvert k\rvert, & \alpha=k\div t \\ k^3, & \textit{иначе} \end{cases}$	k, m, α
9	$l=\begin{cases} 3k^3+3p, & w=12 \\ \lvert k-p\rvert, & w=34 \\ (k-p)^2, & w=89 \end{cases}$	k, p, w
10	$k=\begin{cases} \ln(\lvert f\rvert+\lvert q\rvert), & h=1\div 23 \\ e^{f+q}, & h=34\div 89 \\ f+q, & h=90\div 123 \end{cases}$	f, q, h
11	$y=\begin{cases} ax^3+b\ln\lvert 2x\rvert, & m=12\div 45 \\ \sqrt{\lvert a+\sin 2x\rvert}-b^{3x}, & m=46\div 67 \\ \dfrac{arctg5x}{b\cos x+\lg ax}, & \textit{иначе} \end{cases}$	a, b, x, m
12	$y=\begin{cases} \ln x^2-e^{ax+b}+\lg\lvert a-b\rvert, & \tau=123\div 234 \\ tg(ax-b^2)-b\lvert x^{-3}\rvert, & \tau=236\div 345 \\ arctg(2x-0{,}5)+\sqrt{\lvert a+bx\rvert}, & \tau=1000 \end{cases}$	a, b, x, τ
13	$y=\begin{cases} \dfrac{ax-e^x+b^3}{\ln(2x^2+4)}+\cos(4x-0{,}2), & z=1 \\ b\sin(x^3-a)-e^{-1,4x}, & z=56 \\ tg4x+\dfrac{\lvert x^{-5}\rvert}{\sin 0{,}5x}, & z=123 \end{cases}$	a, b, x, z
14	$y_1=\begin{cases} a\sqrt{\lvert\sin x+\cos^2 x\rvert}+e^{a+bx}, & y=12,45,78 \\ 1-\ln\sqrt{\lvert ax^3-b\rvert-\lvert x^{-10}\rvert}, & y=456 \\ \dfrac{(x^3-b)\cdot\cos(3x-0{,}5)}{\lg x^3-a\sin(a-b)}, & \textit{иначе} \end{cases}$	a, b, x, y

15	$y=\begin{cases}\ln\left\lvert\dfrac{x^{-9}}{ax-b}\right\rvert-e^{2x^2}, & s=34\div 56\\ arctg6x+\dfrac{ax+7}{\sqrt{\lvert b^2-x\rvert}}, & s=57\div 78\\ x^3+2{,}3ax+\sqrt{b^2\lvert tgx\rvert}, & \textit{иначе}\end{cases}$	a, b, x, s
16	$\alpha_1=\begin{cases}\sqrt{\sin x^3}+\ln^2 z, & a1=12\div 34\\ e^{\beta\cdot z}+\lvert x^3\rvert, & a1=35\div 56\\ \cos\sqrt{z}+\lg\beta^2 x, & \textit{иначе}\end{cases}$	x, z, a, β , a1
17	$\beta_2=\begin{cases}3q+\sqrt{\lvert x^3+\sin z\rvert}, & \beta=a\div c\\ \sqrt[5]{\alpha_3+q^3+e^{\alpha\cdot z}}, & \beta=d\div m\\ \lvert\alpha^5\rvert\cdot\ln\sqrt{\cos z}, & \textit{иначе}\end{cases}$	q, x, z, α ,β
18	$\gamma_3=\begin{cases}\alpha^3+\sqrt[5]{\cos y^2}, & d=1\\ \lg(x+w)^2+\lvert y^3\rvert, & d=12,45,89\\ \sqrt{tgy^5}+e^{xw}, & \textit{иначе}\end{cases}$	α , y, x, ω ,d

Контрольные вопросы

1. Формат записи оператора **SWITCH**.
2. Как работает оператор **SWITCH**?
2. В чем отличие оператора **SWITCH** от **IF**?
3. Типы селектора.
4. Изобразить схематически блок-схему разветвляющего алгоритма.

Лабораторная работа № 4. ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Составить таблицу значений функции при $x_н<x<x_к$ с шагом h вещественного типа.

Выполнить задания 2 способами:

- С использованием оператора **FOR**;
- С использованием оператора **WHILE**;

Варианты заданий

Номер варианта	**Выражение**	**Исходные данные**
1	$a = \ln(y^{-\sqrt{\lvert x\rvert}}) \cdot \left(\sin(x) + e^{(x+y)}\right)$	$x_н<x<x_к$, $y=const$
2	$b = \sqrt{c(\sqrt{y} + x^2)} \cdot (\cos(x) - \lvert c - y\rvert)$	$x_н<x<x_к$, $y,c=const$
3	$c = arctg(x) - \frac{3}{5}e^{xy} + 0{,}5\frac{\lvert x+y\rvert}{(x+y)^b}$	$x_н<x<x_к$, $y,b=const$
4	$d = \frac{e^{\lvert x-y\rvert} \cdot tg(z)}{arctg(y) + \sqrt{x}} + \ln(x)$	$x_н<x<x_к$, $y,z=const$
5	$e = \frac{(\cos(x) - \sin(y))^3}{\sqrt{tg(z)}} + \ln^2(x \cdot y \cdot z)$	$x_н<x<x_к$, $y,z=const$
6	$f = y^x + \sqrt{\lvert x\rvert + e^y} - \frac{z^3 \cdot \sin^2(y)}{y + z^2/(y-x)}$	$x_н<x<x_к$, $y,z=const$
7	$g = \frac{1 + \cos(x+y)}{\lvert e^x - 2y/(1 + x^2 \cdot y^2)\rvert} \cdot x^3 + \arcsin(y)$	$x_н<x<x_к$, $y=const$
8	$h = 2 + \frac{x^2}{\sqrt{2}} + \frac{\lvert y^3\rvert}{\sqrt{2}} + \frac{z^4 \cdot (\ln(x) + 1) \cdot \sqrt{2}}{\sqrt{3}}$	$x_н<x<x_к$, $y,z=const$
9	$j = \left((1+y) \cdot \sqrt{\sin^2(z)} - \frac{\lvert y-x\rvert}{5}\right)^3$	$x_н<x<x_к$, $y,z=const$
10	$k = \ln\lvert(y - \sqrt{\lvert x\rvert}\rvert \cdot (x - \frac{y}{z + x^2/4})$	$x_н<x<x_к$, $y,z=const$

11	$l = 0{,}5x^2 + 3 \cdot \cos(x+y) + e^{-0{,}1y \cdot z} - \sqrt{\lvert x \cdot y \rvert}$	$x_н<x<x_к$, $y,z=const$
12	$m = \sqrt{e^x + tg(x) + 1} \cdot (\lg(y) + \cos(x \cdot y) + \sqrt[3]{x})$	$x_н<x<x_к$, $y=const$
13	$p = \dfrac{\lg(x) - e^{x+y}}{\sqrt{2 + y^2 + \lvert x^3 - \ln(y) \rvert}}$	$x_н<x<x_к$, $y=const$
14	$p = \dfrac{\lg(x) - e^{x+y}}{\sqrt{2 + y^2 + \lvert x^3 - \ln(y) \rvert}}$	$x_н<x<x_к$, $y=const$
15	$q = \sqrt{12x^4 - 3x^2 + 4x^2 - 5x + 6} - \lg^2(z)$	$x_н<x<x_к$, $z=const$
16	$\alpha = \dfrac{ax^2 + \sin^2 z}{\sqrt{1 + e^a}}$	$x_н<x<x_к$, $a,z=const$
17	$t = \dfrac{\beta^2 + \sqrt{\lvert q \rvert}}{\cos^2 x + \beta \ln x}$	$x_н<x<x_к$, $\beta,z=const$
18	$q = \dfrac{\sin^2(z+a)^3}{t\sqrt[3]{e^{2+3t}}}$	$t_н<t<t_к$, $a,z=const$

Контрольные вопросы

1. Формат записи оператора **FOR**.
2. Формат записи оператора **WHILE**.
3. Составить блок-схему циклического алгоритма для всех операторов цикла.

Лабораторная работа № 5. ВЫЧИСЛЕНИЕ КОНЕЧНЫХ СУММ

Варианты заданий

Вычислить конечные суммы.

№ варианта	Сумма	Диапазон	n
1	$1+\frac{\ln 3}{1!}x+\frac{\ln^2 3}{2!}x^2+\ldots+\frac{\ln^n 3}{n!}x^n$	$0{,}1 \le x \le 1$	10
2	$\cos x+\frac{\cos 2x}{2!}+\ldots+\frac{\cos nx}{n!}$	$\frac{\pi}{5} \le x \le \frac{9\pi}{5}$	10
3	$x-\frac{x^3}{3!}+\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$	$0{,}1 \le x \le 1$	10
4	$1+\frac{x}{1!}+\frac{x^2}{2!}+\ldots+\frac{x^n}{n!}$	$1 \le x \le 2$	10
5	$1+\frac{\cos\frac{\pi}{4}}{1!}x+\ldots+\frac{\cos n\frac{\pi}{4}}{n!}x^n$	$0{,}1 \le x \le 1$	15
6	$1-\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$	$0{,}1 \le x \le 1$	10
7	$1+\frac{\cos x}{1!}+\ldots+\frac{\cos nx}{n!}$	$0{,}1 \le x \le 1$	10
8	$1+3x^2+\ldots+\frac{2n+1}{n!}x^{2n}$	$0{,}1 \le x \le 1$	10
9	$1+\frac{x^2}{2!}+\ldots+\frac{x^{2n}}{(2n)!}$	$0{,}1 \le x \le 1$	10
10	$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	$0{,}1 \le x \le 1$	10
11	$1+2\frac{x}{2!}+\ldots+\frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$	$0{,}1 \le x \le 1$	10
12	$1-\frac{3}{2!}x^2+\ldots+(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$	$0{,}1 \le x \le 1$	15
13	$-\frac{(2x)^2}{2!}+\frac{(2x)^4}{4!}+\ldots+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	$0{,}1 \le x \le 1$	15

14	$-\frac{x}{1!}+\frac{x}{2!}+\ldots+(-1)^n\frac{x}{n!}$	$0{,}1\le x\le 1$	15
15	$\frac{e^x}{2!}+\frac{e^x}{4!}+\frac{e^x}{6!}+\ldots+\frac{e^x}{2n!}$	$2\le x\le 4$	10
16	$-x+\frac{\sqrt{x}}{2!}-\frac{\sqrt[3]{x}}{3!}+\ldots+(-1)^n\frac{\sqrt[n]{x}}{n!}$	$10\le x\le 20$	10
17	$1+\frac{\sin^2 x}{3!}+\frac{\sin^4 x}{5!}+\ldots+\frac{\sin^{2n} x}{(2n+1)!}$	$\frac{\pi}{2}\le x\le\frac{3\pi}{2}$	10
18	$1+\frac{x^2}{3!}+\frac{x^4}{5!}+\ldots+\frac{x^{2n}}{(2n+1)!}$	$0{,}1\le x\le 1$	10

Контрольные вопросы

4. Что такое конечные суммы?
5. Как осуществляется ввод-вывод конечных сумм?
6. Составить блок-схему для вычисления конечных сумм.

Лабораторная работа №6. ОДНОМЕРНЫЕ МАССИВЫ

Варианты заданий

1. Найти наименьший из положительных элементов массива (X1, X2,…,X14).
2. Записать в массив Y десять первых положительных элементов массива (X1, X2,…,X20).
3. Вычислить сумму элементов массива (A1, A2,…,A12), стоящих на четных местах.
4. Подсчитать количество положительных и отрицательных элементов в массиве (A1, A2,…,A25). Нулевые элементы не считать.
5. Для целочисленного массива (B1, B2,…,B10) определить, является ли сумма его элементов четным числом, и вывести на печать «Да» или «Нет».
6. Записать +1 вместо максимального элемента массива (X1, X2,…,X12), а -1 вместо минимального.
7. Переписать положительные элементы массива (X1, X2,…,X30) подряд в массив Y.
8. Определить разность между наибольшим и наименьшим элементами массива (Y1,Y2,…,Y20).
9. Все элементы массива (A1, A2,…, A25) уменьшить на величину среднего арифметического этих элементов.
10. Массив (X1, X2,…,X30) разбить на два массива: массив положительных элементов и массив отрицательных элементов. Нуль относить к положительным элементам.
11. Переписать элементы массива (X1, X2,…,X20) в обратном порядке.
12. Подсчитать сумму элементов массива (A1, A2,…,A22), стоящий на четных местах, и сумму элементов того же массива стоящих на нечетных местах.
13. Вывести на печать номера элементов (Y1, Y2,…,Y15), удовлетворяющих условию $0<Yi<1$.
14. В массе (A1, A2,…,A25) поменять местами минимальный и максимальный элементы.
15. Дан массив (D1, D2,…,D25). Найти произведение элементов с четными номерами и сумму с нечетными номерами.
16. В массиве (F1, F2,…,F10) найти сумму элементов с номерами, кратными трем.
17. В массиве (A1, A2,…,A50) найти количество элементов, равных X.
18. Из вектора (A1, A2,…,A15) получить вектор (B1, B2,…,B5), очередная компонента которого равна среднему арифметическому очередной тройки компонент вектора A.

Контрольные вопросы

1. Что такое массив?
2. Какие существуют варианты инициализации массива?
3. Как вычислить объем памяти, необходимый для хранения массива?

Лабораторная работа № 7. ДВУМЕРНЫЕ МАССИВЫ

Варианты заданий

1. Из матрицы A(3x5) построить матрицу B, поменяв местами строки и столбцы.
2. Дана матрица H(4x5). Найти номер строки, имеющей максимальную сумму элементов.
3. Дана матрица C(5x5). Вывести на экран элементы главной диагонали и найти сумму их квадратов.
4. Дана матрица G(4x5). Определить координаты её минимального элемента и сам элемент, вывести их на экран.
5. В матрице A(4x3) поменять местами строку, содержащую максимальный элемент массива, со строкой, содержащей минимальный элемент.
6. Вычислить и вывести на экран разность между произведением элементов главной диагонали и элементов побочной диагонали матрицы B(5x5).
7. Вычислить сумму элементов под главной диагональю квадратной матрицы E (5x5).
8. Определить минимальный элемент в каждом из столбцов матрицы C(4x6) и вывести на экран номера строк, в которых они находятся.
9. Вычислить сумму элементов строки и столбца матрицы D(4x5), на пересечении которых находится минимальный элемент матрицы.
10. В матрице X(5x5) найти сумму квадратов элементов, расположенных выше главной диагонали.
11. Описать переменную и присвоить её значение наибольшего из элементов матрицы A(5x5), расположенных на главной диагонали и выше ее.
12. В матрице C(4x5) определить все положительные элементы, их сумму вывести на экран.
13. Заполнить одномерный массив элементами главной диагонали матрицы R(5x5), вывести его на экран.

14. Определить и вывести на экран максимальный и минимальный элементы матрицы B(4x4).
15. Определить и вывести на экран минимальный положительный и максимальный отрицательный элементы матрицы M(5x4).
16. Вывести на экран координаты трёх наибольших элементов матрицы T(5x5).

Контрольные вопросы

1. Для чего применяется двумерный массив и в чём заключается его взаимосвязь с одномерным массивом?
2. Как описать в программе двумерный массив?
3. Как заполнить двумерный массив данными?
4. Какие основные элементы блок-схемы программы двумерного массива?
5. Что такое главная диагональ двумерного массива?

Лабораторная работа №8. ПОДПРОГРАММЫ

Варианты заданий

При выполнении заданий вычисления выполнять в подпрограмме. В теле основной программы осуществить ввод массива и обратиться к подпрограмме.

1. Найти наименьший из положительных элементов массива (A1, A2, …, A15).
2. Записать в массив C десять первых положительных элементов массива (A1, A2, …, A20).
3. Вычислить сумму чётных элементов массива (A1, A2, …, A15).
4. Подсчитать количество положительных и отрицательных элементов в массиве (A1, A2, …, A30). Ноль не является ни положительным, ни отрицательным числом.
5. Дан целочисленный массив из 10 элементов. Определить, является ли сумма его элементов чётным числом, и вывести на экран буквенный ответ.
6. В массиве из 12 элементов заменить максимальный элемент числом +1, а минимальный – числом -1.
7. Дан массив из 30 элементов. Записать все положительные элементы данного массива в новый массив.
8. Определить разность между наибольшим и наименьшим элементами массива (A1, A2, …, A20).
9. Все элементы массива (B1, B2, …, B20) уменьшить на величину среднего арифметического этих элементов.
10. Массив, состоящий из 30 элементов, разбить на два массива: массив положительных элементов и массив отрицательных элементов. Ноль относить к положительным элементам.
11. Переписать элементы массива (C1, C2, …, C20) в обратном порядке.
12. Вычислить сумму элементов, стоящих на чётных позициях и элементов, стоящих на нечётных позициях, массива (A1, A2, …, A20).
13. В массиве, состоящем из 25 элементов, поменять местами максимальный и минимальный элементы.
14. Дан массив из 45 элементов. Вычислить произведение элементов на чётных позициях и сумму элементов на нечётных позициях.
15. В массиве, состоящем из 70 элементов, найти сумму элементов с номерами, кратными трём.
16. Дан массив из 50 элементов. Подсчитать количество элементов, равных A.

Контрольные вопросы

1. Что такое подпрограммы и зачем они применяются.
2. Привести пример программы, в которой использование подпрограмм позволит значительно уменьшить количество кода.
3. Как осуществляется описание подпрограммы, как ввести входные данные в подпрограмму?
4. Как осуществить вывод данных из подпрограммы и какие типы вывода возможны?
5. Как изобразить блок-схему программы с подпрограммой?

Лабораторная работа №9. ФАЙЛЫ

Варианты заданий

Создать текстовый файл исходных данных. В программе элементы массива считывать из этого файла. Результат преобразований записать в новый текстовый файл.

1. Из матрицы A(3x5) построить матрицу B, поменяв местами строки и столбцы.
2. Дана матрица H(4x5). Найти номер строки, имеющей максимальную сумму элементов.
3. Дана матрица C(5x5). Вывести на экран элементы главной диагонали и найти сумму их квадратов.
4. Дана матрица G(4x5). Определить координаты её минимального элемента и сам элемент, вывести их на экран.
5. В матрице A(4x3) поменять местами строку, содержащую максимальный элемент массива, со строкой, содержащей минимальный элемент.
6. Вычислить и вывести на экран разность между произведением элементов главной диагонали и элементов побочной диагонали матрицы B(5x5).
7. Вычислить сумму элементов под главной диагональю квадратной матрицы E (5x5).
8. Определить минимальный элемент в каждом из столбцов матрицы C(4x6) и вывести на экран номера строк, в которых они находятся.
9. Вычислить сумму элементов строки и столбца матрицы D(4x5), на пересечении которых находится минимальный элемент матрицы.
10. В матрице X(5x5) найти сумму квадратов элементов, расположенных выше главной диагонали.
11. Описать переменную и присвоить её значение наибольшего из элементов матрицы A(5x5), расположенных на главной диагонали и выше ее.
12. В матрице C(4x5) определить все положительные элементы, их сумму вывести на экран.
13. Заполнить одномерный массив элементами главной диагонали матрицы R(5x5), вывести его на экран.
14. Определить и вывести на экран максимальный и минимальный элементы матрицы B(4x4).
15. Определить и вывести на экран минимальный положительный и максимальный отрицательный элементы матрицы M(5x4).
16. Вывести на экран координаты трёх наибольших элементов матрицы T(5x5).

Контрольные вопросы

1. Какие типы файлов можно создать в C++ и какие типы данных можно записывать в данные файлы?
2. Зачем числовые переменные в файлах разделять нечисловыми символами?
3. Назвать основные команды для записи и чтения файлов в C++.
4. Какие основные особенности работы с файлами в C++?
5. Привести пример кода с записью некоторых данных в файл и последовательно описать операции, которые при этом происходят.

Лабораторная работа №10. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Решить нелинейные уравнения методами:

1. бисекции
2. хорд
3. простой итерации
4. Ньютона

Варианты заданий

Вариант	Уравнение	Интервал изоляции корня
1	$x^2 - 5\sin x = 0$	[1,57;3,14]
2	$e^x - 10x = 0$	[0;1]
3	$x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$	[0,4;1]
4	$\sin x - x + 0{,}15 = 0$	[0,5;1]
5	$x - \sqrt{9 + x} + x^2 - 4 = 0$	[2;3]
6	$0{,}1x - x\ln x = 0$	[1;2]
7	$x^4 - 26x^3 + 131x^2 - 226x + 120 = 0$	[19,5;21,2]
8	$x^4 - 0{,}486x^3 - 5{,}792x^2 + 0{,}486x + 4{,}792 = 0$	[2;3]
9	$0{,}1\sin x + x^3 - 1 = 0$	[0,8;1,0]
10	$0{,}1e^x - \sin^2 x + 0{,}5 = 0$	[-2;1]
11	$e^x - x - 1{,}25 = 0$	[0,618;0,667]
12	$\sin x - 2x + 0{,}5 = 0$	[0,4;0,5]
13	$e^x - 2(x-1)^2 = 0$	[0;1]
14	$x - 1{,}25\ln x - 1{,}25 = 0$	[2,2;2,4]

15	$3\sin\sqrt{x}+0{,}35x-3{,}8=0$	[2;3]
16	$\sqrt{1-x}-tgx=0$	[0;1]

Контрольные вопросы

1. В чем сущность методов бисекции, хорды, простой итерации, Ньютона?

2. Как выбирается начальное приближение в методах простой итерации и Ньютона?

3. К какому виду нужно преобразовать уравнение для метода простой итерации?

Лабораторная работа №10. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Решить системы ЛАУ методом Гаусса

Варианты заданий

№ примера	**Система уравнений**	**Точность ε**
1	$\begin{cases} x+y+z=2 \\ 2x-y-z=-1 \\ x-y+z=0 \end{cases}$	10^{-5}
2	$\begin{cases} 2{,}7x+3{,}3y+1{,}3z=2{,}1 \\ 3{,}5x-1{,}7y+2{,}8z=1{,}7 \\ 4{,}1x+5{,}8y-1{,}7z=0{,}8 \end{cases}$	10^{-5}
3	$\begin{cases} 2x-y-z=0 \\ y+z=5 \\ 0{,}5x+y+z=2 \end{cases}$	10^{-5}
4	$\begin{cases} 3{,}1x+2{,}8y+1{,}9z=0{,}2 \\ 1{,}9x+3{,}1y+2{,}1z=2{,}1 \\ 7{,}5x+3{,}8y+4{,}8z=5{,}6 \end{cases}$	10^{-5}
5	$\begin{cases} 6x+y-z=10 \\ x-y+2z=5 \\ -x-y+6z=40 \end{cases}$	10^{-5}
6	$\begin{cases} 3{,}6x+1{,}8y-4{,}7z=3{,}8 \\ 2{,}7x-3{,}6y+1{,}9z=0{,}4 \\ 1{,}5x+4{,}5y+3{,}3z=-1{,}6 \end{cases}$	10^{-5}

7	$\begin{cases} 0,1x - y - z = 2 \\ x + y + z = 1 \\ x + y - z = -8 \end{cases}$	10^{-5}
8	$\begin{cases} 2,7x + 0,9y - 1,5z = 3,5 \\ 4,5x - 2,8y + 6,7z = 2,6 \\ 5,1x + 3,7y - 1,4z = -0,14 \end{cases}$	10^{-5}
9	$\begin{cases} x + 2y + z = 1 \\ -x - 2y + 2z = 1 \\ y + z = 2 \end{cases}$	10^{-5}
10	$\begin{cases} 3,8x + 6,7y - 1,2z = 5,2 \\ 6,4x + 1,3y - 2,7z = 3,8 \\ 2,4x + 4,5y + 3,5z = -0,6 \end{cases}$	10^{-5}
11	$\begin{cases} 0,5x - y + 0,5z = 3 \\ x + y + z = 1 \\ -x + 0,5y - z = -1 \end{cases}$	10^{-5}
12	$\begin{cases} 2,74x - 1,18y + 3,17z = 2,18 \\ 1,12x + 0,83y - 2,16z = -1,15 \\ 0,81x + 1,27y + 0,76z = 3,23 \end{cases}$	10^{-5}
13	$\begin{cases} -2x - y = 0,333 \\ -x + 2y - z = 1 \\ y + 2z = -0,333 \end{cases}$	10^{-5}
14	$\begin{cases} 10x + y + z = 12 \\ 2x + 10y + z = 13 \\ 2x + 2y + 10z = 14 \end{cases}$	10^{-5}
15	$\begin{cases} 10x + y + z = 10 \\ 2x + 10y + z = 13 \\ 2x + y + z = 5 \end{cases}$	10^{-5}

Контрольные вопросы

1. К какому виду приводится матрица в методе Гаусса?
2. В каком случае нельзя применить метод Гаусса?
3. Что нужно предусмотреть при использовании метода Гаусса?

Лабораторная работа №11. ИТЕРАЦИОННЫЕ МЕТОДЫ

Решить системы линейных алгебраических уравнений
методами:

1. простой итерации;
2. метод Зейделя.

Варианты заданий

№ примера	**Система уравнений**	**Точность ε**
1	$\begin{cases} 4x+0,24y-0,08z=8 \\ 0,09x+3y-0,15z=9 \\ 0,04x-0,08y+4z=20 \end{cases}$	10^{-5}
2	$\begin{cases} 2x-y+z=-3 \\ 3x+5y-2z=1 \\ x-4y+10z=0 \end{cases}$	10^{-5}
3	$\begin{cases} 10x+y+z=12 \\ 2x+10y+z=13 \\ 2x+2y+10z=14 \end{cases}$	10^{-5}
4	$\begin{cases} 2x-y=0,333 \\ -x+2y-z=1 \\ -y+2z=-0,333 \end{cases}$	10^{-5}

5	$\begin{cases} 7,6x+0,5y+2,4z=1,9 \\ 2,2x+9,1y+4,4z=9,7 \\ -1,3x+0,2y+5,8z=-1,4 \end{cases}$	10^{-5}
6	$\begin{cases} 25x+y-3,5z=5 \\ 9,4y-3,4z=-3 \\ x-y+7,3z=0 \end{cases}$	10^{-5}
7	$\begin{cases} 2,7x+3,3y+1,3z=2,1 \\ 3,5x-1,7y+2,8z=1,7 \\ 4,1x+5,8y-1,7z=0,8 \end{cases}$	10^{-5}
8	$\begin{cases} 3,1x+2,8y+1,9z=0,2 \\ 1,9x+3,1y+2,1z=2,1 \\ 7,5x+3,8y+4,8z=5,6 \end{cases}$	10^{-5}
9	$\begin{cases} 3,6x+1,8y-4,7z=3,8 \\ 2,7x-3,6y+1,9z=0,4 \\ 1,5x+4,5y+3,3z=-1,6 \end{cases}$	10^{-5}
10	$\begin{cases} 2,7x+0,9y-1,5z=3,5 \\ 4,5x-2,8y+6,7z=2,6 \\ 5,1x+3,7y-1,4z=-0,14 \end{cases}$	10^{-5}
11	$\begin{cases} 3,8x+6,7y-1,2z=5,2 \\ 6,4x+1,3y-2,7z=3,8 \\ 2,4x-4,5y+3,5z=-0,6 \end{cases}$	10^{-5}
12	$\begin{cases} 1,02x-0,05y-0,1z=0,795 \\ -0,11x+1,03y-0,05z=0,849 \\ -0,11x-0,12y+1,04z=1,398 \end{cases}$	10^{-5}
13	$\begin{cases} 6x+2y+z=12 \\ 2x+4y-z=16 \\ x-y+5z=15 \end{cases}$	10^{-5}
14	$\begin{cases} 10x-2y-2z=6 \\ -x+10y-2z=7 \\ -x-y+10z=8 \end{cases}$	10^{-5}

15	$\begin{cases} 2x - y + z = -3 \\ 3x + 5y - 2z = 1 \\ x - 4y + 10z = 0 \end{cases}$	10^{-5}

Контрольные вопросы

1. Чем отличаются точные методы от итерационных?
2. Как выбираются начальные приближения в методах простой итерации и Зейделя?
3. Каково условие прекращения итерации в методе простой итерации и в методе Зейделя для решения систем линейных алгебраических уравнений?

Лабораторная работа №12. ИНТЕРПОЛИРОВАНИЕ

Задание

Решить задания методами:

3. линейной интерполяции
4. Лагранжа

Варианты заданий

№ вар.	Условие							f(x)
1	**2**							**3**
1	*x*	93	96,2	100	104,2	108,7		*f*(102)
	f(x)	11,38	12,8	14,7	17,07	19,91		
2	*x*	0	2	3	6	7	9	*f*(5)
	f(x)	658503	704969	729000	804357	830584	884736	
3	*x*	2	2,3	2,5	3,0	3,5	3,8	4	*f*(3,75)
	f(x)	5,848	6,127	6,3	6,694	7,047	7,243	7,368	
4	*x*	14	17		31	35			*f*(20)
	f(x)	68,7	64		44	39,1			
5	*x*	10	15		17	20			*x* при
	f(x)	3	7		11	17			*f*(*x*)=10
6	*x*	1	2		2,5	3			*x* при
	f(x)	-6	-1		5,625	16			*f*(*x*)=10
7	*x*	4	6		8	10			*x* при
	f(x)	11	27		50	83			*f*(*x*)=10
8	*x*	1	1,1	1,2	1,3	1,4			*f*(1,13)
	f(x)	1,1752	1,33565	1,50946	1,69838	1,9043			
9	*x*	1,5	1,6		1,7	1,8			*f*(1,75)
	f(x)	2,12928	2,37587		2,64563	2,94217			
10	*x*	1	1,1		1,2	1,3	1,4		*f*(1,23)
	f(x)	0,6827	0,7287		0,7699	0,8064	0,8385		

№ вар.	Условие							*f(x)*
1	**2**							**3**
11	*x*	1,5	1,6	1,6	1,7	1,8	1,9	*f*(1,61)
	f(x)	0,8664	0,8904	0,8904	0,9109	0,9281	0,9426	
12	*x*	0,3	0,4	0,5	0,6	0,7		*f*(0,55)
	f(x)	0,2913	0,3799	0,4621	0,5380	0,6044		
13	*x*	0,8	0,9	1,0	1,1			*f*(0,87)
	f(x)	0,664	0,7163	0,7616	0,8005			
14	*x*	1	1,1		1,2	1,3	1,4	*f*(1,25)
	f(x)	1	0,95135		0,91817	0,98747	0,88726	
15	*x*	2	2,2		2,4	2,6		*x* при *f*(*x*)=0,1
	f(x)	0,224	0,1104		0,0025	-0,0968		
16	*x*	0,1	0,15		0,19	0,25		*x* при *f*(*x*)=0,2
	f(x)	1,1052	1,1618		1,2092	1,284		
17	*x*	0,2	0,24		0,26	0,29		*f*(0,21)
	f(x)	1,2214	1,2712		1,2969	1,3364		
18	*x*	0,1	0,13		0,17	0,2		*f*(0,15)
	f(x)	0,0998	0,1296		0,1692	0,1987		
19	*x*	0,1	0,15		0,18	0,22		*x* при *f*(*x*)=0,16
	f(x)	1,1052	1,1618		1,1972	1,2466		
20	*x*	0,2	0,24		0,27	0,3		*f*(0,29)
	f(x)	1,2214	1,2712		1,31	1,35		

Контрольные вопросы

1. В каких случаях прибегают к интерполяции?
2. В чем заключается метод линейной интерполяции?
3. В чем заключается метод Лагранжа?

Лабораторная работа №13. РЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ НЬЮТОНА

Варианты заданий

№ примера	Система Уравнений	Точность ε
1	$\begin{cases}\sin x - y = 0 \\ \cos y - x + 1{,}386 = 0\end{cases}$	10^{-3}
2	$\begin{cases}x^7 - 5x^2y^2 + 1510 = 0 \\ y^5 - 3x^4y - 105 = 0\end{cases}$	10^{-5}
3	$\begin{cases}5x - 6y + 20\lg x + 16 = 0 \\ 2x - y - 10\lg y - 4 = 0\end{cases}$	10^{-3}
4	$\begin{cases}x^3 - y^2 - 1 = 0 \\ xy^3 - y - 4 = 0\end{cases}$	10^{-5}
5	$\begin{cases}\sin(x+1) - y = 0 \\ 2x + \cos y = 2\end{cases}$	10^{-3}
6	$\begin{cases}x^2y^2 - 3x^3 - 6y^3 + 8 = 0 \\ x^2 - 9y + 2 = 0\end{cases}$	10^{-5}
7	$\begin{cases}\cos\frac{1}{3}(x-y) - 2y = 0 \\ \sin\frac{1}{3}(x+y) - 2x = 0\end{cases}$	10^{-3}
8	$\begin{cases}e^{xy} = x^2 - y + 1{,}1 \\ (x+0{,}5)^{1{,}1} + y^2 = 0{,}1\end{cases}$	10^{-5}
9	$\begin{cases}\sin(x-y) - 1{,}3x = 0{,}1 \\ x^2 - y^2 = \frac{3}{4}\end{cases}$	10^{-5}

10	$\begin{cases}\sin(x+y)-1{,}3x=0{,}1\\ x^2+y^2=1\end{cases}$	10^{-2}
11	$\begin{cases}\sin(x+y)-1{,}3x=0{,}1\\ x^2+y^2=1\end{cases}$	10^{-5}
12	$\begin{cases}x^{10}+y^{10}=1024\\ e^x-e^y=1\end{cases}$	10^{-2}
13	$\begin{cases}x^3-y^2=1\\ xy^3-y-4=0\end{cases}$	10^{-3}
14	$\begin{cases}\sin(x-2y)-xy+1=0\\ x^2-y^2=1\end{cases}$	10^{-2}
15	$\begin{cases}x+3\lg x-y^2=0\\ 2x^2-xy-5x+1=0\end{cases}$	10^{-5}

Контрольные вопросы

1. Как выбрать начальное приближение в методе Ньютона да решения систем нелинейных уравнений?
2. Как определяется матрица Якоби?
3. Какое должно выполняться условие сходимости в методе Ньютона?

Лабораторная работа №14. РЕШЕНИЕ ЗАДАЧИ АППРОКСИМАЦИИ

Выполнить задания методом наименьших квадратов

Варианты заданий

№ вар.	Выборка							
1	x	0	1,5	3	4,5	6		
	y	0	2	3	3,5	3,8		
2	x	0	0,9	1,5	1,7	1,7	1,6	
3	x	2	2,4	3,2	4,5	6,5		
	y	0	1	2	3	4		
4	x	3	4,1	4,5	4,3	3,3	1,8	
	y	3	4	5	6	7	8	
5	x	2,5	3	3,5	4	4,5	5	5,5
	y	0	0,6	1,2	1,8	2,9	4,5	6,5
6	x	0	1	2	3	4		
	y	3	4,5	4,5	3,6	2,1		
7	x	0	1	2	3	4		
	y	6,3	3,8	2,5	1,85	1,4		
8	x	4	5	6	7	8		
	y	6	5,5	5,2	5	4,9		
9	x	1,6	2	3	4	5	6	
	y	6	3,4	2,8	3,3	4,5	5,6	
10	x	4	5	6	7	8		
	y	1	0,1	0,7	1,75	3,1		
11	x	0	0,1		0,2	0,3	0,4	
	y	1	-1,2		-2,4	-3	-3,5	
12	x	0	0,1		0,2	0,3	0,4	
	y	3	0,2		0,6	1,75	3,1	
13	x	0,3	0,4		0,5	0,6	0,7	
	y	-1	1,1		1,65	1,25	-0,5	

14	x	0,1	0,2	0,3	0,4	0,5
	y	1,75	2	2,55	4,35	5,5
15	x	0,3	0,4	0,5	0,6	0,7
	y	-2,85	-2,55	-2,1	-1,4	0
16	x	0,4	0,5	0,6	0,7	
	y	5	3	2,1	1,5	
17	x	0	0,1	0,2	0,3	0,4	0,5
	y	2	3,6	4,5	4,85	4,3	3
18	x	0,3	0,4	0,5	0,6	0,7	
	y	2	1,3	1,6	2,4	3,6	
19	x	0,1	0,2	0,3	0,4	0,5	
	y	4	2	0,9	0	-0,25	
20	x	0	0,1	0,2	0,3	0,4	0,5
	y	-3	-2	-1,2	-1,1	-1,5	-2

Контрольные вопросы

1. Что такое аппроксимация?
2. В чем заключается задача аппроксимации?
3. В чем заключается критерий метода наименьших квадратов?

Лабораторная работа №15. ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННОГО ИНТЕГРАЛА

Вычислить определенный интеграл

методами:

4. трапеций
5. прямоугольников
6. Симпсона

Варианты заданий

Вар.	**Подъинтегральная функция**	**Интервал интег-рирования [a,b]**	**Кол. частей разбие-ния N**	**Шаг h**	**Первообразная функция F(x)**
1	$\frac{x}{(x+3)^2}$	[0;2]	40	0,05	$\frac{3}{x+3}+\ln(x+3)$
2	$\frac{\sqrt{4-x^2}}{x}$	[0;2]	80	0,01	$\sqrt{4-x^2}-2\ln\frac{2+\sqrt{4-x^2}}{x}$
3	$x\cdot\sin 2x$	[0,2;1]	20	$\pi/80$	$\frac{\sin 2x}{4}-\frac{x\cos 2x}{2}$
4	2^{3x}	$[0;\frac{\pi}{4}]$	50	0,02	$\frac{2^{3x}}{3\ln 2}$
5	$\frac{\ln^2 x}{x}$	[0;1]	50	0,08	$\frac{\ln^3 x}{3}$
6	$e^{2x}\sin x$	[1;5]	30	$\pi/60$	$\frac{e^{2x}}{5}(2\sin x-\cos x)$

7	$\frac{x^2}{2x+3}$	$[0; \frac{\pi}{2}]$	100	0,02	$\frac{1}{8}(2x^2-6x+9\ln(2x+3))$
8	$x^2\sqrt{x+2}$	[1;3]	50	0,06	$\frac{2((15x^2-24x+32)}{105}\sqrt{(x+2)^3}$
9	$\frac{x}{\sin^2 3x}$	[0,2;1]	25	0,04	$-\frac{x}{3}ctg3x+\frac{1}{9}\ln\sin 3x$
10	$x\cdot e^{0.8x}$	[2;3]	40	0,025	$\frac{e^{0,8x}}{0,64}(0,8x-1)$
11	$\frac{1}{x\sqrt{x^2+0,25}}$	[1;2]	50	0,02	$-21n(\frac{0,5+\sqrt{x^2+0.25}}{x})$
12	$\frac{x^2}{(2x+0,3)^2}$	[1;2]	80	0,0125	$0,25x-0,075\ln(2x+3)-\frac{0,01125}{2x+0,3}$
13	$\frac{x}{0,5x+0,1}$	[3;5]	50	0,04	$\frac{8(0,5x-0,2)}{3}\sqrt{0,5x+0,1}$
14	$x^2\sin x$	[0;1]	40	0,025	$2x\sin x-(x^2-2)\cos x$
15	$x\cdot 2^{3x}$	[1;4]	60	0,05	$\frac{x\cdot 2^{3x}}{3\ln 2}-\frac{2^{3x}}{9(\ln 2)^2}$

Контрольные вопросы

1. Геометрический смысл определенного интеграла.
2. Какой зависимостью связан шаг интегрирования с количеством интервалов?
3. Какой из методов вычисления определенного интеграла является самым точным, и как это определяется?

Лабораторная работа №16

РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПЕРВОГО ПОРЯДКА

Решить дифференциальные уравнения методами:

1. Эйлера
2. Модифицированный метод Эйлера
3. Рунге-Кутта

Варианты заданий

Вариант	**Дифференциальное уравнение**	**Начальные условия**	**Интервал интегрирования**	**Точное решение**
1	$y' + y \cdot tgx = \frac{1}{\cos x}$	$y(\pi) = 5$	**[π ; 2π]**	$y = -5\cos x + \sin x$
2	$y' + 3y = e^{2x} y^2$	$y(0) = 1$	**[0;2]**	$y = e^{-2x}$
3	$y' - \frac{y}{x-3} = \frac{y^2}{x-3}$	$y(1) = -2$	**[1;1,5]**	$y = \frac{x-3}{2-x}$
4	$y' = 2y - x + e^x$	$y(0) = -1$	**[0;1,5]**	$y = \frac{1}{2}x - e^x + \frac{1}{4}\left(1 - e^{2x}\right)$
5	$(x^2 - x)y' + y = x^2(2x-1)$	$y(-2) = 2$	**[-2;-1]**	$y = x^2 - \frac{3x}{x-1}$
6	$x \cdot y' = x \cdot \sin\frac{y}{x} + y$	$y(2) = \pi$	**[2;3]**	$y = 2x \cdot arctg(x/2)$
7	$xy' = y(1 + \ln y - \ln x)$	$y(1) = e^2$	**[1;1,5]**	$y = x \cdot e^{2x}$
8	$y' = y/\left(3x - y^2\right)$	$y(0) = 1$	**[0;1]**	$x = y^2 - y^3$

9	$x(y'-y)=e^x$	$y(1)=0$	**[1;2]**	$y=e^x\ln x$
10	$xy'-2y=2x^4$	$y(1)=0$	**[1;2]**	$y=x^4-x^2$
11	$y'=2x(x^2+y)$	$y(0)=0$	**[0;1]**	$y=x^2+1-e^{x^2}$
12	$y'-y=e^x$	$y(0)=1$	**[0;1]**	$y=(x+1)e^x$
13	$yx'+x=4y^3+3y^2$	$y(2)=1$	**[2;3]**	$x=y^3+y^2$
14	$x^2y'+xy+1=0$	$y(1)=0$	**[1;2]**	$y=-(\ln x)/x$
15	$y=x(y'-x\cdot\cos x)$	$y(\pi/2)=0$	**[$\pi/2$;1]**	$y=(\sin x-1)x$
16	$y'+ytgx=\sec x$	$y(0)=0$	**[0;1]**	$y=\sin x$

Контрольные вопросы

1. В чем заключается задача Коши для решения обыкновенных дифференциальных уравнений 1-го порядка?
2. Насколько точнее модифицированный метод Эйлера от простого?
3. В чем отличие метода усредненных точек от метода Эйлера-Коши?

Лабораторная работа №17

РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ВЫСШИХ ПОРЯДКОВ

Решить дифференциальные уравнения методом Эйлера

Варианты заданий

Вариант	**Дифференциальное уравнение**	**Начальные условия**	**Интер-вал Интег-ри-рования**	**Точное решение**
1	$y'' - 2y' + y = 0$	$y(2) = 1;$ $y'(2) = -2$	**[2;4]**	$y = (7 - 3x)e^{x-2}$
2	$y'' - 3y' + 2y - 2x + 3 = 0$	$y(0) = 1;$ $y'(0) = 2.$	**[0;2]**	$y = e^x + x$
3	$y'' + y = 4e^x$	$y(0) = 4;$ $y'(0) = -3.$	**[0;1]**	$y = 2\cos x - 5\sin x + 2e^x$
4	$x^2 y'' + xy' = 0$	$y(0) = 5;$ $y'(0) = -1.$	**[0;1,5]**	$y = 5 - \ln x$
5	$y'' - 2y' = 2e^x$	$y(1) = -1;$ $y'(1) = 0.$	**[1;2]**	$y = e^{2x-1} - 2e^x + e - 1$
6	$y'' + 4y = \cos 3x$	$y(0) = 0.8;$ $y'(0) = 2.$	**[0;1]**	$y = \cos 2x + \sin 2x - 0{,}2\cos 3x$
7	$y'' + 2y' + 2y = xe^x$	$y(0) = 0;$ $y'(0) = 0.$	**[0;1,5]**	$y = e^{-x}(x - \sin x)$
8	$(1 + x^2)y'' + (y')^2 + 1 = 0$	$y(0) = 1;$ $y'(0) = 1.$	**[0;0,5]**	$y = 1 - x + 2\ln(1 + x)$

9	$y'' + 4y' + 4y = 0$	$y(0) = 1;$ $y'(0) = -1.$	**[0;1]**	$y = (1 + x)e^{-2x}$
10	$y'' - 3y' = e^{5x}$	$y(0) = 2{,}2;$ $y'(0) = 0{,}8.$	**[0;0,2]**	$y = 2 + 0{,}1(e^{3x} + e^{5x})$
11	$x^2 y'' - 2y = 0$	$y(1) = 0{,}83;$ $y'(1) = 0{,}66.$	**[1;2]**	$y = 0{,}5x^2 + \frac{1}{3x}$
12	$y'' - 5y' + 6y = e^x$	$y(0) = 0;$ $y'(0) = 0.$	**[0;0,2]**	$y = -e^{2x} + 0{,}5(e^{3x} + e^x)$
13	$y'' + y = 1 + e^x$	$y(0) = 2{,}5;$ $y'(0) = 1{,}5.$	**[0;1]**	$y = \cos x + \sin x + \frac{e^x}{2} + 1$
14	$x^2 y'' + 2{,}5y'x - y = 0$	$y(1) = 2;$ $y'(1) = 3{,}5.$	**[1;2]**	$y = 3\sqrt{x} - x^{-2}$
15	$y'' + y = x^2 - x + 2$	$y(0) = 1;$ $y'(0) = 0.$	**[0;1]**	$y = \cos x + \sin x + x^2 - x$

Контрольные вопросы

1. В чем заключается задача Коши для решения обыкновенных дифференциальных уравнений высших порядков?
2. В чем заключается метод Эйлера для решения обыкновенных дифференциальных уравнений высших порядков?
3. В чем отличие метода Эйлера для решения обыкновенных дифференциальных уравнений высших порядков от обыкновенных дифференциальных уравнений 1-го порядка?

Лабораторная работа №18. ОПТИМИЗАЦИЯ. ЗАДАЧА МИНИМИЗАЦИИ

Решить задачи оптимизации

методами:

1. Фибоначчи
2. «Золотого сечения»

Варианты заданий

№ варианта	Целевая функция Ф(x)	Точность ε	Интервал
1	$e^{-x} - x^4$	10^{-2}	[-3;2]
2	$e^{-x} - x^4$	10^{-2}	[-3;2]
3	$e^{-x} - x^4$	10^{-2}	[-3;2]
4	$x^5 - 4x$	10^{-3}	[0;2]
5	$x^5 - 4x$	10^{-3}	[0;2]
6	$x^5 - 4x$	10^{-3}	[0;2]
7	$\ln^2 x - \ln x$	10^{-3}	[0,5;2]
8	$\ln^2 x - \ln x$	10^{-2}	[0,5;2]
9	$\ln^2 x - \ln x$	10^{-2}	[0,5;2]
10	$e^{2x} - e^x$	10^{-5}	[-2;0]
11	$e^{2x} - e^x$	10^{-5}	[-2;0]
12	$e^{2x} - e^x$	10^{-5}	[-2;0]
13	$2x^3 - 3x + 1$	10^{-3}	[0;2]
14	$2x^3 - 3x + 1$	10^{-3}	[0;2]
15	$2x^3 - 3x + 1$	10^{-3}	[0;2]
16	$x^5 - x^3 + 1$	10^{-4}	[0;1,5]
17	$x^5 - x^3 + 1$	10^{-4}	[0;1,5]
18	$x^5 - x^3 + 1$	10^{-4}	[0;1,5]
19	$\sin x$	10^{-3}	[0;3,14]
20	$\sin x$	10^{-3}	[0;3,14]

Контрольные вопросы

1. Что такое числа Фибоначчи?
2. В чем заключается алгоритм Фибоначчи?
3. Что такое точка «золотого сечения»?
4. В чем заключается алгоритм метода золотого сечения?

Электронный учебно-методический комплекс

Раздел контроля знаний

ИНФОРМАТИКА

Перечень вопросов к экзамену и заданий к курсовой работе

Минск 2016

220

3.1. ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ

по дисциплине «Информатика»

для студентов 1 курса дневного отделения

(зимняя экзаменационная сессия)

1. Структура программы.
2. Вывод с использованием cout , printf
3. Ввод с помощью cin,
4. Вещественные типы
5. Вещественные константы
6. Тип bool
7. Символьные константы
8. Переменные целого типа
9. Монипулятор endl
10. Беззнаковые типы данных
11. Арифметические операции
12. Операции отношения
13. Цикл for
14. Цикл while
15. Условный оператор if
16. Вложенные ветвления if..else
17. Оператор switch
18. Оператор break
19. Условная операция
20. Логические операции (И,ИЛИ,НЕ)
21. Оператор continue
22. Оператор goto
23. Объявление функции
24. Определение функции
25. Рекурсия
26. Локальные и глобальные переменные
27. Фактические и формальные параметры
28. Вызов функции
29. Одномерные массивы (определение)

30. Элементы массива
31. Доступ к элементам массива
32. Ввод и вывод массива
33. Инициализация массива
34. Многомерные массивы
35. Возрастание и убывание одномерных массивов
36. Возрастание и убывание двумерных массивов
37. Файлы
38. Класс iostream
39. Класс istream
40. Класс ostream
41. Файл Fstream
42. Запись данных
43.Объект Outfile
44. Класс Ofstream
45. Чтение данных

Вопросы

по курсу «Информатика и интегрированные прикладные системы»

для студентов 1 курса дневного отделения

(весенняя экзаменационная сессия)

1. Численные методы решения нелинейных уравнений

1.1 Методы отделения корней

1.2 Оценка погрешности методов решения НУ

1.3 Метод бисекции (деления отрезка пополам)

1.4 Метод хорд

1.5 Метод простой итерации

1.6 Метод Ньютона

1.7 Метод секущих

2. Численные методы решения систем линейных алгебраических уравнений

2.1 Метод Гаусса

2.2 Метод итерации

2.3 Метод Зейделя

3. Интерполяция

3.1 Линейная интерполяция

3.2 Интерполяционный многочлен Лагранжа

3.3 Интерполяционный многочлен Ньютона

3.4 Интерполяция сплайнами

4. Аппроксимация

4.1 Метод наименьших квадратов

5. *Численное интегрирование*

5.1 Метод трапеций

5.2 Метод средних прямоугольников

5.3 Метод Симпсона

5.4 Метод Гаусса

6. *Численные методы решения диф. уравнений 1-го порядка*

6.1 Метод Эйлера

6.2 Модифицированный метод Эйлера

6.3 Метод Рунге-Кутта 4-го порядка

7. *Численные методы решения диф. уравнений высших порядков*

7.1 Метод Эйлера

8. *Оптимизация*

8.1 Метод общего поиска

8.2 Метод сканирования

8.3 Метод Фибоначчи

8.4 Метод «золотого сечения»

3.2. ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕМ КУРСОВОГО ПРОЕКТИРОВАНИЯ

1. Решить нелинейных уравнений методом деления отрезка пополам, метод простой итерации, методом хорд, методом Ньютона;
2. Решить систему линейных уравнений методом Гаусса и итерации;
3. Решить систему нелинейных уравнений методом Ньютона, Зейделя;
4. Вычислить определенный интеграл методом трапеций, прямоугольников, Симпсона
5. Решить обыкновенные дифференциальные уравнения;
6. Оптимизировать функцию методом золотого сечения, методом координатного спуска, метод градиентного спуска.

Электронный учебно-методический комплекс

Вспомогательный раздел

ИНФОРМАТИКА

Типовая учебная программа

Минск 2016

4. ТИПОВАЯ ПРОГРАММА

Министерство образования Республики Беларусь

Учебно-методическое объединение вузов Республики Беларусь по образованию в области энергетики и энергетического оборудования

УТВЕРЖДАЮ
Первый заместитель Министра образования Республики Беларусь

_____А. И. Жук
“ 03 ” 01. 2010 г.

Регистрационный № ТД-I 535___ /тип.

ИНФОРМАТИКА

Типовая учебная программа
для высших учебных заведений по специальностям:
1-43 01 04 Тепловые электрические станции;
1-53 01 04 Автоматизация и управление энергетическими процессами

СОГЛАСОВАНО

Председатель учебно-методического объединения вузов Республики Беларусь по образованию в области энергетики и энергетического оборудования
_____Ф.А. Романюк
_____

СОГЛАСОВАНО

Начальник Управления высшего и среднего специального образования Министерства образования Республики Беларусь
_____Ю.И. Миксюк
_____

Проректор по учебной и воспитательной работе Государственного учреждения образования «Республиканский институт высшей школы»
_____В.И. Шупляк
_____

Эксперт-нормоконтролер
_____
_____

Минск 2010

СОСТАВИТЕЛЬ:
Л.А. Тарасевич, доцент кафедры «Тепловые электрические станции» Белорусского национального технического университета, кандидат технических наук, доцент

РЕЦЕНЗЕНТЫ:
Кафедра энергосбережения, гидравлики и теплотехники Учреждения образования «Белорусский государственный технологический университет» (протокол № 7 от 28.01.2010 г.);

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:

Кафедрой "Тепловые электрические станции" Белорусского национального технического университета
(протокол № 7 от 18. 01.2010 г.)

Научно-методической комиссией Белорусского национального технического университета
(протокол №____ от _____ 2010 г.)

Учебно-методическим объединением вузов Республики Беларусь по образованию в области

энергетики и энергетического оборудования

(протокол №____ от _____ 2010 г.)

Ответственный за редакцию: Л.А. Тарасевич
Ответственный за выпуск: А.Г. Герасимова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Типовая учебная программа «Информатика» разработана на основе образовательного стандарта по специальностям 1-43 01 04 Тепловые электрические станции, 1-53 01 04 Автоматизация и управление энергетическими процессами. Целью изучения дисциплины является обучение студентов методам решения научно-технических и информационных задач, приобретение ими навыков работы на современных вычислительных средствах, изучение новых информационных технологий. Каждый инженер в своей профессиональной деятельности решает задачи математического моделирования, обработки массивов данных, представленных в виде таблиц или списков, и представления результатов обработки в виде отчетов, либо презентационной и деловой графики. В качестве базового учебного языка программирования выбран объектно-ориентированный язык C++, позволяющий осваивать классические приемы и современные технологии программирования. Полученные базовые навыки далее развиваются посредством обучения визуальному и объектно-ориентированному программированию с использованием языка Java.

Типовая программа «Информатика» написано в рамках изучения курса информатики студентами технических специальностей.

Целью изучения дисциплины является ознакомление студентов с современной технологией использования вычислительной техники в профессиональной деятельности. Каждый инженер в своей профессиональной деятельности решает задачи математического моделирования, обработки массивов данных, представленных в виде таблиц или списков, и представления результатов обработки в виде отчетов, либо презентационной и деловой графики.

Для практической подготовки студентов учебным процессом предусмотрено проведение лабораторных работ.

В результате освоения дисциплины «Информатика» студент должен:

знать:

- методы решения научно-технических и информационных задач;
- современные информационные технологии;

уметь:

- решать типовые программы математики и информатики;
- работать на современных вычислительных средствах;

- применять современные информационные технологии и методы реализации решения прикладных задач.

Методы (технологии) обучения

Основными методами (технологиями) обучения, отвечающими целям изучения дисциплины «Информатика и интегрированные прикладные системы», являются:

- элементы проблемного обучения, реализуемые на лекционных занятиях;
- элементы учебно-исследовательской деятельности, реализуемые при самостоятельной работе;
- проектные технологии, используемые при проектировании конкретного объекта, реализуемые при выполнении курсовой работы.

Организация самостоятельной работы студентов

При изучении дисциплины используются следующие формы самостоятельной работы:

- контролируемая самостоятельная работа в виде решения индивидуальных задач в аудиториях во время проведения лабораторных занятий под контролем преподавателя в соответствии с расписанием;
- подготовка курсового работы по индивидуальным заданиям.

Диагностика компетенций студента

Оценка уровня знаний студента при защите курсовой работы производится по десятибалльной шкале в соответствии с критериями, утвержденными Министерством образования Республики Беларусь.

Оценка промежуточных учебных достижений студента также осуществляется по десятибалльной шкале.

Для оценки достижений студента используется следующий диагностический инструментарий:

– защита выполненных на лабораторных занятиях индивидуальных заданий;

– защита курсовой работы;

– проведение текущих контрольных вопросов по отдельным темам;

– сдача экзамена.

Согласно типовому учебному плану на изучение дисциплины «Информатика и интегрированные прикладные системы» отведено всего на 494 учебных часов, в том числе -- 198 часов аудиторных занятий, из них лекции — 72 часа; лабораторные работы — 126 часов.

Примерный тематический план

Наименование темы	Лекции (часы)	Лабораторные занятия (часы)	Всего аудиторных часов
1	**2**	**3**	**4**
Раздел I. Основы программирования			
Введение	2		2
Тема 1. Простые типы данных	2	4	6
Тема 2. Функции	4	8	12
Тема 3. Структурированные типы данных	2	6	8
Тема 4. Файлы	4	10	14
Тема 5. Объектно-ориентированное программирование	4	8	12
Тема 6. Ввод-вывод.	4	8	12
Тема 7. Управляющие компоненты. Меню.	4	8	12
Тема 8. Работа с текстом	2	6	8
Тема 9. Графические возможности	2	4	6
Тема 10. Язык Java	6	10	16
Раздел II. Численные методы			
Тема 11. Решение нелинейных уравнений	6	6	12
Тема 12. Интерполяция	4	6	10
Тема 13. Численные методы решения системы линейных алгебраических уравнений	6	8	14
Тема 14. Численные методы решения систем нелинейных уравнений	4	6	10
Тема 15. Метод численного интегрирования	4	6	10
Тема 16. Аппроксимация	4	8	12
Тема 17. Численные методы решения обыкновенных дифференциальных уравнений	4	8	12
Тема 18. Безусловная оптимизация функций	4	6	10
ВСЕГО	72	126	198

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

РАЗДЕЛ I. ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

ВВЕДЕНИЕ

Основные понятия и обозначения: алгоритмы, языки, программы. Современное объектно-ориентированное программирование: C, C++, достоинства и недостатки. Влияние Internet на появление нового языка программирования Java, основные достоинства. Базовые термины Java.

Тема 1. ПРОСТЫЕ ТИПЫ ДАННЫХ

Объявление, представление в памяти, операции, определенные для различных типов.

Тема 2. ФУНКЦИИ

Объявление, определение, вызов. Способы передачи параметров, их типы. Перегрузка функций, указатели на функцию. Рекурсии. Правила использования стандартных функций, их классификация.

Тема 3. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

Статически и динамические массивы, строки, структуры, поля битов, объединения, перечисления, множества, списки и их виды (стеки, очереди, деревья). Объявление и инициализация данных структурированных типов, поиск, сортировка и другие алгоритмы работы с ними.

Тема 4. ФАЙЛЫ

Объявление, создание, чтение, корректировка. Прямой и последовательный доступ. Типы файлов. Стандартные файлы. Потоки.

Тема 5. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Абстракция данных. Три принципа объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм. Ввод программы, компиляция программ. Хранение объектов. Функции-шаблоны и классы-шаблоны. Контроль ошибок, обработка исключительных ситуаций. Потоки. Статические члены класса. Встроенные и вложенные классы. Библиотеки и пакеты классов.

Тема 6. ВВОД-ВЫВОД

Компоненты и функции, используемые для ввода-вывода.

Тема 7. УПРАВЛЯЮЩИЕ КОМПОНЕНТЫ, МЕНЮ

Разработка главного и контекстного меню. Разветвленные меню, екселераторы, горячие клавиши.

Тема 8. РАБОТА С ТЕКСТОМ

Стандартные процедуры и функции для работы со строками.

Тема 9. ГРАФИЧЕСКИЕ ВОЗМОЖНОСТИ

Технология вывода графики, рисование элементарных фигур, задание параметров изображений, вывод стандартных иллюстраций. Вывод диаграмм и графиков функций.

Тема 10. ЯЗЫК JAVA

Апплеты и приложения. Базовые типы и классы. Внутренние классы и интерфейсы. Обработка строк. Потоки данных. Графические средства. События. Элементы управления. Потоки выполнения. Сети. Использование пакетов языка Java.

РАЗДЕЛ II. ЧИСЛЕННЫЕ МЕТОДЫ

Тема 11. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Метод бисекции, метод простой итерации, метод хорд, метод Ньютона, метод секущих.

Тема 12. ИНТЕРПОЛЯЦИЯ

Табличный способ задания функций. Линейная интерполяция. Узлы интерполяции. Многочлен Лагранжа. Разделенные разности. Многочлен Ньютона. Интерполяция сплайнами.

Тема 13. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Точные и приближенные методы. Метод Гаусса. Метод итерации. Метод Зейделя. Сравнение методов.

Тема 14. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Метод Ньютона. Метод Зейделя.

Тема 15. МЕТОД ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

Метод трапеций. Метод прямоугольников. Формула Симпсона. Квадратурная формула Гаусса. Точностные оценки методов

Тема 16. АППРОКСИМАЦИЯ

Обработка экспериментальных данных. Метод наименьших квадратов.

Тема 17. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Типы задач для обыкновенных дифференциальных уравнений. Методы Эйлера, Рунге-Кутта второго и четвертого порядка, Адамса.

Тема 18. БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ ФУНКЦИЙ

Метод золотого сечения, метод координатного спуска, метод градиентного спуска.

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕМ ЛАБОРАТОРНЫХ РАБОТ

1. Линейные алгоритмы (программы). Арифметические операции. Выражения. Расчет по формулам.

2. Разветвляющие алгоритмы (программы). Логические выражения. Операции сравнения. Логические операции. Условные операторы if (if…else). Вложенные условные операторы. Оператор выбора switch.

3. Циклические алгоритмы (программы). Операторы цикла while, do…while, for. Вложенные операторы цикла.

4. Функции, возвращающие значение. Функции типа void. Фактические и формальные параметры, их связь. Передача параметров по значению. Перегрузка функций. Параметры по умолчанию. Передача параметров по ссылке.

5. Массивы. Одномерные и двумерные массивы.

6. Символы и строки.

7. Указатели.

8. Строки.

9. Структуры.

10. Классы и объекты.

11. Файлы. Текстовые и бинарные файлы. Создание файлов. Работа с файлами.

12. Работа с графикой. Графические компоненты. Использование канвы для рисования. Построение графиков функций на плоскости и в трехмерном пространстве.

13. Основные типы приложений на языке Java. Приложения и апплеты. Запуск приложений и апплетов.

14. Типы данных и операторов в Java. Базовые типы данных, классы-оболочки, массивы.

15. Классы в Java.

16. Строки в Java.

17. Потоки ввода-вывода в Java.

18. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Методы деления отрезка пополам. Метод простой итерации. Метод хорд. Метод Ньютона.

19. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГАБРАИЧЕСКИХ УРАВНЕНИЙ

Методы Гаусса.

Метод простой итерации и метод Зейделя.

20. ИНТЕРПОЛЯЦИЯ

Линейная интерполяция. Многочлены Лагранжа и Ньютона.

21. РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Метод Ньютона.

22. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Методы трапеций, прямоугольников (левых, правых, средних), Симпсона.

23. АППРОКСИМАЦИЯ ФУНКЦИЙ

Метод наименьших квадратов.

24. РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ1-ГО ПОРЯДКА

Метод Эйлера, модифицированные методы Эйлера (метод Эйлера-Коши, метод усредненных точек), метод Рунге-Кутты 4-го порядка.

25. РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ 2-ГО ПОРЯДКА

Метод Эйлера.

26. БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ ФУНКЦИЙ

Метод золотого сечения, метод координатного спуска, метод градиентного спуска

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕМ КУРСОВОГО ПРОЕКТИРОВАНИЯ

2. Решить нелинейных уравнений методом деления отрезка пополам, метод простой итерации, методом хорд, методом Ньютона;
2. Решить систему линейных уравнений методом Гаусса и итерации;
3. Решить систему нелинейных уравнений методом Ньютона, Зейделя;
4. Вычислить определенный интеграл методом трапеций, прямоугольников, Симпсона
5. Решить обыкновенные дифференциальные уравнения;

6. Оптимизировать функцию методом золотого сечения, методом координатного спуска, метод градиентного спуска.

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ КОМПЬЮТЕРНЫХ ПРОГРАММ

1. Word 2003;
2. Excel 2003;
3. Mathcad 2000;
4. Mathematica 5.2;
5. C++ Builder.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Павловская, Т. А. C/C++. Программирование на языке высокого уровня : учебник для студ. вузов, обуч. по напр. "Информатика и вычислительная техника" / Т. А. Павловская. - Санкт-Петербург : Питер, 2006. - 460с.
2. Юров В. Assembler : [Учебник] / В.Юров. - СПб. и др. : Питер, 2000. - 623с.
3. Зубков С.В. Assembler для DOS, Windows и UNIX для программистов. - Издательство: Питер, 2004. –608 с.
4. Финогенов К.Г. Win32.Основы программирования. –Издательство ДИАЛОГ-МИФИ, 2002. – 416 с.
5. Хорстманн Кей С., Корнелл Гари. Java 2. Библиотека профессионала, том 1. Основы, 7-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2007. – 896 с.: ил. – Парал. тит. англ.
6. Гудрич М.Т. Структуры данных и алгоритмы в Java / М.Т. Гудрич, Р. Тамас-сия; Пер. с англ. А.М. Чернухо. – Мн.: Новое знание, 2003. – 671 с.: ил. / NY, John Willey & Sons, 2001.
7. Джамса К. Библиотека программиста Java. — Мн.: Попурри, 1996. — 638 с.
8. Симкин С. Программирование на Java. Путеводитель.: пер. с англ. / Стив Симкин, Нейл Барлет, Алекс Лесли. — Киев: DIASOFT, 1996. — 736.
9. Демидович, Б.П. Численные методы анализа/по ред. Б.П. Демидовича, И.А. Марон, Э.З. Шувалова. - Москва, 1967.-368 с.

10. Демидович, Б.П. Основы вычислительной математики/ Б.П. Демидович, И.А. Марон. - Москва, 1966.-664 с.
11. А.В. Пантелеев,Т.А. Методы оптимизации в примерах и задачах.: учебное пособие/ А.В. Пантелеев,Т.А. Летова. – 2-е изд., исправл.-Москва.: Высш.школа.,2005.-544 с.
12. Березин, И.С. Методы вычислений/ И.С. Березин, Н.П. Жидков. – М., 1962.-464 с.
13. Самарский,А.А. Численные методы/ А.А. Самарский, А.В. Гулин.: учебное пособие для вузов.- М: Наука.,1989.-432 с.
14. Дегтярев, Ю.И. Методы оптимизации.- М., Советское радио, 1980.-560 с.
15. Марчук, Г.И. Методы вычислительной математики. - М., Наука, 1989.-460с.
16. Осипенко, К.Ю. Аппроксимация функций многочленами и численное дифференцирование: методические указания по курсу “Численные методы”. - М., МАТИ, 1994.-370 с.

17.Осипенко, К.Ю. Квадратурные формулы: методические указания по курсу “Численные методы”.- М., МГАТУ, 1995.-65 с.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1.Язык программирования C = Programming Language C / Брайан Керниган, Деннис Ритчи ; [пер. с англ. и ред. В. Л. Бродового]. - 2-е изд., перераб. и доп. - Москва ; Санкт-Петербург ; Киев : Вильямс, 2006. - 290 с.

2.C/C++ для студента / А. П. Побегайло. - Санкт-Петербург : БХВ-Петербург, 2006. - 526 с. Объектно-ориентированное программирование в C++ = Object-Oriented Programming in C++ / Р. Лафоре; [пер. с англ. А. Кузнецова, М. Назарова, В. Шрага]. - 4-е изд. - Санкт-Петербург [и др.] : Питер, 2005. - 924с.

3.В.Ю.Пирогов. Ассемблер. Учебный курс. - Издательство: BHV,2003 г.

4.Ганеев Р. Проектирование интерфейса пользователя средствами Win32 API Горячая Линия – Телеком, 2006. – 358 с.

5.Программирование на Microsoft Visual C++6.0 = Programming Microsoft Visual C++6.0 : Пер. с англ. / Дэвид Дж. Круглински, Скотт Уингоу, Джордж Шеферд. - 5-е изд. - М. ; СПб. : Русская редакция : Питер, 2003. - 819с.

6.Грегори Кэйт. Использование Visual C++.NET: Специальное издание: Пер. с англ. / К.Грегори; Под ред.Г.П.Петриковца. - М. и др. : Изд. дом "Вильямс", 2003. - 784с.

7.Патрик Ноутон, Герберт Шилдт. Java 2 в подлиннике. - Издательство: BHV. Серия: В подлиннике, 2005. -1072 с.