**Программирование (2 семестр)**

На данной странице представлен перечень лабораторных работ. Чтобы подробнее ознакомиться с интересующей Вас информацией, нажмите Ctrl и щелкните по необходимой ссылке.

**Теоретические сведения:**

[№1. Знакомство с языком С. Выполнение программ с линейной вычислительной структурой](#_Лабораторная_работа_№1)

### [№2. Операторы языка Си](#_Лабораторная_работа_№2)

[№3. Операторы цикла](#_Лабораторная_работа_№3)

[№4. Примитивная](#_Лабораторная_работа_№4) графика

[№5. Обработка одномерных массивов](#_Теоретические_сведения_об)

**Задания к лабораторным работам:**

[№1. Знакомство с языком С. Выполнение программ с линейной вычислительной структурой](#_"Знакомство_с_языком)

### [№2. Операторы языка Си](#_"Операторы_языка_Си")

[№3. Операторы цикла](#_Задания_к_лабораторной)

[№4. Примитивная](#_Лабораторная_работа_№4) графика

[№5. Обработка одномерных массивов](#_Задание_к_лабораторной)

### **Лабораторная работа №1**

### **«Знакомство с языком С. Выполнение программ с линейной вычислительной структурой»**

**Цель:** Знакомство со средой программирования, создание, отладка и выполнение простой программы, содержащей ввод/вывод информации и простейшие вычисления.

### **Краткие теоретические сведения**

Язык С был создан в 1972 г. сотрудником фирмы BellLaboratories в США Денисом Ритчи.

Первым системным программным продуктом, разработанным с помощью С, стала операционная система UNIX.

Стандарт С был утвержден в 1983 г. Американским национальным институтом стандартов (ANSI) и получил название ANSI С.

В начале 1980-х гг. в той же фирме BellLaboratories ее сотрудником Бьерном Строуструпом было разработано расширение языка С, предназначенное для объектно-ориентированного программирования. По сути дела, был создан новый язык, первоначально названный «С с классами», а позднее (в 1983 г.) получивший название С++ (Си-плюс-плюс). Язык С++ принято считать языком объектно-ориентированного программирования. Однако этот язык как подмножество включает в себя С и по-прежнему сохраняет свойства языка для системного программирования. Все существующие версии трансляторов для С++ поддерживают стандарт ANSI С.

Язык С++ поддерживает как процедурную, так и объектно-ориентированную парадигмы программирования.

### **Структура С-программ**

Программы на языке С имеют следующую структуру

|  |
| --- |
| #директивы препроцессора  ...  функция f1(список параметров) //функция, которая будет использована в программе  {тело функции;}  заголовок программы () //функция, с которой начинается выполнение программы  {тело программы;  завершение программы;  } |

**Директива препроцессора**

Препроцессором называется первая фаза компилятора. Инструкции препроцессора называются директивами. Они должны начинаться с символа #, перед которым в строке могут находиться только пробельные символы.

Существует три основные директивы препроцессора:

1. *#include* – директива для подключения файлов.

Эта директива имеет две формы:

*#include " имя\_файла "* - поиск файла осуществляется в соответствии с

заданным маршрутом, а при его отсутствие – в текущем каталоге.

*#include <имя\_файла>* - поиск файла производится в стандартных

директориях операционной системы, задаваемых в файле AUTOEXEC

командой PATH.

2) *#define*. Используется для выполнения простого присваивания значения в программу.

Эта директива имеет следующую форму:

*# define идентификатор подставляемое\_значение*

3) *#undef* - отмена простого присваивания.

Форма представления:

*#undef идентификатор;*

**Заголовок и завершение программы**

В каждой программе на языке С используется главная функция *main()*. Описание функций состоит из заголовка и тела.

Функция может возвращать значение в программу с помощью оператора return. Этот оператор также означает выход из главной функции. Если же указанный оператор отсутствует, то функция автоматически возвращает значение типа *void* (пустой).

Пример 1:

|  |
| --- |
| void main() // заголовок программы  {int b;  b=2;  cout<<”Число b равно”<<b;  getchar (); // завершение программы  } |

После того как компьютер заканчивает выполнений инструкций, заданных в программе, программа завершается, и компьютер возвращается в исходное состояние. Возврат в исходную среду в случаях, когда функция не возвращает значения, как правило, осуществляется автоматически. Компиляторы языка С и С++ требуют явного указания возврата. Для них вводится инструкция *return 0;*, которую помещают перед фигурной скобкой, завершающей тело функции *main()*. Если функция возвращает значение, то тело функции должно содержать оператор *return.* Форма записи:

*return возвращаемое\_значение;*

Пример 2:

|  |
| --- |
| int main() // заголовок программы  {int b;  b=2;  cout<<”Число b равно”<<b;  return 0; // завершение программы  } |

### **Константы и переменные**

Константа - это значение, которое не может быть изменено. Синтаксис языка определяет 5 типов констант: символы, константы перечисляемого типа, вещественные числа, целые числа, нулевой указатель (NULL).

Переменные можно изменять. При задании значения переменной в соответствующую ей область памяти помещается код этого значения. Доступ к значению возможен через имя переменной, а доступ к участку памяти - по его адресу.

Каждая переменная перед использованием в программе должна быть определена, т. е. ей должна быть выделена память.

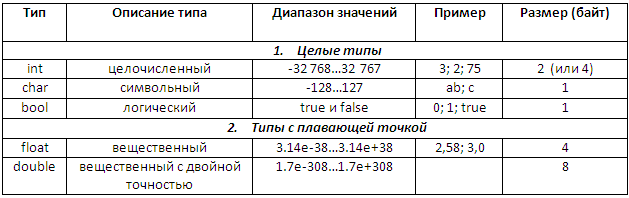
Размер участка памяти, выделяемой для переменной и интерпретация содержимого зависят от типа, указанного в определении переменной.

Форма определения переменной:

*тип список\_имен\_переменных;*

Все типы языка C++ можно разделить на основные и составные.В языке C++ определено шесть основных типов данных для представления целых, вещественных, символьных и логических величин. На основе этих типов программист может вводить описание составных типов. К ним относятся массивы, перечисления, функции, структуры, ссылки, указатели, объединения и классы.

Таблица 1 – Основные типы данных



Существует четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений основных типов:

- short – короткий;

- long – длинный;

- signed – знаковый;

- unsigned – беззнаковый.

Минимальные и максимальные допустимые значения для целых типов

зависят от реализации и приведены в заголовочном файле limits.h, характеристики вещественных типов — в файле float.h.

### **Операции языка Си**

Существуют унарные, бинарные операции и операции присваивания.

Таблица 2 – Операции языка С

|  |  |  |
| --- | --- | --- |
| Знак операции | Наименование операции | Группа операций |
| 1. Унарные операции | | |
| - | Арифметическое  отрицание | Отрицание |
| ~ | Дополнение, поразрядное логическое отрицание | Дополнение |
| ! | Логическое отрицание | Логическая |
| \* | Обращение по адресу | Адресная |
| & | Определение по адресу | Адресная |
| + | Унарный плюс | Арифметическая |
| sizeof() | Размер | Работа с памятью |
| (тип) x | Приведение типа  операнда x | Работа с памятью |
| 1. Бинарные операции | | |
| \*, / | Умножение, деление | Мультипликативные |
| % | Остаток от деления | Мультипликативные |
| +, - | Сложение, вычитание | Аддитивные |
| << | Побитовый сдвиг влево | Поразрядные |
| >> | Побитовый сдвиг вправо | Поразрядные |
| > | Больше | Отношения |
| < | Меньше | Отношения |
| <= | Не больше | Отношения |
| >= | Не меньше | Отношения |
| == | Равно | Отношения |
| != | Не равно | Отношения |
| & | Поразрядное логическое И | Поразрядные |
| | | Поразрядное логическое ИЛИ | Поразрядные |
| ^ | Поразрядное логическое ЛИБО | Поразрядные |
| && | Логическое И (конъюнкция) | Логические |
| || | Логическое ИЛИ (дизъюнкция) | Логические |
| , | Следование |  |
| () | Приоритет |  |
| () | Функция |  |
| [] | Массив |  |
| 1. Операции присваивания | | |
| ++ | Инкремент | Увеличение на 1 |
| -- | Декремент | Уменьшение на 1 |
| = | Простое присваивание |  |
| +=, -=, \*=, /=, %=,  <<=, >>=, &=, |=, ^= | Совмещенное присваивание | Совмещенные |
| . , -> | Селектор | Работа с элементами структур |
| 1. Тернерная (условная) операция | | |
| ? : | Тернерная | Условная |

При выполнении вычисления значений выражений, необходимо следить за следованием той или иной операции.

Таблица 3 – Приоритеты операций языка С

|  |  |
| --- | --- |
| Ранг | Операции |
| 1 | ( ), [ ], ->, . |
| 2 | !, ~, ++, --, &, \*, (тип), sizeof( ) |
| 3 | \*, /, % |
| 4 | +, - |
| 5 | <<, >> |
| 6 | <, >, <=, >= |
| 7 | ==, != |
| 8 | & |
| 9 | ^ |
| 10 | | |
| 11 | && |
| 12 | || |
| 13 | ? : |
| 14 | +=, \*=, /=, %=, -=, &=, ^=, |=, <<= , >>= |
| 15 | , |

### **Ввод и вывод данных**

В языке C++ нет встроенных средств ввода/вывода — он осуществляется с помощью функций, типов и объектов, содержащихся в стандартных библиотеках. Используется два способа: функции, унаследованные из языка С, и объекты C++.

**Форматный ввод и вывод данных**

Объекты, унаследованные из языка С:

- *printf (“форматная строка”, список аргументов)* – функция вывода

Информации;

- *scanf (“спецификатор формата”, список аргумента)* – функция ввода данных.

Для использования данных объектов необходимо подключить заголовочный файл stdio.h (также рекомендуется подключить файл conio.h – для старых версий С).

Например:

|  |
| --- |
| printf( “\nВведите целое число x и вещественное число y”);  scanf(“%d %f”,&x,%y);  printf(“\nВы ввели числа: %d и %f”,x,y); |

Каждому аргументу соответствует своя спецификация преобразования:

%d - десятичное целое число;

%f - число с плавающей точкой;

%c - символ;

%s - строка.

\n - управляющий символ новая строка.

**Потоковый ввод и вывод данных**

Объекты С++:

- *cout << “строка для вывода”;*– функция вывода информации на экран;

*- cin >>список аргументов;* – функция ввода данных с клавиатуры.

Заголовочный файл <iostream.h> содержит описание набора классов для управления вводом/выводом. В нем определены стандартные объекты-потоки *сin*, *cout*, а также операции помещения в поток << и чтения из потока >>.

Например:

|  |
| --- |
| cout<<“\nВведите целое число x и вещественное число y”;  cin>>x>>y;  cout<<“\nВы ввели числа: ”<<x<<“ и ”<<y; |

Можно пользоваться этими двумя способами, но в одной программе смешивать их не рекомендуется.

### **Основные математические функции**

Рассмотрим основные математические функции стандартной библиотеки (заголовочный файл <math.h>)

Таблица 4 – Основные математические функции



### **Вспомогательные символы**

Для удобства редактирования и написания текста программы, можно воспользоваться комментариями. Комментарий, начинающийся с символа // - однострочный, им можно закомментировать одну строку в программе.

Для многострочного комментирования используют следующую форму:

*/\* Здесь\_написан\_комментарий\*/*

Для облегчения многих действий в программе, можно воспользоваться следующими управляющими символами:

Таблица 5 – Управляющие символы С

|  |  |
| --- | --- |
| Символ | Назначение |
| \t | Горизонтальная табуляция |
| \v | Вертикальная табуляция |
| \n | Символ новой строки |
| endl | Переход на новую строку (аналогично \n) |

### **Вопросы на защиту**

1. Структура простейшей программы на Си. Синтаксис, примеры.
2. Ввод и вывод текста на экран. Способы, синтаксис, примеры.
3. Директивы препроцессора. Формы записи, синтаксис.
4. Переменная. Типы данных. Синтаксис объявления переменных.
5. Типы данных. Основные и составные. Спецификаторы, диапазон значений, размер в байтах.
6. Потоковый ввод и вывод данных, синтаксис.
7. Арифметические выражения. Приоритет операций арифметических выражений. Инкремент и декремент.

**Задания к лабораторной работе №1**

### **«Знакомство с языком С. Выполнение программ с линейной вычислительной структурой»**

**Задание 1.** Вычислить значение выражения при различных типах данных (int, float,double). Вычисления следует выполнять с использованием промежуточных переменных. Сравнить и объяснить полученные результаты.

**Варианты**

1. ****
2. ****
3. ****
4. ****
5. ****
6. ****
7. ****
8. ****
9. ****
10. ****
11. ****
12. ****
13. ****
14. ****
15. ****
16. ****
17. ****
18. ****
19. ****
20. ****
21. ****
22. ****
23. ****
24. ****
25. ****
26. ****
27. ****
28. ****
29. ****

****

**Задание 2**. Написать программу вычисления значений выражений. Все необходимые для вычисления данные вводятся с клавиатуры. Объяснить полученные результаты. Сверить полученные результаты с ручным подсчетом.

**Варианты**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Задание | № | Задание | № | Задание |
| 1 | m-=n+5, f<<2 | 11 | m-=n-----5, f>>3 | 21 | m%=--n+5, f>>2 |
| 2 | a-=b+++++c, a<<3 | 12 | a%=b+++++c, a>>3 | 22 | a+=--b+c, --a<<3 |
| 3 | s+=r-----t, t<<2 | 13 | s+=++r---t, t>>2 | 23 | s^=--r-t, 145>>t |
| 4 | a=b^c, c-- | 14 | a-=b%c, c++ | 24 | a-=++b^c, --c |
| 5 | x+=25%d, x>>2 | 15 | x-=35%d, x<<2 | 25 | x%=125%y, 145>>y |
| 6 | c-=d++-++e, e<<3 | 16 | c-=d++-++e, e<<3 | 26 | c+=--d++-e, 154<<d |
| 7 | d-=a+c++d, d>>2 | 17 | d-=--a+c++, d<<2 | 27 | d-=--a---c, 146>>c |
| 8 | s-=r+++++t, t<<2 | 18 | s=-r++t, t>>2 | 28 | s+=--r+t, 167<<r |
| 9 | a=b%c, b^c | 19 | a-=--b%c, b-- | 29 | a=--b%c, 178>>b |
| 10 | a-=b+r, b<<12 | 20 | a-=b---r, 132>>b | 30 | a%=--b+++r, 14<<b |

**Задание 3.** Решить задачу в соответствие с вариантом. Вычисления следует выполнять с использованием промежуточных переменных. Сравнить и объяснить полученные результаты.

**Варианты**

1. Определить периметр многоугольника, заданного координатами вершин М1, М2, М3,М4.
2. По известным линейным размерам прямой правильной шестиугольной призмы (длины сторон основания и высота призмы) определить ее объем и площадь полной поверхности.
3. Треугольник задан периметром и радиусом вписанной окружности. Найти площадь ортогональной проекции треугольника на плоскость, расположенную под углом α к плоскости треугольника.
4. По известным длинам двух сторон треугольника и углу между ними найти длины его высот.
5. Определить расстояние, пройденное физическим телом за время t, если тело движется равноускоренно с ускорением а, имеет в начальный момент времени скорость v0 и положение х0
6. Определить скорость движения физической точки на ободе колеса, если известны радиус и частота вращения колеса.

7. По известным радиусу основания конуса и углу между образующей и плоскостью основания найти объем и площадь боковой поверхности конуса.

8. Найти длину дуги и площадь кругового сегмента, соответствующего дуге, если известны диаметр окружности и радианная мера выделяемого угла.

9. Найти площадь ромба, если известны длина большей диагонали и градусная мера тупого угла.

10. Найти площадь окружности, вписанной в равнобочную трапецию с основаниями a и b и боковой стороной с.

11. Найти площадь диагонального сечения прямоугольной призмы по известным длинам сторон основания и высоте призмы.



12. Вычислить значение функции

13. Найти отношение площадей кругов, описанного около треугольника со сторонами a,b,c и вписанного в него.

14. Найти меньшее основание прямоугольной трапеции с площадью S и периметром Р.

15. Найти площадь правильного треугольника по известному радиусу описанной окружности.

16. Вычислить производную функции  в точке α0.

17. Вычислить первообразную функции  на отрезке [a,b].

18. Найти объем правильного тетраэдра со стороной a.

19. Вычислить значение функции  в точке z0.

20. Найти площадь ромба, если известны длина большей диагонали и градусная мера острого угла.

21. По известным линейным размерам правильной треугольной пирамиды (длины сторон основания и боковое ребро пирамиды) определить ее объем и площадь полной поверхности.

22. Найти угол наклона бокового ребра длины b правильной треугольной пирамиды к плоскости основания, если сторона основания равна a.

23. Вычислить производную функции в точке а0.

24. Вычислить первообразную функции  на отрезке [s,t].

25. Найти угол наклона образующей длины b к плоскости основания конуса, если диаметр основания равен a.

26. Вычислить значение функции  в точке х0.

27.Найти периметр и площадь правильного шестиугольника по известному радиусу описанной окружности.

[Возврат к начальной странице](#_top)

### **Лабораторная работа №2**

### **«Операторы языка Си»**

### **Краткие теоретические сведения**

Операторы управления работой программы называют управляющими конструкциями программы. К ним относят:

- составные операторы;

- операторы выбора;

- операторы циклов;

- операторы перехода.

Рассмотрим основные операторы и запись их на языке Си.

Таблица 1 – Операторы языка Си

|  |  |  |  |
| --- | --- | --- | --- |
| Язык блок-схем | Школьный  алгоритмический язык | | С, С++ |
| 1. *Составной оператор*   К составным операторам относят собственно составные операторы и блоки. В обоих случаях это последовательность операторов, заключенная в фигурные скобки. Блок отличается от составного оператора наличием определений в теле блока. Составной оператор выполняется как единое целое. | | | |
|  | | действие1 действие2 … | {//пример блока  int a=2;  S=sqrt(a)+1/(a\*a-2)-exp(10\*ln(a));  P=sin(S)+tan(a\*a);  } |
| 1. *Оператор выбора*   Операторы выбора - это условный оператор и переключатель. Условный оператор имеет полную и сокращенную форму. | | | |
|  | | **если** условие  **то** действия | if (условие)  действие1;  Пример:  if ((a<=-1) || (a>=1))  S=sqrt(a\*a-1); |
|  | | **если** условие  **то** действи1  иначе действие2 | if (условие)  действие1;  else действие2;  Пример:  if ((a>2) && (a<6))  S=exp(7\*ln(a))+2;  else S=exp(8\*ln(a))-3; |
| 1. *Операторы циклов*   А) *Цикл с предусловием*. Если условие истинно, то тело цикла выполняется до тех пор пока условие истинно (т.е. не станет ложным).  Б) *Цикл с постусловием*. Тело цикла выполняется до тех пор, пока условие истинно (т.е. не станет ложным).  В) *Цикл с параметром*. Выполняется до тех пор, пока условие истинно. | | | |
| А) | **пока** условие  действие | | while ( *условие* )  {  *тело цикла;*  } |
| Б) | **выполняй** действие  **пока** условие | | do *оператор*  while ( *условие* )  {  *тело цикла;*  }; |
| В) | **пока** условие верно  **выполняй** действие | | for (*нач знач;условие; действие*)  {  *тело цикла;*  } |

Операторы перехода выполняют безусловную передачу управления.

1) break – оператор прерывания цикла.

*while(условие)*

*{ операторы;*

*if (условие) break;*

*операторы; }*

Т. е. оператор break целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла.

2) continue – переход к следующей итерации цикла. Он используется, когда тело цикла содержит ветвления.

*while(условие)*

*{ операторы;*

*if (условие) continue;*

*операторы; }*

**Вопросы на защиту:**

1. Битовые операции. Приоритет битовых операций.
2. Операции отношений. Приоритет операций отношений. Таблица значений.
3. Логические операции и отношения. Таблица значений.
4. Операция определения размера данных. Формы записи, синтаксис.
5. Разделы программы на Си. Библиотеки языка программирования Си.
6. Условный оператор. Смысл, синтаксис, блок схема.
7. Условная операция. Смысл, синтаксис, блок схема.
8. Оператор выбора вариантов. Смысл, синтаксис, блок схема.

**Задание к лабораторной работе №2**

### **«Операторы языка Си»**

**Постановка задачи:** Написать 3 программы (с использованием условного оператора, условной операции, оператора switch) определяющие расположение точки А (х,у) относительно заштрихованной области**.**

Варианты

|  |  |  |
| --- | --- | --- |
| **1**  -4  3  5  8 | **2**  7  -7  14 | **3**  -10  -5  -7  6 |
| **4**  3  -4  -3  5 | **5**  5  -3  -3  3 | **6**  3  4  -5  -3  8 |
| **7**  9  2,5  -2,5 | **8**  3  3  7  6  11 | **9**  1  5  4  7  2  5 |
| **10**  3  -6  -3  6 | **11**  6  -4  4  -4  6 | **12**  1  -4  -1  5  3 |
| **13**  6  3  7 | **14**  4  -12  -8  8 | **15**  -2  -2  -7  5 |
| **16**  -4  -3  -3  2  2 | **17**  5  -5  -4  -1 | **18**  5  -4  4  8  1  -8 |
| **19**  1  -2  4 | **20**  4  -4  8  2 | **21**  2  -3  -6  3 |
| **22**  3  3  -1  -3  -3 | **23**  5  -5  3 | **24**  5  9  -3  1 |
| **25**  6  2  -6  -2  -5  1  -1 | **26**  3  -2  -6  3 | **27**  3  -1  4  4  -2 |
| **28**  3  3  6 | **29**  3  -4  7  -8 | **30**  3  -1  -9  6 |

[Возврат к начальной странице](#_top)

### **Лабораторная работа №3**

**«Операторы цикла»**

**Теоретические сведения о циклах**

## Если мы знаем точное количество действий (итераций) цикла, то можем использовать цикл for. Синтаксис его выглядит примерно так:

**for** (действие до начала цикла;

условие продолжения цикла;

действия в конце каждой итерации цикла)

{

инструкция цикла;

инструкция цикла 2;

инструкция цикла N;

}

Итерацией цикла называется один проход этого цикла

Существует частный случай этой записи, который мы сегодня и разберем:

**for** (счетчик = значение; счетчик < значение; шаг цикла)

{

тело цикла;

}

**Счетчик цикла** — это переменная, в которой хранится количество проходов данного цикла.

### Описание синтаксиса

1)Сначала присваивается первоначальное значение счетчику, после чего ставится точка с запятой.

2)Затем задается конечное значение счетчика цикла. После того, как значение счетчика достигнет указанного предела, цикл завершится. Снова ставим точку с запятой.

3)Задаем шаг цикла. **Шаг цикла** — это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

### Пример кода

Напишем программу, которая будет считать сумму всех чисел от 1 до 1000.



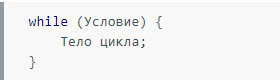
Если мы скомпилируем этот код и запустим программу, то она покажет нам ответ: 500500. Это и есть сумма всех целых чисел от 1 до 1000. Если считать это вручную, понадобится очень много времени и сил. Цикл выполнил всю рутинную работу за нас.

Заметьте, что конечное значение счетчика мы задали нестрогим неравенством ( <= — меньше либо равно), поскольку, если бы мы поставили знак меньше, то цикл произвел бы 999 итераций, т.е. на одну меньше, чем требуется. Это довольно важный момент, т.к. здесь новички часто допускают ошибки, особенно при работе с [массивами](https://code-live.ru/post/cpp-arrays/) Значение шага цикла равное единице. **i++** — это тоже самое, что и **i = i + 1**.

В теле цикла, при каждом проходе программа увеличивает значение переменной sum на i. Еще один очень важный момент — в начале программы присвоили переменной sum значение нуля.

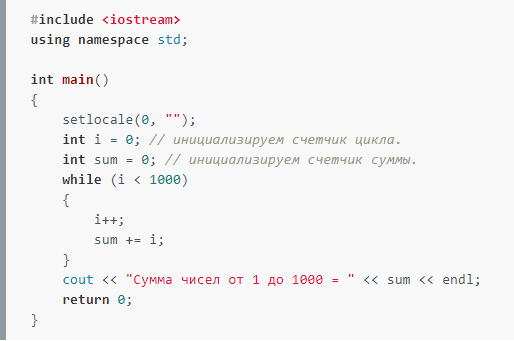
## Цикл while

Когда мы не знаем, сколько итераций должен произвести цикл, нам понадобится цикл **while** или **do...while**. Синтаксис цикла **while** в C++ выглядит следующим образом.

****

Данный цикл будет выполняться, пока условие, указанное в круглых скобках является истиной. Решим ту же задачу с помощью цикла **while**. Хотя здесь мы точно знаем, сколько итераций должен выполнить цикл, очень часто бывают ситуации, когда это значение неизвестно.

Ниже приведен исходный код программы, считающей сумму всех целых чисел от 1 до 1000.



После компиляции программа выдаст результат, аналогичный результату работы предыдущей программы. Но поясним несколько важных моментов. Мы задали строгое неравенство в условии цикла и инициализировал счетчик i нулем, так как в цикле **while** происходит на одну итерацию больше, потому он будет выполняться, до тех пор, пока значение счетчика перестает удовлетворять условию, но данная итерация все равно выполнится. Если бы мы поставили нестрогое неравенство, то цикл бы закончился, когда переменная i стала бы равна 1001 и выполнилось бы на одну итерацию больше.

Теперь давайте рассмотрим по порядку исходный код нашей программы. Сначала мы инициализируем счетчик цикла и переменную, хранящую сумму чисел.

В данном случае мы обязательно должны присвоить счетчику цикла какое-либо значение, т.к. в предыдущей программе мы это значение присваивали внутри цикла **for**, здесь же, если мы не инициализируем счетчик цикла, то в него попадет «мусор» и компилятор в лучшем случае выдаст нам ошибку

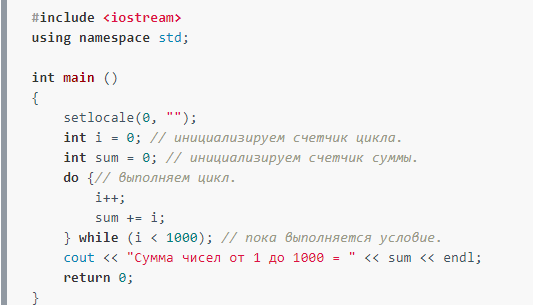
Затем мы описываем условие цикла — **«пока переменная i меньше 1000 — выполняй цикл»**. При каждой итерации цикла значение переменной-счетчика i увеличивается на единицу внутри цикла.

Когда выполнится 1000 итераций цикла, счетчик станет равным 999 и следующая итерация уже не выполнится, поскольку 1000 не меньше 1000. Выражение sum += i является укороченной записью sum = sum + i.

После окончания выполнения цикла, выводим сообщение с ответом.

## Цикл do while

Цикл **do while** очень похож на цикл **while**. Единственное их различие в том, что при выполнении цикла **do while** один проход цикла будет выполнен независимо от условия. Решение задачи на поиск суммы чисел от 1 до 1000, с применением цикла *do while*.



Принципиального отличия нет, но если присвоить переменной i значение, большее, чем 1000, то цикл все равно выполнит хотя бы один проход.

Попрактикуйтесь, поэкспериментируйте над собственными примерами задач. Циклы — очень важная вещь, поэтому им стоит уделить побольше внимания.

**Вопросы на защиту:**

1. Глобальные и локальные переменные. 4 группы операторов языка Си.
2. Форматный ввод и вывод, синтаксис. Форматы для вывода данных.
3. Операторы break, continue. Смысл, синтаксис.
4. Цикл с параметром. Смысл, синтаксис, блок схема.
5. Цикл с предусловием. Смысл, синтаксис, блок схема.
6. Цикл с постусловием. Смысл, синтаксис, блок схема.
7. Методы отладки программ.

### **Задание к лабораторной работе №3**

**«Операторы цикла»**

**Постановка задачи:** Написать 3 программы с использованием трех видов циклов (цикл с предусловием, с постусловием, с параметром) табулирующих функцию в заданном диапазоне.

Варианты

1. , , , , 
2. , , , , 
3. , , , , 
4. , , , 
5. , , 
6. , , , , 
7. , , , , 
8. , , , 
9. , , , 
10. , , , , 
11. , , 
12. , , 
13. , , 
14. , , 
15. , , 
16. , , 
17. , , 
18. , , , , 
19. , , 
20. , , , , 
21. , , 
22. , , 
23. , , 
24. , , 
25. , , 
26. , , , , 
27. , , , , 
28. , ,, , 
29. , , 
30. , , , , 

[Возврат к начальной странице](#_top)

# Лабораторная работа №4.

**Примитивная графика. Исследование функций графических библиотек**

Графическая библиотека языков фирмы Borland называется BGI. Она содержит драйверы, имеющие расширение \*.bgi. Имя драйвера соответствует имени адаптера: в каталоге BGI имеются файлы cga.bgi, egavga.bgi, herc.bgi и др. Графическая библиотека (заголовочный файл) подключается директивой компилятора С (С++) **#include <graphics.h>**. В этом файле содержится определение графического пакета - описаны режимы различных адаптеров (разрешающая способность в пикселах, цветовая палитра и т.д.), стили линий и шрифтов и др.

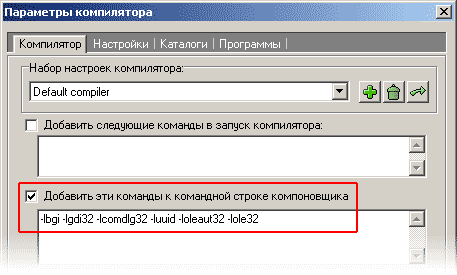
В графическом режиме экран представляется совокупностью пикселов. Количество точек (разрешающая способность) зависит от типа видеоадаптера и установленного для него режима.

В текстовом режиме верхний левый угол экрана имеет позицию (1,1), координата Х растет вправо, координата Y растет вниз. В графическом режиме верхний левый угол имеет координаты (0,0), координаты Х и Y направлены аналогично.

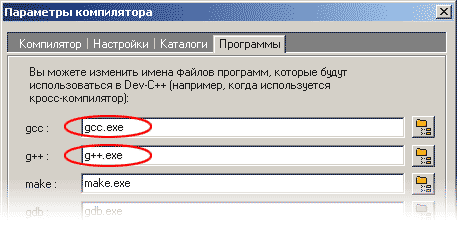
**Для того, чтобы подключить модуль для работы с графикой, сделайте следующее:**

1. Скачайте архив (devcppgr), содержащий файлы graphics.h (заголовочный файл) и libbgi.a (библиотека). Размер архива 53 Кб*.*
2. Скопируйте файл graphics.h в папку C:\Dev-Cpp\include.
3. Скопируйте файл libbgi.a в папку C:\Dev-Cpp\lib.
4. Запустите оболочку Dev-C++ и войдите в меню Сервис-Параметры компилятора.
5. Перейдите на вкладку Компилятор, включите флажок ***Добавить эти команды к командной строке компоновщика*** и добавьте в окно под этим флажком строчку:

*-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32*



1. Чтобы выводить русский текст в графическом режиме, в окне *Сервис-Параметры компилятора* на вкладке *Программы* нужно установить программы по умолчанию **gcc.exe** и **g++.exe**:



**Функции graphics.h:**

initgraph( &grdriver, &grmode, “path”); – функция инициализации графического режима, где &grdriver, &grmode, “path” – параметры загружаемого режима;

grdriverи grmode– переменные типа integer;

“path” – путь к загружаемому драйверу графического режима.

Функция закрытия графического режима: closegraph(); – данная функция без каких либо параметров и закрывает текущий графический режим.

Функция определения графического драйвера и видео режима: detectgraph( &grdriver, &grmode);

Функция, возвращающая структуру определения палитры: getdefaultpalette(); – возвращает указатель на заданную по умолчанию структуру палитры для текущего драйвера инициализированного по initgraph.

getdrivername(); – функция возвращает имя текущего графического драйвера.

getgraphmode(); – функция возвращает текущий графический режим.

getmoderange( grdriver, &low, &high); – функция возвращает диапазон режимов для данного графического драйвера, где grdriver, low, high – переменные типа integer.

graphdefaults(); – сбрасывает текущие настройки в настройки по умолчанию.

\_graphgetmem(); – функция выделения памяти. Эту функцию вызывают подпрограммы в графической библиотеке, чтобы распределить память. Данную функцию можно использовать независимо от графической библиотеки, вызвав её с указанием нужного объёма памяти в байтах.

\_graphfreemem(); – функция освобождения памяти. Используется в сочетании с \_graphgetmem().

setgraphbufsize(); – функция изменяет внутренний размер графического буфера. Вызывается до initgraph().

graphresult(); – возвращает код ошибки для последней неудачно выполненной операции.

installuserdriver( int \*name, inthuge( \*detect)); – функция устанавливает драйвер определённый программистом. \*name – название нового драйвера, \*detect – указатель на дополнительную функцию авто определения, которая может сопровождать новый драйвер. Эта функция (авто определения) не берёт ни каких параметров и возвращает целочисленное значение.

registerbgidriver( \*grdriver) – Функция сообщает графической системе, что драйвер устройства, на который указывает \*grdriver был включён во время “линковки”.

registerbgifont( \*font) – Функция информирует графическую систему, что шрифт, на который указывает \*font был включен во время “линковки”.

restorecrtmode() – Функция восстанавливает режим, существовавший до активизации графического режима функцией initgraph().

arc( intx, inty, intstangle, intendangle, intradius); – Функция рисует дугу, по заданным координатам x,y, начальным и конечным углом stangle, endangle и заданным радиусом radius.

bar( intleft, inttop, intright, intbottom); – Функция рисует прямоугольник начиная от левого верхнего угла с координатами left, top до правого нижнего угла с координатами right, bottom.

bar3d( intleft, inttop, intright, intbottom, intdepth); – Функция рисует “трёхмерный прямоугольник” или, так сказать, подобие параллелепипеда, начиная от верхнего левого угла, до правого нижнего угла и с указанием глубины depth.

circle( intx, inty, intradius); – Функция рисует окружность с центром в точке с координатами ( x, y) и радиусом radius.

cleardevice(); – Функция очищает экран в графическом режиме и возвращает текущую позицию указателя в точку с координатами ( 0, 0).

clearviewport(); – Функция стирает область просмотра и перемещает текущую позицию в точку с координатами ( 0, 0), относительно области просмотра.

drawpoly( intnumpoints, int \*polypoints); – Функция рисует полигон с количеством вершин numpoints и координатами вершин \*polypoints. \*polypoints является одномерным массивом.

ellipse( intx, inty, intstangle, intendangle, intxradius, intyradius); – Функция рисует эллипс с координатами в точке ( x, y), начальным и конечным углами stangle, endangle, и радиусами xradius, yradius по осям x и y.

fillpoly( intnumpoints, int \*polypoints); – Функция рисует и закрашивает заданным цветом полигон.

fillelipse( intx, inty, intxradius, intyradius); – Функция рисует закрашенный эллипс.

floodfill( intx, inty, intborder); – Область, ограниченная цветной границей закрашивается установленным образцом и цветом.

getarccoords( &arcinfo); – Функция возвращает координаты последней точки, на которую был переведён указатель, где параметр &arcinfo объявлен как structarccoordstypearcinfo.

getaspectratio( &xasp, &yasp); – Получает значения коэффициента сжатия в &xasp, &yasp, где xasp, yasp являются переменными типа integer.

getbkcolor() – Возвращает значение типа integer текущего цвета фона.

getcolor() – Возвращает значение типа integer установленного на данный момент цвета.

getdefaultpalette() – ( дополнение) Возвращает структуру определения палитры в переменную объявленную как structpalettetypefar \*<имя\_переменной>=(void \*) 0;.

getfillpattern( charfar \*pattern); – Копирует определённый пользователем заполненный образец в память.

getfillsetting( &fillinfo); – Получает информацию о текущих настройках стиля и цвета, где параметр &fillinfo объявлен как structfillsettingstypefillinfo.

getimage( intleft, inttop, intright, intbottom, voidfar \*bitmap); – Сохраняет картинку в специально отведённой области памяти, где left, top, right, bottom координаты картинки, а \*bitmap указатель на область памяти.

getlinesettings( &lineinfo); – Получает текущий тип линии, образец и толщину, где параметр &lineinfo объявлен как struct linesettingstypelineinfo.

getmaxcolor(); – Возвращает значение типа integer, которое соответствует номеру цвета.

getmaxmode(); – Возвращает значение типа integer, которое соответствует максимальному номеру графического режима, для текущего драйвера.

getmaxx(); – Возвращает значение типа integer, которое соответствует максимальному количеству точек по оси x для текущего графического режима.

getmaxy(); – Возвращает значение типа integer, которое соответствует максимальному количеству точек по оси y для текущего графического режима.

getmodename( intmode\_number); – Возвращает значение типа integer, которое соответствует указанному графическому режиму.

getpalette( &palette); – Получает информацию о текущей палитре, где параметр &palette объявлен как structpalettetypepalette.

getpalettesize(); – Функция возвращает значение типа integer, соответствующее количеству цветов палитры для текущего графического режима.

getpixel( intx, inty); – Функция возвращает номер цвета точки, находящейся по координатам ( x, y).

gettextsettings( &textinfo); – Функция возвращает текущие настройки фонта для текущего графического режима, где параметр &textinfo определён как structtextsettingstypetextinfo.

getviewsettings( &viewinfo); – Получает информацию о текущей области просмотра, где параметр &viewinfo определяется как structviewporttypeviewinfo.

getx(); – Функция типа integer возвращает текущую позицию указателя по оси x.

gety(); – Функция типа integer возвращает текущую позицию указателя по оси y.

grapherrormsg( interrorcode); – Возвращает код ошибки в виде переменной errorcode типа intger.

imagesize( intleft, inttop, intright, intbottom); – Определяет размер области памяти, необходимый для сохранения двоичного образа.

installuserfont( charfar \*name); – Функция типа intger возвращает номер идентификатора шрифта, который можно передать функции settextstyle для выбора данного шрифта. Параметр \*name это путь к файлу шрифта. Одновременно данной функцией можно подключить до двадцати шрифтов.

line( intx0, inty0, intx1, inty1); – Функция рисует линию от точки с координатами x0, y0 до точки с координатами x1, y1. Не обновляет текущую позицию.

linerel( intdx, intdy); – Функция рисует линию от текущей позиции до точки, находящейся на относительном расстоянии от текущей позиции, затем передвигает текущую позицию.

lineto( intx, inty); – Рисует линию от текущей позиции до точки с координатами ( x, y), затем переносит текущую позицию в ( x, y).

moverel( intdx, intdy); – Перемещает текущую позицию на относительное растояние.

moveto( intx, inty); – Перемещает текущую позицию в точку с координатами ( x, y).

outtext( charfar \*textstring); – Выводит строку в графическом режиме. Строка должна заключатся в “”.

outtextxy( intx, inty, charfar \*textstring); – Выводит строку в графическом режиме предварительно перейдя в точку с координатами ( x, y).

pieslice( int x, int y, int stangle, int endangle, int radius); – Рисуетизакрашиваетзамкнутыйсекторкруга.

putimage( int left, int top, void far \*bitmap, int op); – Выводитбитовыйобразнаэкран.

putpixel( int x, int y, int color); – Рисуетточкускоординатами( x, y)ицветомcolor.

rectangle( int left, int top, int right, int bottom); – Рисуетпрямоугольникотточкискоординатами(left, top)доточкискоординатами( right, bottom).

Registerfarbgidriver(voidfar \*driver); – Данная функция используется для регистрации удалённых графических драйверов.

Registerfarbgifont (voidfar \*font); – Данная функция используется для регистрации удалённых шрифтов.

sector (intx, inty, intstangle, intendangle, intxradius, intyradius); – Функция рисует сектор круга установленным цветом, затем заполняет его используя образец и цвет, установленные функциями setfillstyle или setfillpattern.

setactivepage( intpage); – Устанавливает активную страницу для графического вывода.

setallpalette( &palette); – Устанавливает полную палитру цветов для данного режима, где переменная &palette определена как structpalettetypepalette.

setaspectratio( intxasp, intyasp); – Заменяет заданный по умолчанию аспект сжатия графической системы.

setbkcolor( intcolor); – Устанавливает цвет фона по его номеру заданному переменной color.

setcolor( intcolor); – Устанавливает цвет по его номеру заданному переменной color.

setfillpattern( charfar \*upattern, intcolor); – Выбирает определённый пользователем образец заливки.

setfillstyle( intpattern, intcolor); – Функция устанавливает образец и цвет заливки.

setgraphmode( intmode); – Устанавливает систему в графический режим, определённый параметром mode. Сбрасывает все настройки в положение “по умолчанию”. Очищает экран.

setlinestyle( intlinestyle, unsignedupattern, intthickness); – Устанавливает стиль рисуемых линий для функций рисования line, lineto, rectangle, drawpoly, etc.

setpalette( intcolornum, intcolor); – Устанавливает палитру от цвета colornum до цвета color.

setrgbpalette( intcolornum, intred, intgreen, intblue); – Данная функция может использоваться с IBM 8514 и VGA драйверами.

settextjustify( inthoriz, intvert); – Устанавливает выравнивание текста для графического режима. По умолчанию стоит выравнивание по верхнему левому углу.

settextstyle( intfont, intdirection, intcharsize); – Устанавливает характеристики текста: стиль, положение( горизонтальное или вертикальное), размер шрифта.

setusercharsize( intmultx, intdivx, intmulty, intdivy); – Устанавливает определённый пользователем “коэффициент сжатия” шрифта. По умолчанию ширина шрифта определена как multx:divx, высота как multy:divy.

setviewport( intleft, inttop, intright, intbottom, intclip); – Устанавливает новую область просмотра для графического вывода. Параметр clip определяет отсечены или нет все рисунки текущей области просмотра.

setvisualpage( intpage); – Устанавливает страницу с номером page визуальной графической страницей.

setwritemode( intmode); – Устанавливает режим вывода объектов в графическом режиме. Параметр mode может принимать значения 0, которое соответствует режиму copy\_put или 1, соответствующее режиму xor\_put. Параметр copy\_put использует ассемблерную команду MOV, рисуя поверх всех объектов новый объект. Параметр xor\_put использует команду XOR (команду неэквивалентности), замещая предыдущий объект таким же новым. Используя дважды параметр xor\_put, вы получите первоначальный результат.

textheight( charfar \*textstring); – Функция возвращает высоту строки в количестве точек.

textwidth( charfar \*textstring); – Функция возвращает ширину строки в количестве точек.

initgraph( &grdriver, &grmode, “path”); – функция инициализации графического режима, где &grdriver, &grmode, “path” – параметры загружаемого режима;

grdriverи grmode– переменные типа integer;

“path” – путь к загружаемому драйверу графического режима.

Функция закрытия графического режима: closegraph(); – данная функция без каких либо параметров и закрывает текущий графический режим.

Функция определения графического драйвера и видео режима: detectgraph( &grdriver, &grmode);

Функция, возвращающая структуру определения палитры: getdefaultpalette(); – возвращает указатель на заданную по умолчанию структуру палитры для текущего драйвера инициализированного по initgraph.

getdrivername(); – функция возвращает имя текущего графического драйвера.

getgraphmode(); – функция возвращает текущий графический режим.

getmoderange( grdriver, &low, &high); – функция возвращает диапазон режимов для данного графического драйвера, где grdriver, low, high – переменные типа integer.

graphdefaults(); – сбрасывает текущие настройки в настройки по умолчанию.

\_graphgetmem(); – функция выделения памяти. Эту функцию вызывают подпрограммы в графической библиотеке, чтобы распределить память. Данную функцию можно использовать независимо от графической библиотеки, вызвав её с указанием нужного объёма памяти в байтах.

\_graphfreemem(); – функция освобождения памяти. Используется в сочетании с \_graphgetmem().

setgraphbufsize(); – функция изменяет внутренний размер графического буфера. Вызывается до initgraph().

graphresult(); – возвращает код ошибки для последней неудачно выполненной операции.

installuserdriver( int \*name, inthuge( \*detect)); – функция устанавливает драйвер определённый программистом. \*name – название нового драйвера, \*detect – указатель на дополнительную функцию авто определения, которая может сопровождать новый драйвер. Эта функция (авто определения) не берёт ни каких параметров и возвращает целочисленное значение.

registerbgidriver( \*grdriver) – Функция сообщает графической системе, что драйвер устройства, на который указывает \*grdriver был включён во время “линковки”.

registerbgifont( \*font) – Функция информирует графическую систему, что шрифт, на который указывает \*font был включен во время “линковки”.

restorecrtmode() – Функция восстанавливает режим, существовавший до активизации графического режима функцией initgraph().

arc( intx, inty, intstangle, intendangle, intradius); – Функция рисует дугу, по заданным координатам x,y, начальным и конечным углом stangle, endangle и заданным радиусом radius.

bar( intleft, inttop, intright, intbottom); – Функция рисует прямоугольник начиная от левого верхнего угла с координатами left, top до правого нижнего угла с координатами right, bottom.

bar3d( intleft, inttop, intright, intbottom, intdepth); – Функция рисует “трёхмерный прямоугольник” или, так сказать, подобие параллелепипеда, начиная от верхнего левого угла, до правого нижнего угла и с указанием глубины depth.

circle( intx, inty, intradius); – Функция рисует окружность с центром в точке с координатами ( x, y) и радиусом radius.

cleardevice(); – Функция очищает экран в графическом режиме и возвращает текущую позицию указателя в точку с координатами ( 0, 0).

clearviewport(); – Функция стирает область просмотра и перемещает текущую позицию в точку с координатами ( 0, 0), относительно области просмотра.

drawpoly( intnumpoints, int \*polypoints); – Функция рисует полигон с количеством вершин numpoints и координатами вершин \*polypoints. \*polypoints является одномерным массивом.

ellipse( intx, inty, intstangle, intendangle, intxradius, intyradius); – Функция рисует эллипс с координатами в точке ( x, y), начальным и конечным углами stangle, endangle, и радиусами xradius, yradius по осям x и y.

fillpoly( intnumpoints, int \*polypoints); – Функция рисует и закрашивает заданным цветом полигон.

fillelipse( intx, inty, intxradius, intyradius); – Функция рисует закрашенный эллипс.

floodfill( intx, inty, intborder); – Область, ограниченная цветной границей закрашивается установленным образцом и цветом.

getarccoords( &arcinfo); – Функция возвращает координаты последней точки, на которую был переведён указатель, где параметр &arcinfo объявлен как structarccoordstypearcinfo.

getaspectratio( &xasp, &yasp); – Получает значения коэффициента сжатия в &xasp, &yasp, где xasp, yasp являются переменными типа integer.

getbkcolor() – Возвращает значение типа integer текущего цвета фона.

getcolor() – Возвращает значение типа integer установленного на данный момент цвета.

getdefaultpalette() – ( дополнение) Возвращает структуру определения палитры в переменную объявленную как structpalettetypefar \*<имя\_переменной>=(void \*) 0;.

getfillpattern( charfar \*pattern); – Копирует определённый пользователем заполненный образец в память.

getfillsetting( &fillinfo); – Получает информацию о текущих настройках стиля и цвета, где параметр &fillinfo объявлен как structfillsettingstypefillinfo.

getimage( intleft, inttop, intright, intbottom, voidfar \*bitmap); – Сохраняет картинку в специально отведённой области памяти, где left, top, right, bottom координаты картинки, а \*bitmap указатель на область памяти.

getlinesettings( &lineinfo); – Получает текущий тип линии, образец и толщину, где параметр &lineinfo объявлен как struct linesettingstypelineinfo.

getmaxcolor(); – Возвращает значение типа integer, которое соответствует номеру цвета.

getmaxmode(); – Возвращает значение типа integer, которое соответствует максимальному номеру графического режима, для текущего драйвера.

getmaxx(); – Возвращает значение типа integer, которое соответствует максимальному количеству точек по оси x для текущего графического режима.

getmaxy(); – Возвращает значение типа integer, которое соответствует максимальному количеству точек по оси y для текущего графического режима.

getmodename( intmode\_number); – Возвращает значение типа integer, которое соответствует указанному графическому режиму.

getpalette( &palette); – Получает информацию о текущей палитре, где параметр &palette объявлен как structpalettetypepalette.

getpalettesize(); – Функция возвращает значение типа integer, соответствующее количеству цветов палитры для текущего графического режима.

getpixel( intx, inty); – Функция возвращает номер цвета точки, находящейся по координатам ( x, y).

gettextsettings( &textinfo); – Функция возвращает текущие настройки фонта для текущего графического режима, где параметр &textinfo определён как structtextsettingstypetextinfo.

getviewsettings( &viewinfo); – Получает информацию о текущей области просмотра, где параметр &viewinfo определяется как structviewporttypeviewinfo.

getx(); – Функция типа integer возвращает текущую позицию указателя по оси x.

gety(); – Функция типа integer возвращает текущую позицию указателя по оси y.

grapherrormsg( interrorcode); – Возвращает код ошибки в виде переменной errorcode типа intger.

imagesize( intleft, inttop, intright, intbottom); – Определяет размер области памяти, необходимый для сохранения двоичного образа.

installuserfont( charfar \*name); – Функция типа intger возвращает номер идентификатора шрифта, который можно передать функции settextstyle для выбора данного шрифта. Параметр \*name это путь к файлу шрифта. Одновременно данной функцией можно подключить до двадцати шрифтов.

line( intx0, inty0, intx1, inty1); – Функция рисует линию от точки с координатами x0, y0 до точки с координатами x1, y1. Не обновляет текущую позицию.

linerel( intdx, intdy); – Функция рисует линию от текущей позиции до точки, находящейся на относительном расстоянии от текущей позиции, затем передвигает текущую позицию.

lineto( intx, inty); – Рисует линию от текущей позиции до точки с координатами ( x, y), затем переносит текущую позицию в ( x, y).

moverel( intdx, intdy); – Перемещает текущую позицию на относительное растояние.

moveto( intx, inty); – Перемещает текущую позицию в точку с координатами ( x, y).

outtext( charfar \*textstring); – Выводит строку в графическом режиме. Строка должна заключатся в “”.

outtextxy( intx, inty, charfar \*textstring); – Выводит строку в графическом режиме предварительно перейдя в точку с координатами ( x, y).

pieslice( int x, int y, int stangle, int endangle, int radius); – Рисуетизакрашиваетзамкнутыйсекторкруга.

putimage( int left, int top, void far \*bitmap, int op); – Выводитбитовыйобразнаэкран.

putpixel( int x, int y, int color); – Рисуетточкускоординатами( x, y)ицветомcolor.

rectangle( int left, int top, int right, int bottom); – Рисуетпрямоугольникотточкискоординатами(left, top)доточкискоординатами( right, bottom).

Registerfarbgidriver(voidfar \*driver); – Данная функция используется для регистрации удалённых графических драйверов.

Registerfarbgifont (voidfar \*font); – Данная функция используется для регистрации удалённых шрифтов.

sector (intx, inty, intstangle, intendangle, intxradius, intyradius); – Функция рисует сектор круга установленным цветом, затем заполняет его используя образец и цвет, установленные функциями setfillstyle или setfillpattern.

setactivepage( intpage); – Устанавливает активную страницу для графического вывода.

setallpalette( &palette); – Устанавливает полную палитру цветов для данного режима, где переменная &palette определена как structpalettetypepalette.

setaspectratio( intxasp, intyasp); – Заменяет заданный по умолчанию аспект сжатия графической системы.

setbkcolor( intcolor); – Устанавливает цвет фона по его номеру заданному переменной color.

setcolor( intcolor); – Устанавливает цвет по его номеру заданному переменной color.

setfillpattern( charfar \*upattern, intcolor); – Выбирает определённый пользователем образец заливки.

setfillstyle( intpattern, intcolor); – Функция устанавливает образец и цвет заливки.

setgraphmode( intmode); – Устанавливает систему в графический режим, определённый параметром mode. Сбрасывает все настройки в положение “по умолчанию”. Очищает экран.

setlinestyle( intlinestyle, unsignedupattern, intthickness); – Устанавливает стиль рисуемых линий для функций рисования line, lineto, rectangle, drawpoly, etc.

setpalette( intcolornum, intcolor); – Устанавливает палитру от цвета colornum до цвета color.

setrgbpalette( intcolornum, intred, intgreen, intblue); – Данная функция может использоваться с IBM 8514 и VGA драйверами.

settextjustify( inthoriz, intvert); – Устанавливает выравнивание текста для графического режима. По умолчанию стоит выравнивание по верхнему левому углу.

settextstyle( intfont, intdirection, intcharsize); – Устанавливает характеристики текста: стиль, положение( горизонтальное или вертикальное), размер шрифта.

setusercharsize( intmultx, intdivx, intmulty, intdivy); – Устанавливает определённый пользователем “коэффициент сжатия” шрифта. По умолчанию ширина шрифта определена как multx:divx, высота как multy:divy.

setviewport( intleft, inttop, intright, intbottom, intclip); – Устанавливает новую область просмотра для графического вывода. Параметр clip определяет отсечены или нет все рисунки текущей области просмотра.

setvisualpage( intpage); – Устанавливает страницу с номером page визуальной графической страницей.

setwritemode( intmode); – Устанавливает режим вывода объектов в графическом режиме. Параметр mode может принимать значения 0, которое соответствует режиму copy\_put или 1, соответствующее режиму xor\_put. Параметр copy\_put использует ассемблерную команду MOV, рисуя поверх всех объектов новый объект. Параметр xor\_put использует команду XOR (команду неэквивалентности), замещая предыдущий объект таким же новым. Используя дважды параметр xor\_put, вы получите первоначальный результат.

textheight( charfar \*textstring); – Функция возвращает высоту строки в количестве точек.

textwidth( charfar \*textstring); – Функция возвращает ширину строки в количестве точек.

**Вопросы на защиту:**

1. Графика в Си. Координаты точек. Основные цвета. Состав графического пакета.
2. Графика в Си. Основные графические функции. setcolor, line, rectangle, bar, circle, floodfill, outtextxy, getimage, delay.
3. Процедуры. Синтаксис. Параметры формальные и фактические.
4. Функция. Формальные и фактические параметры. Синтаксис.
5. Функция. Определение функции (семантика), объявление функции (прототип, сигнатура), вызов функции.

# Задания к лабораторной работе №4

**Примитивная графика. Исследование функций графических библиотек**

**Постановка задачи:**

1. Выбрать объект, в соответствие с вариантом:
2. Цветок.
3. Дом.
4. Машина.
5. Телефон.
6. Компьютер.
7. Собака.
8. Человек.
9. Телевизор.
10. Книга.
11. Дерево.
12. Кошка.
13. Птица.
14. Звездное небо.
15. Планета Земля.
16. Часы.
17. Окно.
18. Тетрадь.
19. Клавиатура.
20. Первый корпус УГНТУ.
21. Общежитие №6
22. Стол
23. Кресло.
24. Принтер.
25. Стул.
26. План помещения.
27. Озеро с рыбой.
28. Изобразить объект произвольного размера и цвета.
29. Общие требования к изображению:
30. Использование более пяти различных цветов.
31. Использовать более трех функций изображения графических элементов в С.
32. Показать пример движения рисунка в произвольном направлении.
33. Предусмотреть возможность рекурсивного повторения рисунка n-раз.
34. Вывести текст на экране (по заданной теме).

[Возврат к начальной странице](#_top)

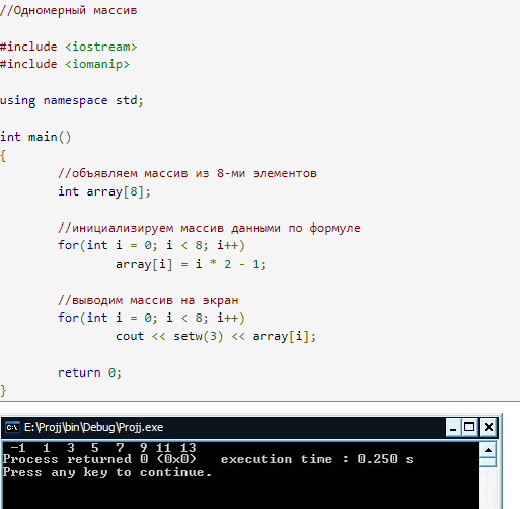
**Лабораторная работа №5**

**«Обработка одномерных массивов»**

# Теоретические сведения об одномерных массивах:

Что такое одномерный массив? Это массив, у которого есть лишь один параметр - это количество элементов. Эти элементы расположены последовательно, т.е. идут друг за другом.

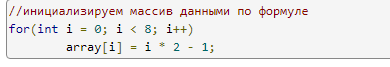
Допустим, что нам нужно объявить одномерный массив, состоящий из 8-ми элементов и присвоить его элементам значения, вычисленные по формуле: номер элемента массива, умноженный на 2 и минус 1. Т.е. для 3-го элемента будет 3 \* 2 - 1 = 5. Смотрим





Объявляется массив путем указания типа его элементов, в нашем случае это целые (*int*), затем указывается его имя, в нашем случае это *array*, а затем в квадратных скобках указывается количество элементов, в нашем случае их *восемь*. Итак, память для массива мы зарезервировали и на этапе компиляции программы она будет выделена (8 \* 4 байта = 32 байта, т.к один элемент типа *int* занимает в памяти 4 байта). Но в массиве на данный момент содержится различный "мусор", т.е. различные значения, которые возможно применялись в других программах и так далее. Нужно изменить (задать) эти значения, т.е. выполнить инициализацию массива. Проще всего ее выполнять с помощью цикла, в котором мы последовательно проходим по всем элементам массива. Инициализируем, применяя формулу i \* 2 - 1, где i - значение индекса массива.

//инициализируем массив данными по формуле



Теперь каждый элемент нашего одномерного массива содержит свое заданное значение. Хотелось бы их просмотреть, а для этого нам нужно подумать, как вывести их на экран. Для этого опять же проще всего использовать цикл, в котором мы будет последовательно проходить по всем элементам массива, выводя их на экран в одну строку. Для каждого значения мы выделим на экране место в 3 ячейки, для этого будем использовать манипулятор вывода потока setw с параметром 3.



Теория - это хорошо, но практика лучше. Немного теории об одномерных массивах я вам рассказал, а теперь давайте займемся практикой решения задач.

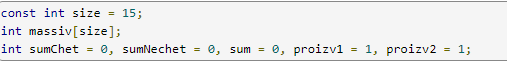
#### Одномерные массивы - задача № 1

Создать одномерный массив, содержащий 15 элементов, наполнить его случайными значениями в интервале от 1 до 200. Посчитать и вывести:

1. Сумму всех четных элементов массива.
2. Сумму всех нечетных элементов массива.
3. Сумму всех элементов массива.
4. Произведение всех элементов массива, значения которых меньше 50.
5. Произведение элементов массива с индексами от 2 и 7.

Давайте во всех тонкостях программы разбираться по порядку, даже, несмотря на то, что некоторые моменты нам знакомы, заодно и повторим выученное.

Итак, для начала объявим массив и пять переменных, которые будут нам служить для хранения результатов работы с элементами массива



Как мы увидим позже, размер массива часто используется в различных моментах кода программы (например, в циклах прохода по массиву), поэтому удобнее его объявлять не конкретным числом в самом объявлении массива, а отдельно, как константу. Почему как константу? Потому что размер массива не может изменяться во время работы программы, на то он и статический, а не динамический (работа с динамическими массивами будет рассматриваться позже, это отдельная тема). Для того, чтобы компилятор знал о том, что это константа и не выдавал ошибку во время компиляции программы, нужно обязательно объявлять константное значение, в частности размер массива, со спецификатором *const*.

Массив мы объявили, теперь нужно выполнить его инициализацию. По заданию его нужно заполнить случайными значениями от 1 до 200. Естественно, для этого нам нужно использовать функцию рандомизации (*rand()* - вырабатывающую случайным значения по определенному алгоритму). Для того, чтобы функция работала правильно ее нужно использовать в паре с другой функцией (*srand()* ), позволяющей задавать начальную точку отчета для работы функции рандомизации rand(). Эти две функции описаны в заголовочном файле *stdlib.h*. Для начальной точки отчета функция *srand()* использует системное время, поэтому подключаем заголовочный файл *time.h*.

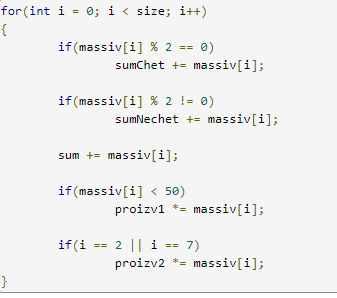


Задаем нашему одномерному массиву случайные значения



Естественно, это удобнее всего делать в цикле, проходя по всем элементам массива последовательно и присваивая каждому свое случайное значение. Для того, чтобы функция *rand()* генерировала случайные величины в интервале от 1 до 200, нужно использовать вот такой синтаксис: *начальная точка* плюс *rand()*, остаток от деления (знак %) *конечная точка*.

Теперь начинается самая важная работа - это подсчет нужных сумм и произведений. Опять же для прохода по массиву используем цикл *for*



Как обычно, для прохода по всему одномерному массиву мы используем цикл *for* в интервале от 0 до *size*, где *size* является размером массива. Условие используется строгое, без включения самого *size*, т.к. нумерация элементов массива начинается с нуля. Т.е., если размер массива равен 15, то индексы его будут от 0 до 14 включительно, что есть 15 штук.

Далее нам по условию нужно найти сумму всех четных элементов массива. Для этого мы будем использовать соответствующее условие выборки



Думаю здесь все понятно: если остаток от деления значения элемента на 2 равен нулю, то значит значение этого элемента массива четное и, соответственно, его можно приплюсовывать в общую сумму. Т.к. переменную *sumChet* мы используем для накопления общей суммы, то при объявлении мы сразу ее инициализировали значением 0. Если вы забыли, то напомню, что это сокращенное выражение *sumChet += massiv[i];* в полной форме будет таким *sumChet = sumChet + massiv[i];*

Далее по условию нам нужно найти сумму всех нечетных элементов массива



Для нахождения этой сумму используется обратное условие, т.е. если остаток от деления не равен 0, то выполняем приплюсовывание суммы в общую переменную накопления.

Сумма всех элементов одномерного массива считается довольно просто



Нет никаких условий, все элементы массива просто последовательно, поочередно складываются.



Для подсчета произведения всех элементов, значения которых менее 50 используем соответствующее условие выборки.

И последнее, что нам осталось сделать, это найти произведение элементов массива с индексами 2 и 7. Задаем условие, что если в данный момент мы находимся на элементах массива 2 или (||) 7, то выполнить умножение



Полный код программы будет выглядеть так:

*//Работа с одномерным массивом*

*#include <iostream>*

*#include <stdlib.h>*

*#include <time.h>*

*//прототип функции для печати массива*

*void printArray(int[], const int);*

*using namespace std;*

*int main()*

*{*

//объявляем необходимые переменные

const int size = 15;

int massiv[size];

int sumChet = 0, sumNechet = 0, sum = 0, proizv1 = 1, proizv2 = 1;

//задаем начало отчета для рандомизатора rand()

srand(time(NULL));

//наполняем массив случайными величинами

for(int i = 0; i < size; i++)

massiv[i] = 1 + rand() % 200;

//выводим содержимое массива на экран

printArray(massiv, size);

//выполняем необходимые подсчеты в цикле

for(int i = 0; i < size; i++)

{

if(massiv[i] % 2 == 0)

sumChet += massiv[i];

if(massiv[i] % 2 != 0)

sumNechet += massiv[i];

sum += massiv[i];

if(massiv[i] < 50)

proizv1 \*= massiv[i];

if(i == 2 || i == 7)

proizv2 \*= massiv[i];

}

//выводим результаты работы программы

cout << "\nSumma chetnih: " << sumChet << endl;

cout << "Summa nechetnih: " << sumNechet << endl;

cout << "Summa vseh: " << sum << endl;

cout << "Proizvedeniye elementov do 50: " << proizv1 << endl;

cout << "Proizvedeniye elementov ot 0 do 4: " << proizv2 << endl;

return 0;

}

//функция вывода массива на экран

void printArray(int a[], const int s)

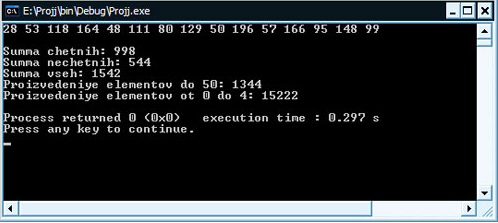
{

for(int k = 0; k < s; k++)

cout << a[k] << " ";

cout << endl;

}



**Вопросы на защиту:**

1. Массивы данных. Основные понятия. Объявление массива.
2. Способы ввода данных в массив. Синтаксис, блок схемы.
3. Массивы данных. Ввод с клавиатуры и вывод на экран. Заполнение случайными числами.
4. Массивы данных. Работа с текстовыми и двоичными файлами. Функции fread и fwrite.
5. Массивы данных. Перестановка элементов массива. Инверсия, циклический сдвиг.
6. Массивы данных. Сортировка массивов. Метод пузырька.
7. Массивы данных. Сортировка массивов. Метод выбора минимального элемента.
8. Массивы данных. Сортировка массивов. Двоичный поиск в массиве.

### 

### **Задание к лабораторной работе № 5**

### **«Обработка одномерных массивов»**

### **Постановка задачи:**

Написать программу выполняющую:

1) получение элементов массива по формуле;

### 2) вычисление параметра полученного массива;

3) определение параметра массива;

4) упорядочение элементов массива.

5) вывести полученные данные в файл.

Организовать обработку массива с использованием ***функций***.

**Варианты:**

|  |  |
| --- | --- |
| 1  1. .  2. Вычислить среднеарифметическое значение четных по значению элементов первой трети массива.  3. Определить наибольший по абсолютной величине среди четных по номеру элементов второй четверти массива.  4. Упорядочить каждые третьи элементы последней трети массива по возрастанию обратных значений. | 2  1.  2. Вычислить среднегеометрическое значение ненулевых элементов всего массива.  3. Определить наименьший элемент среди нечетных по значению элементов последней трети массива.  4. Упорядочить нечётные по номеру элементы последней четверти массива по убыванию квадратов значений. |
| 3  1. .  2. Вычислить среднеарифметическое значение положительных элементов первой половины массива.  3. Определить наименьший по модулю элемент среди четных по значению элементов второй трети массива.  4. Упорядочить отрицательные элементы первой трети массива по убыванию абсолютных значений. | 4  1. .  2. Вычислить сумму отрицательных элементов второй трети массива.  3. Определить наибольший элемент среди нечетных по номеру элементов первой половины массива.  4. Упорядочить чётные по номеру элементы первой четверти массива по убыванию значений. |
| 5  1. .  2. Вычислить произведение четных по значению элементов третьей четверти массива.  3. Определить максимальное значение среди каждых третьих элементов второй половины массива.  4. Упорядочить отрицательные элементы последней четверти массива по убыванию модулей обратных значений. | 6  1.  2. Вычислить среднегеометрическое значение ненулевых элементов всего массива.  3. Определить наибольший элемент среди нечетных по номеру элементов второй четверти массива.  4. Упорядочить нечётные по номеру элементы третьей четверти массива по убыванию квадратов значений. |
| 7  1.  2. Вычислить сумму каждых вторых элементов второй половины массива.  3. Определить наименьший элемент среди ненулевых элементов второй четверти массива.  4. Упорядочить ненулевые элементы всего массива по возрастанию абсолютных значений. | 8  1.  2. Вычислить среднеарифметическое значение ненулевых элементов последней трети массива.  3. Определить наибольший по абсолютной величине среди каждых третьих элементов второй половины массива.  4. Упорядочить каждые третьи элементы последней трети массива по возрастанию обратных значений. |
| 9  1.  2. Вычислить произведение отрицательных элементов второй трети массива.  3. Определить максимальное значение среди четных по значению элементов второй трети массива.  4. Упорядочить нечётные по номеру элементы первой трети массива по убыванию модулей обратных значений. | 10  1.  2. Вычислить произведение каждых третьих элементов второй четверти массива.  3. Определить минимальное значение среди ненулевых элементов второй четверти массива.  4. Упорядочить чётные по номеру элементы первой четверти массива по убыванию значений. |
| 11  1.  2. Вычислить среднеарифметическое значение положительных элементов всего массива.  3. Определить наибольший элемент среди каждых третьих элементов второй трети массива.  4. Упорядочить ненулевые элементы третьей четверти массива по убыванию обратных значений. | 12  1.  2. Вычислить среднегеометрическое значение четных по значению элементов последней четверти массива.  3. Определить наибольший по абсолютной величине среди ненулевых  элементов первой половины массива.  4. Упорядочить нечётные по номеру элементы последней четверти массива по возрастанию абсолютных значений. |
| 13  1.  2. Вычислить среднеквадратичное значение ненулевых элементов второй половины массива.  3. Определить наименьший по модулю элемент среди четных по значению элементов второй четверти массива.  4. Упорядочить положительные элементы всего массива по убыванию абсолютных значений. | 14  1.  2. Вычислить сумму каждых третьих элементов первой трети массива.  3. Определить наименьший элемент среди нечетных по номеру элементов второй половины массива.  4. Упорядочить каждые третьи элементы третьей четверти массива по убыванию квадратов значений. |
| 15  1.  2. Вычислить среднеарифметическое значение отрицательных элементов третьей четверти массива.  3. Определить минимальное значение среди ненулевых элементов первой половины массива.  4. Упорядочить чётные по номеру элементы последней трети массива по возрастанию обратных значений. | 16  1.  2. Вычислить среднеквадратичное значение четных по значению элементов всего массива.  3. Определить наименьший элемент среди каждых третьих элементов второй четверти массива.  4. Упорядочить нечётные по номеру элементы первой четверти массива по убыванию обратных значений. |
| 17  1.  2. Вычислить произведение каждых третьих элементов второй четверти массива.  3. Определить наименьший элемент среди ненулевых элементов последней четверти массива.  4. Упорядочить элементы первой четверти массива по возрастанию значений. | 18  1.  2. Вычислить среднеарифметическое значение элементов последней трети массива.  3. Определить наибольший по абсолютной величине среди каждых пятых элементов второй половины массива.  4. Упорядочить каждые третьи элементы последней трети массива по возрастанию обратных значений. |
| 19  1.  2. Вычислить произведение отрицательных элементов второй трети массива.  3. Определить максимальное значение среди четных по значению элементов второй трети массива.  4. Упорядочить нечётные по значению элементы первой трети массива по убыванию модулей обратных значений. | 20  1.  2. Вычислить произведение каждых четвертых элементов второй трети массива.  3. Определить минимальное значение среди элементов второй четверти массива.  4. Упорядочить чётные по номеру элементы первой трети массива по убыванию обратных значений. |
| 21  1.  2. Вычислить среднеарифметическое значение положительных элементов всего массива.  3. Определить наибольший элемент среди каждых четвертых элементов второй трети массива.  4. Упорядочить элементы третьей четверти массива по убыванию обратных значений. | 22  1.  2. Вычислить среднегеометрическое значение четных по значению элементов последней четверти массива.  3. Определить наибольший по величине среди ненулевых элементов второй половины массива.  4. Упорядочить нечётные по номеру элементы последней четверти массива по возрастанию абсолютных значений. |
| 23  1.  2. Вычислить среднеквадратичное значение ненулевых элементов второй половины массива.  3. Определить наименьший по модулю элемент среди нечетных по значению элементов второй четверти массива.  4. Упорядочить положительные элементы всего массива по убыванию абсолютных значений. | |

[Возврат к начальной странице](#_top)