Министерство образования и науки Российской Федерации
федеральное государственное образовательное учреждение
высшего образования

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

А.Д. Шишкин
Е.А. Чернецова

ПРАКТИКУМ

по дисциплине

«**ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ**»

Программирование на языках Си и С++

Санкт-Петербург
РГГМУ
2020

УДК 004.438(076.5)
ББК 22.18я73

Рецензент: к.т.н., доцент, сертифицированный тренер Microsoft Ананченко И.В.

Шишкин А.Д., Чернецова Е.А. Практикум по учебной дисциплине “Информатика и программирование”. Программирование на языках Си и С++. – СПб.: РГГМУ, 2020. – 158 с.

Лабораторный практикум ”Программирование на языке Си и С++” охватывает два раздела дисциплины «Информатика и языки программирования», читаемой на кафедре "Информационные технологии и системы безопасности” для студентов, обучающихся по специальности «Информационная безопасность телекоммуникационных систем».

Содержащиеся в практикуме сведения по теории программирования, методические указания и рекомендации по выполнению работ позволяют использовать его в качестве пособия для закрепления курса лекций по разделам «Языки программирования» и «Методы программирования». Включен материал по технологии работы в пакетах Dev - Cpp и C++Builder. Пособие содержит два раздела.

В первый раздел включено девять работ для освоения программирования на языке Си. Во второй раздел включены девять работ по программированию на языке С++.

Практикум предназначен для студентов гидрометеорологического университета и может быть полезным для всех желающих ознакомиться с основами программирования на языке Си и С++.

ISBN 978-5-86813-488-3

СОДЕРЖАНИЕ

Введение в интегрированные среды программирования

Установка и настройка среды программирования Dev-Cpp

В настоящее время существуют разные среды разработки, позволяющие писать программы на разных языках программирования – в частности на C/C++. Имеется достаточное количество таких программ: как платных, так и бесплатных. Вы можете выбрать любую из них. Найти их для скачивания не составит труда – эти вопросы мы здесь обсуждать не будем. Безусловно, для платформы Windows существует большее количество таких программ, нежели для Mac OS.

Одну из них мы рассмотрим ниже. Это – **Dev–C++ (Dev-Cpp)** – бесплатная интегрированная среда разработки приложений для языков программирования C/C++. В дистрибутив среды входит компилятор MinGW. Dev–Cpp можно считать бесплатным аналогом MicrosoftVisualStudio (хотя у последнего есть бесплатная и очень урезанная версия).

Для наших целей по изучению основ программирования ее возможностей хватит "за глаза". Главное, что установить ее не сложно, как и начать работать в операционной среде Windows. Что касается Dev – C++ (Dev–Cpp), то на настоящий момент он не разрабатывается, вместо него активно разрабатывается порт интерфейса Dev– C++ на wxWidgets — wxDev-C++.

Этапы установки программы

Этап выполняется только в случае установки пакета.

1. Запуск осуществляется с помощью ярлыка, показанного на рис.1.

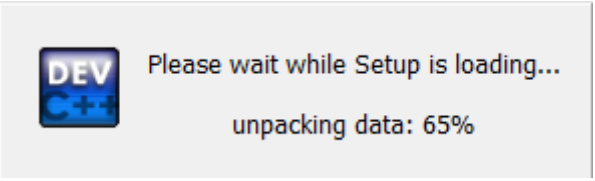


Рис. 1. Ярлык запуска

2. После распаковки архива вам предложат выбрать предпочитаемый язык интерфейса программы. В появившемся окне можно выбрать язык интерфейса. Например, английский или русский. Его можно поменять в дальнейшем на любой из имеющегося в списке, изображенного на рис 2.

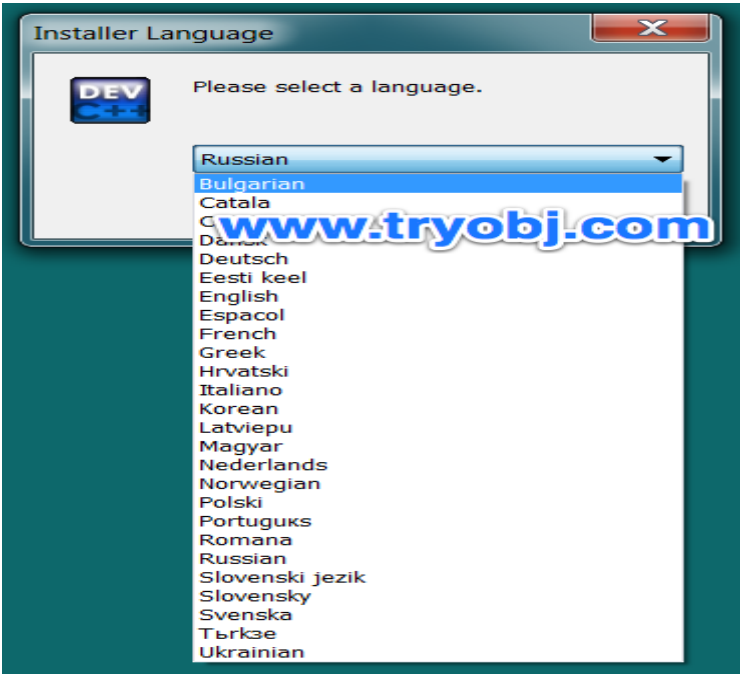


Рис. 2 . Выбор языка интерфейса

Для конкретности выберем русифицированный интерфейс. Необходимо выполнить лицензионное соглашение согласно рис.3 и рис 4.

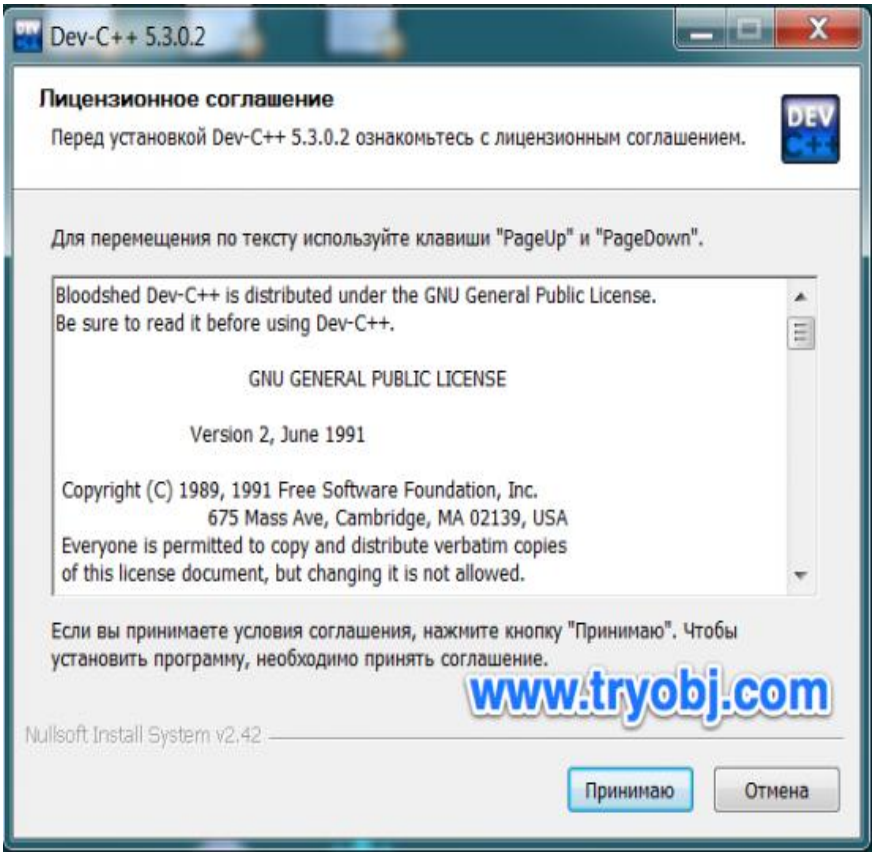


Рис. 3.Установка пакета

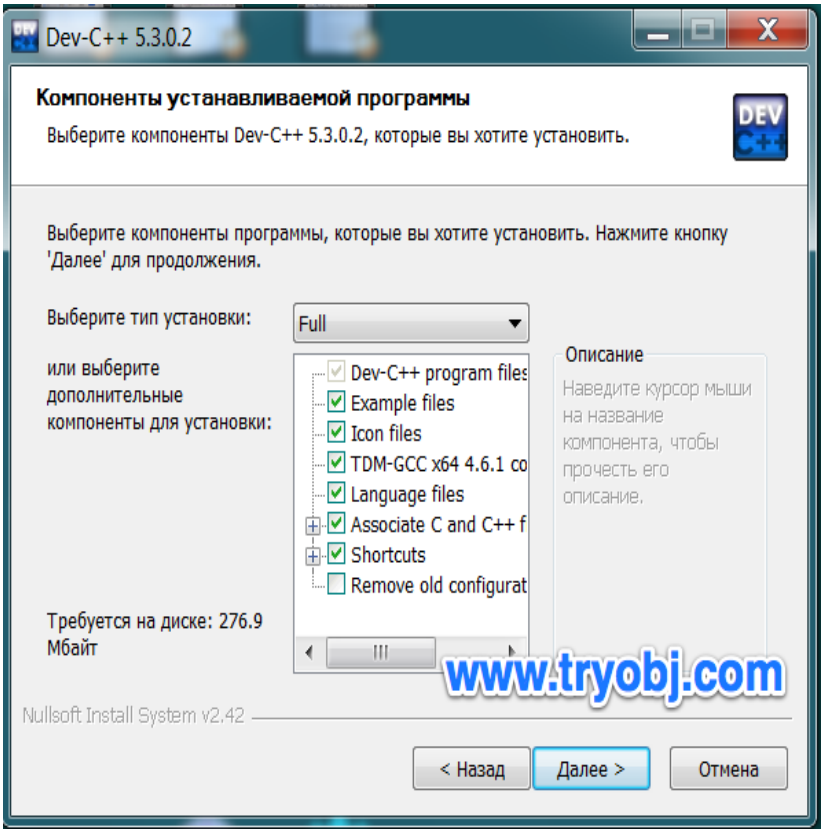


Рис. 4. Установка пакета (Далее)

3. Далее вы выбираете категорию, в которой будет установлена программа. По умолчанию это стандартный ProgramFiles. Выбрать опцию: «Установить». Демонстрация установки показана на рис.5

После принятия лицензионного соглашения вам будет предоставлена возможность выбрать тип установки программы с теми или иными компонентами.

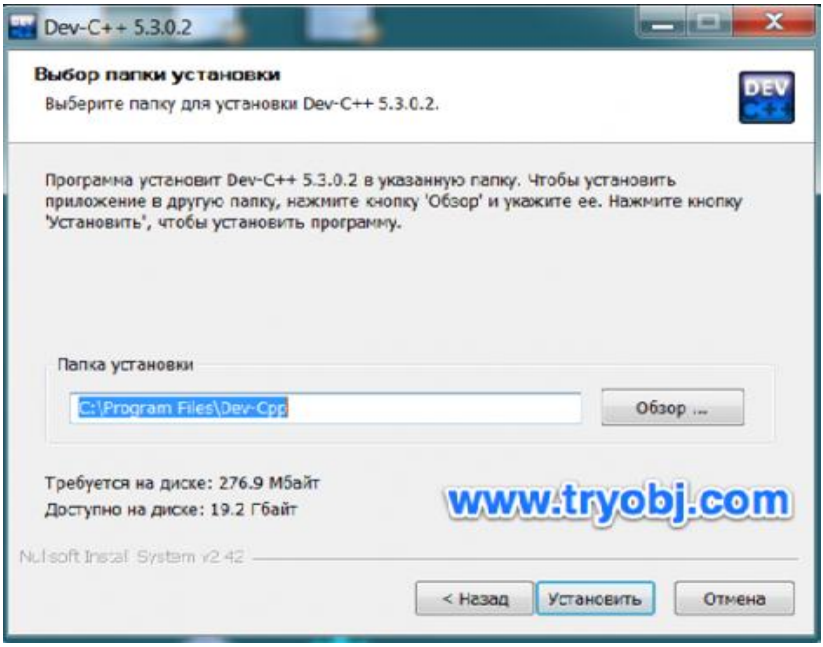


Рис.5.Установить

Установка компонентов программы заканчивается окном приглашения «Запустить Dev– Cpp», что мы и сделаем, нажав клавишу «Готово», показанную на рис. 6.

В завершении установки на экране появится открытое окно Dev– C++ с установленными компонентами. Вид окна приведен на рис .7.

В данном окне можно набрать текст программы или, если она есть, ее можно вызвать. Для создания новой программы нужно выбрать пункт: «Файл» и в открывшемся подменю: «Создать \ Исходный файл», а для открытия уже имеющегося файла нужно выбрать подменю: «Открыть \заново» или же, используя горячие клавиши **Ctrl + O,** что быстрее. Пример загрузки приведен на рис. 8.

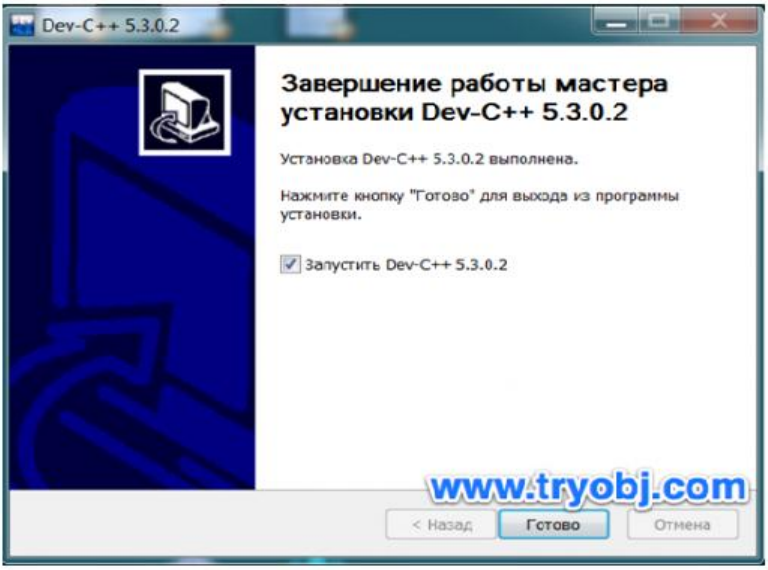


Рис. 6. Завершение установки (Готово)

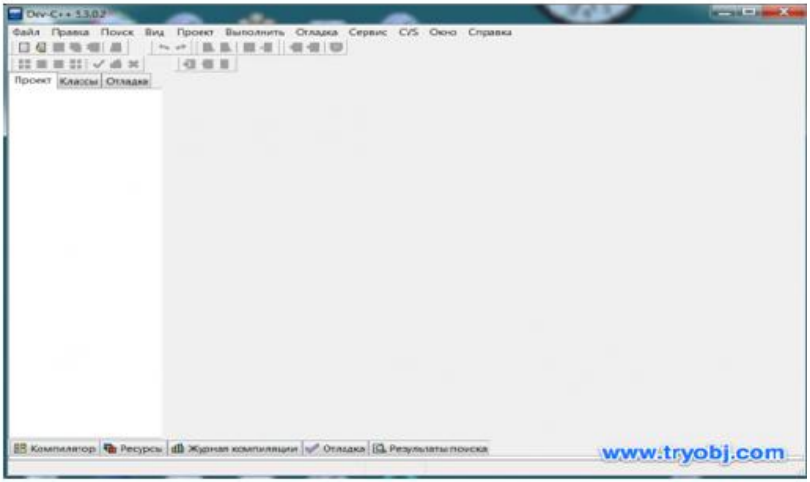


Рис.7.Открытое окно программы Dev–C++

Теперь на примере одного из файлов мы посмотрим некоторые настройки программы, которые позволят работать в Dev–C++ с большим комфортом.

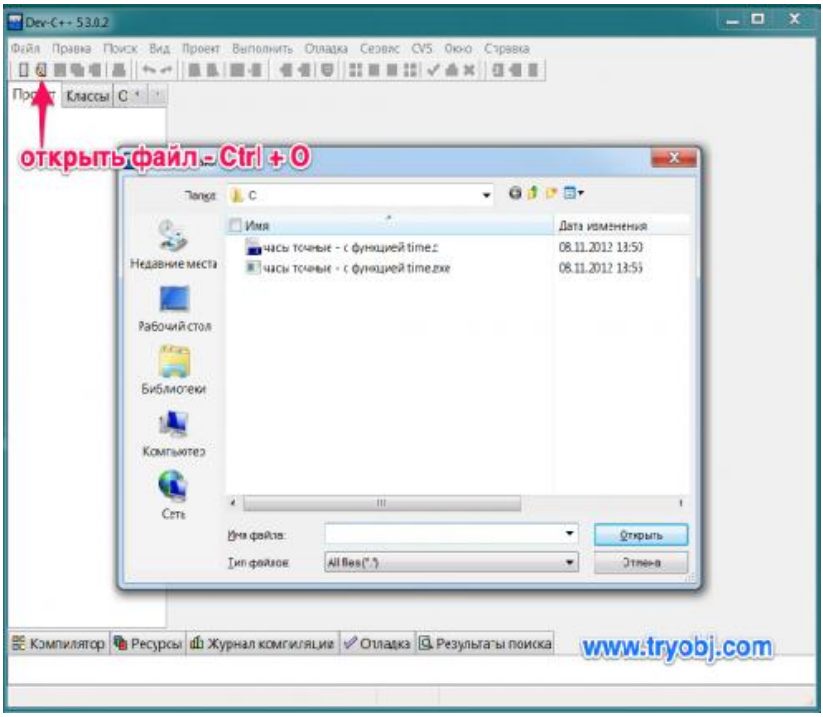


Рис.8. Загрузка файла

Любую из программ написанных на C / C++ перед тем как запустить на выполнение (чтобы посмотреть как работает написанная программа) необходимо скомпилировать и только затем запустить на выполнение.

В C++ используется три пиктограммы/возможности:

1.В окне «Выполнить» выбрать пункт: «Скомпилировать» – простая компиляция программного кода. На данном этапе компилятор проверяет написанный код на наличие ошибок и, если все в порядке – переводит код программы в исполняемый

файл ***.exe. Если же ошибки имеются, то работа компилятора прерывается и в окне "Компилятор" выводятся коды ошибок помогающих их найти и исправить. Тоже действие – набором **«Горячей клавиши «F9»**

2.В окне **«Выполнить»** выбрать пункт: **«Выполнить»** – эта команда позволяет многократно запускать наш код без повторной компиляции кода. Тоже действие – набором **«Горячей клавиши – F10».**

3. В окне **«Выполнить»** выбрать пункт: « **Скомпилировать и выполнить»**, если мы хотим сразу посмотреть выполнение нашей программы в консоли после компиляции. Тоже действие – используем пиктограмму «**Горячие клавиши - F11»**.Эти ситуации показаны на рис.9.

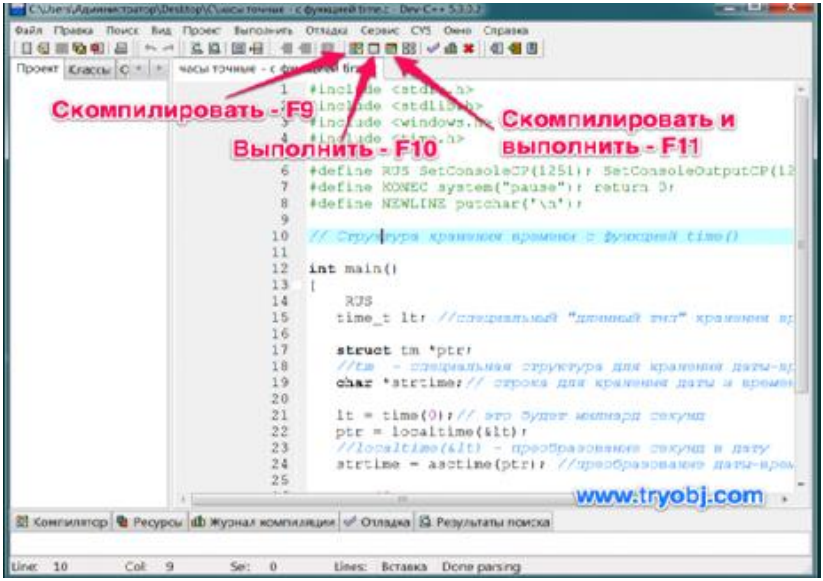


Рис 9. Отладка кода

Все запущенные программы открываются в консольном окне, которое изображено на рис.10.

В данном случае специально выбран файл имеющий русский текст, чтобы вы видели, с чем вам может быть придется столкнуться:

черное окно с белым текстом и нечитаемыми символами. Сейчас мы исправим это. Правой кнопкой мыши кликаем на верхней части консольного окна и выбираем в самом низу пункт – "**Свойства**".

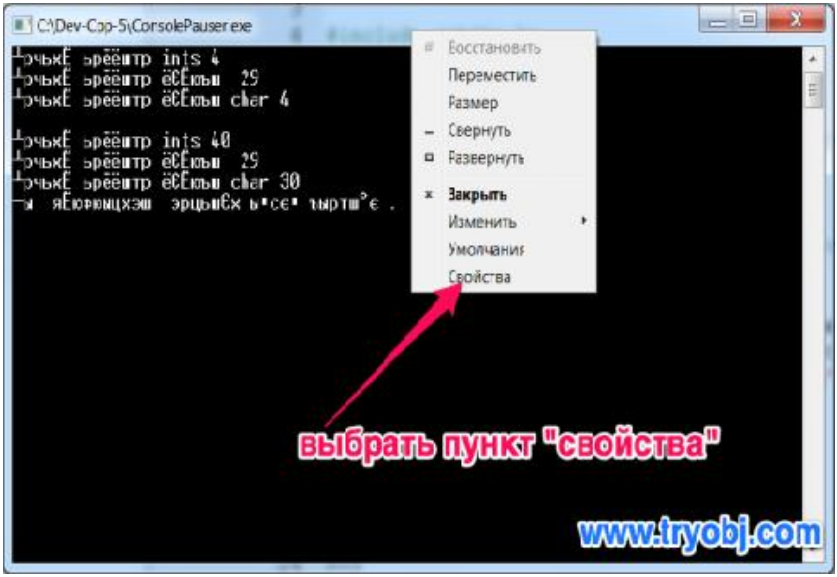


Рис.10. Окно результатов

В открывшемся окне (рис.11) переходим на вкладку "**Шрифт**" и выбираем "**LucidaConsole**".Одновременно можно еще и увеличить размер шрифта для отображения текста в консольном окне.

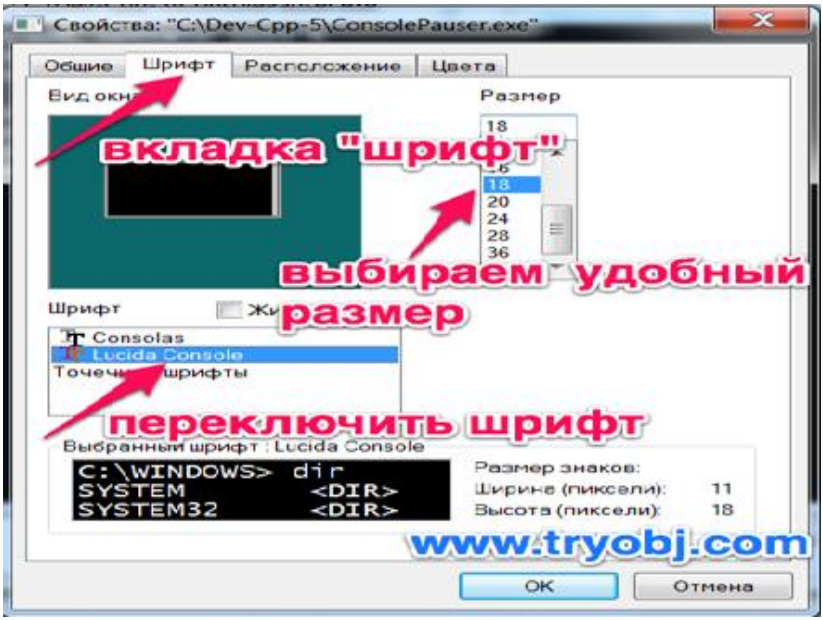


Рис. 11. Выбор шрифта

Переходим на вкладку – "Цвета". Здесь мы можем изменить цвет фона в консольном окне и цвет шрифта. Если вы предпочитаете белый шрифт на черном фоне, то ничего трогать здесь не нужно. Для лучшего чтения лучше выбрать стандартное отображение текста на белом фоне. Как это сделать показано на рис.12.

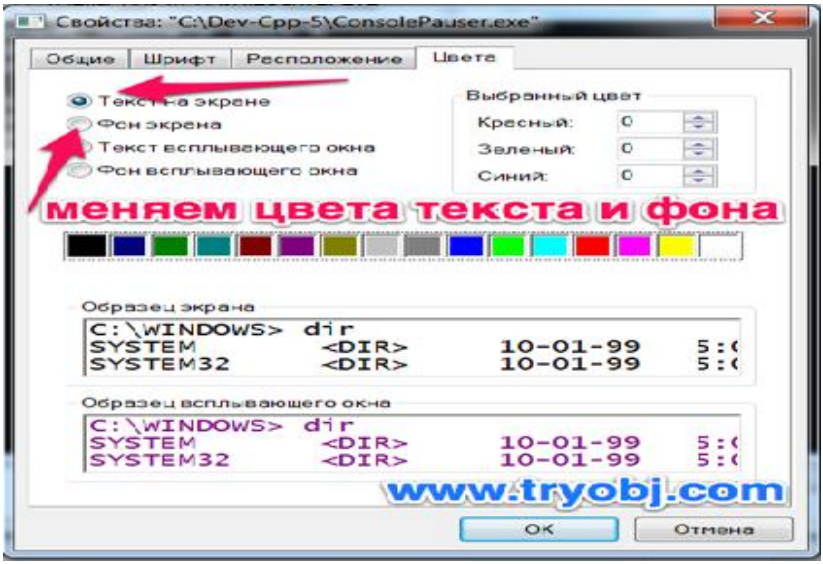


Рис.12.Выбор цвета фона

После внесенных изменений наши программы будут отображаться в окне консоли в виде, показанном на рис. 13.

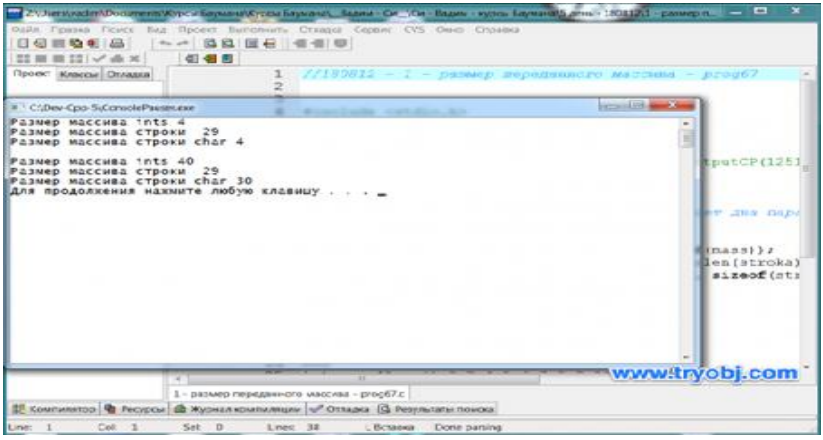


Рис.13. Инверсное изображение окна

Для отладки программы иногда целесообразно пронумеровать строки и выбрать размер шрифта. Это можно сделать в окне, (рис. 14) следующим образом. Выбрать компонент «**Сервис**» и в открывшемся окне последовательно выбрать: «Параметры редактора»\»Вид шрифта»\активировать окно «Номер строки».

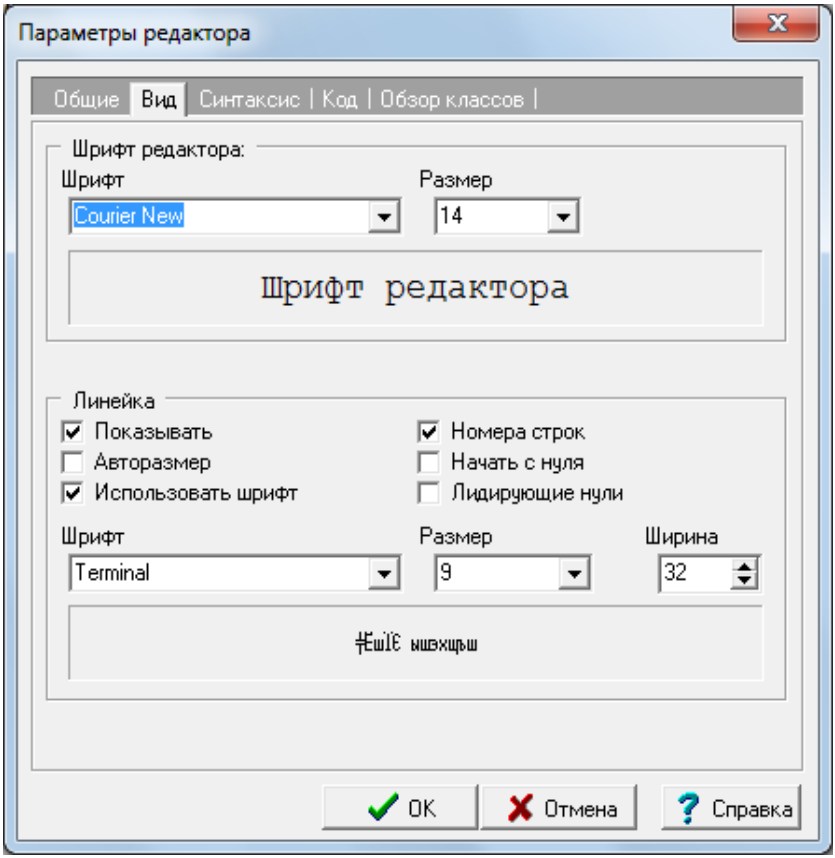


Рис.14. Выбор параметров шрифта

После настройки пакета можно приступить к программированию исходной задачи.

Запуск среды Borland C++Builder и технология работы в ней

Среда Borland C++Builder ориентирована на так называемую "быструю разработку"[5]. В основе систем быстрой разработки или RAD–систем (Rapid Application Development – среда быстрой разработки *приложений*) лежит технология визуального проектирования и событийного программирования, суть которой заключается в том, что среда разработки берет на себя большую
часть рутинной работы по включению в программу базовых элементов программы (например, включение заголовочных файлов, объявление функций), оставляя программисту работу по конструированию диалоговых окон и созданию функций обработки событий, что ускоряет процесс разработки программ в несколько раз. Одной из широко используемых RAD–систем является Borland C++ Builder, которая позволяет создавать различные программы: от простейших однооконных приложений до программ управления распределенными базами данных. В качестве языка программирования в среде Borland C++ Builder используются языки Си и C++.

Процесс создания программы в C++ Builder состоит из двух шагов: сначала нужно создать форму программы (диалоговое окно), а затем – функции обработки *событий.* Форма *приложения* создается путем добавления в нее *компонентов* и последующей их настройки.

В форме практически любого приложения есть компоненты, которые обеспечивают интерфейс (взаимодействие) между программой и пользователем. Такие компоненты называют базовыми. К базовым компонентам можно отнести:

- Label — поле вывода текста;
- Edit — поле редактирования текста;
- Button — командная кнопка;
- CheckBox — независимая кнопка выбора;
- RadioButton — зависимая кнопка выбора;
- ListBox — список выбора;

• ComboBox — комбинированный список выбора.

Основную работу в программе выполняют функции обработки *событий*. Исходную информацию в программу можно ввести из: полей редактирования (компонент Edit), списка выбора (компонент ListBox) или комбинированного списка (компонент ComboBox). Для ввода значений логического типа можно использовать компоненты CheckBox и RadioButton.

Результат программа может вывести в поле вывода текста (компонент Label) или в окно сообщения (функции ShowMessage, MessageDlg).

Расчетные данные вводятся в виде текста в поле редактирования (Edit). Для преобразования текста, находящегося в поле редактирования, в целое число нужно использовать функцию StrToInt, а в дробное – функцию StrToFloat. Следует помнить, что при вводе дробных чисел в поле редактирования нужно отделять целую часть от дробной запятой, как в среде Windows. При выводе результата требуется обратное преобразование числа в строку. Для преобразования целого, например, значения переменной, в строку нужно использовать функцию IntToStr, а для преобразования дробного–функцию FloatToStr или FloatToStrF.

Запускается C++Builder обычным образом, т.е. выбором из меню Borland C++ Builder 6 команды C++ Builder 6 или щелчком кнопки мыши на ярлыке C++Builder. Вид экрана после запуска несколько необычен, поскольку вместо одного окна появляется сразу пять. При этом окна могут перекрываться и даже закрывать одно другое, как показано на рис. 15.

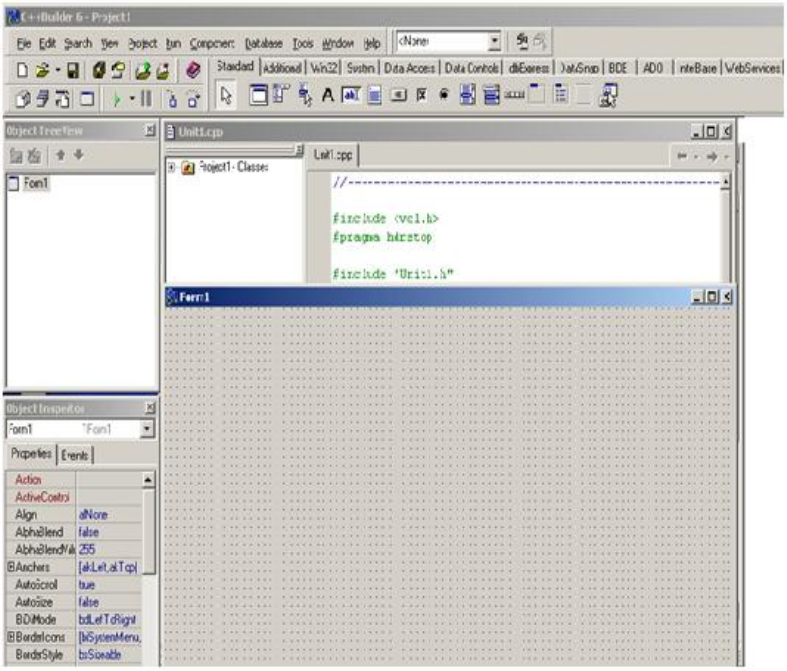


Рис. 15.Основная форма

Запустив C++ Builder, вы увидите заготовку формы будущей программы по центру экрана, за ней Unit1.cpp – окно с листингом (кодом) будущей программы. Сверху экрана расположен список компонентов (объектов), которые можно добавить на форму (Standard, Additional, Win32 и т. п. – это вкладки с различными компонентами). Слева – Object Tree View – дерево (список) компонентов, добавленных в нашу программу. Данный режим работы целесообразно использовать, например, при расчете графиков функций. Когда с помощью компонентов (кнопок) можно управлять вводом данных, запуском на расчет и вывод графика на форму. Более подробные указания на работу в режиме ***Форма*** можно прочитать в [5].

Создание консольного приложения в C++ Builder 6

Консольное приложение удобно тем, что оно позволяет работать с программой без создания формы с использованием библиотек языка C и C++ функций ввода и вывода, языка C (scanf() и printf()). Оно создается следующим образом. Сначала нужно из меню **File** выбрать команду New | **OtherApplication** и на вкладке New появившегося диалогового окна New**Items** щелкнуть на значке **ConsoleWizard**, показанное на рис.16.

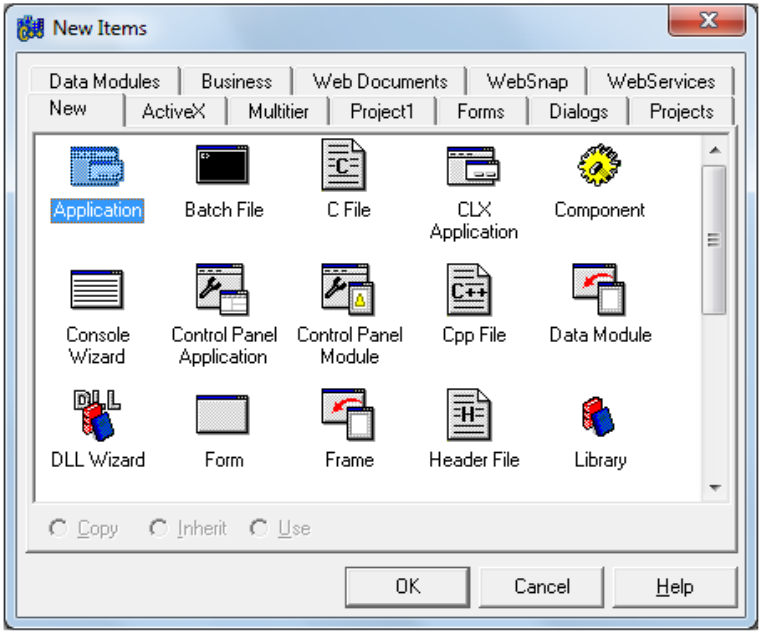


Рис. 16. Задание консольного приложения

В результате этих действий на экране появится окно **Console Wizard**, представленное на рис.17.

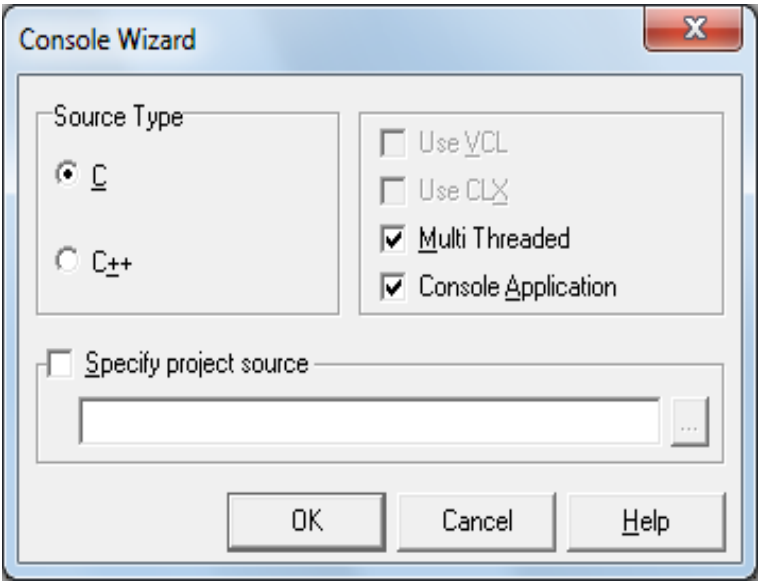


Рис. 17.Задание характеристик консольного приложения

В этом окне можно выбрать язык программирования и указать, будет ли использоваться та или иная библиотека. После того как будут заданы параметры создаваемого консольного приложения, надо щелкнуть на кнопке ОК.

В появившемся поле редактирования (между скобками) можно набрать текст кода программы или же ввести скопированный текст ранее набранной программы в текстовом редакторе.

В результате C++ Builder создаст проект консольного приложения и на экране появится окно редактора кода, показанное на рис.18, в котором находится шаблон консольного приложения – функция main ().

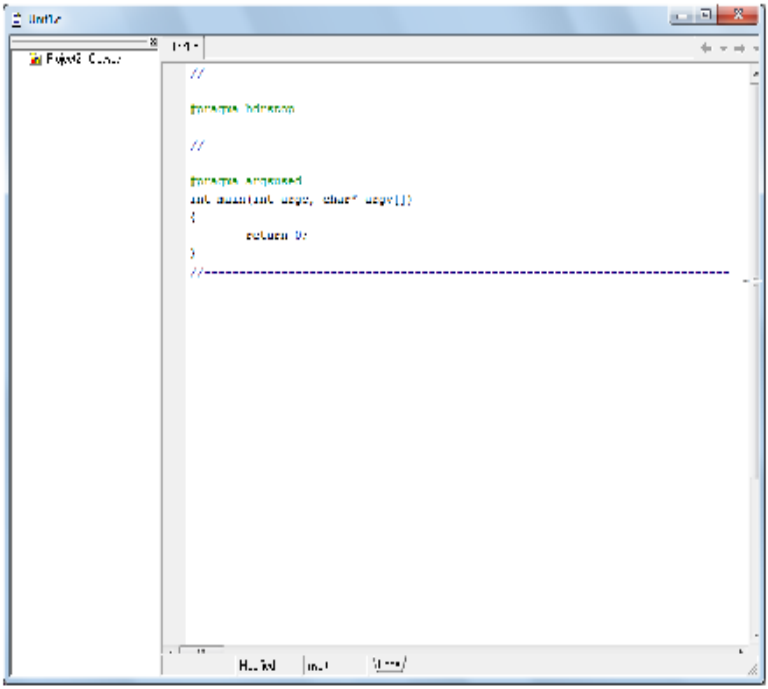


Рис. 18. Окно ввода и редактирования программы

Начинается консольное приложение директивой #pragma hrdstop, которая запрещает выполнение предварительной компиляции подключаемых файлов. После этой директивы надо вставить директивы #include, обеспечивающие подключение необходимых библиотек (например, #include<stdio.h>). Директива #pragma argsused отключает предупреждение компилятора о том, что аргументы, указанные в заголовке функции, не используются.

Следует обратить внимание на то, что консольное приложение разрабатывается в Windows, а выполняется как программа DOS. В DOS и Windows буквы русского алфавита имеют разные коды (в DOS используется кодировка ASCII, а в Windows – ANSI). Это приводит к тому, что консольное

приложение вместо сообщений на русском языке выводит "абракадабру".

Проблему вывода сообщений на русском языке консольными приложениями можно решить, разработав функцию перекодировки ANSI–строки в строку ASCII. Если эту функцию назвать rus, то инструкция вывода сообщения может выглядеть, например, так: printf(rus("Скорость: %3.2f км/час"), v);

Пример программы для консольного приложения с использованием функции rus() приведен в приложении А.

Компиляция консольного приложения выполняется обычным образом, т. е. выбором из меню **Project** команды **Compile.** После успешной компиляции программа может быть запущена выбором из меню **Run** команды Run**.** При запуске консольного приложения на экране появляется стандартное окно командной строки. Процесс сохранения проекта консольного приложения стандартный. В результате выбора из меню **File** команды **Save Project** на экране сначала появляется окно **Save Project,** в котором нужно ввести имя проекта, а затем окно **Save Utit,** в котором надо задать имя модуля.

Получить доступ к модулю консольного приложения (тексту программы) для того, чтобы внести изменения в программу, несколько сложнее. Сначала, выбрав в меню **File** команду **Open Project,** нужно открыть файл проекта. Затем надо открыть окно **Project Manager** (команда **View | Project Manager),** раскрыть список файлов, выбрать cpp –файл и из контекстного меню выбрать команду **Open** (или сделать двойной щелчок на имени cpp–файла).

Раздел 1. Программирование на языке Си

Работа №1

Базовые операции языка Си

Цель работы. Приобретение навыков программирования линейных процессов. Освоить функции ввода/вывода данных, оператора присваивания, инкрементных операций.

Краткие сведения и инструкции

Программы с линейной структурой являются простейшими и используются, как правило, для реализации простых вычислений по формулам. Программа представляет собой последовательную запись инструкций, выполняемых одна за другой.

Алгоритм программы с линейной структурой может быть представлен в виде схемы, показанной на рис.1.1

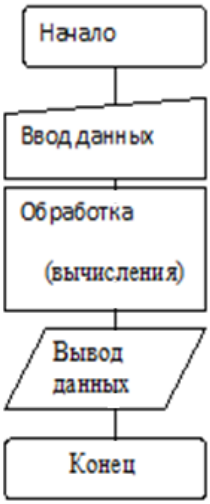


Рис. 1.1

При составлении программы, в первую очередь, необходимо определить исходные данные и объявить типы переменных. Объявления типов переменных обычно помещают в начале программы, вслед за ее заголовком, снабжая инструкцию объявления кратким комментарием о назначении переменной. Инструкция объявления записывается так:

Тип Имя_Переменной;

Объявление переменной можно совместить с ее инициализацией. В этом случае объявлений переменной записывают следующим способом:

Тип Имя_Переменной= Начальное_значение;

В последнем выражении знак "=" обозначает инструкцию присваивания. Она предназначена для изменения значения переменных, в том числе и для вычислений по «формуле».

Пример 1.Объявления переменных:

```
float f = 1.25 ;          //Объявление вещественной переменной
double D = 9.42478 ;      //Объявление переменной двойной точности
unsigned short X = 123 ;  //Объявление без знаковой короткой переменной
char A  = 'a' ;           //Объявление символьной переменной
const char *TRUE = "true" ; //Объявление символьной константной переменной
char* SAYING = "Hello!" ;  //Объявление символьного литерала (строки)
```

В правой части за // записаны комментарии.

Для обработки данных используют арифметические операции

Операция	Знак	Использование	Пояснение
Сложение	+	j = i + 2;	Устанавливает j, равным i плюс 2
Вычитание	–	j = i–5;	Устанавливает j, равным i минус 5
Умножение	*	y = x * 2;	Устанавливает y, равным x, умноженное на 2

Деление	/	y =x / 2;	Устанавливает y, равным x деленное на 2
Деление по модулю	%	y =x % 7;	Устанавливает y, равным остатку от деления x на 7
Инкремент	++	++ i;	Устанавливает i, равным i плюс 1
		i ++ ;	
Декремент	– –	– –i;	Устанавливает i, равным i минус 1
		i – –	

В отличие от других языков программирования в Си и С++ инструкция присваивания, выполняющая некоторое действие, может быть записана несколькими способами. Например, вместо $x = x + dx$ можно записать $x + = dx$, а вместо $i = i+1$ воспользоваться оператором инкремента и записать i++.

Вывод информации и сообщений на экран монитора обеспечивает функция printf(). Первым параметром этой функции является строка вывода, определяющая выводимый текст и формат вывода значений переменных, имена которых указаны в качестве остальных параметров функции.

Для ввода исходных данных с клавиатуры предназначена функция scanf(), первым параметром которой является управляющая строка, остальные параметры – адреса переменных, значения которых были введены. (Использование имени переменной, а не ее адреса в качестве параметра функции scanf() является типичной ошибкой начинающих программистов).

Для иллюстрации действия указанных операций рассмотрим следующий пример.

Пример 2. Введем программу, печатающую на экране: “Сейчас 2019 год”.

```
#include<stdio.h>
/* пример к лаб. работе №1*/
main ( ) // Начало тела главной функции
 int  year, month;  // Объявление переменных
year = 2019;       //Инициализация переменных
```

```
      month =10;
      printf ("Сейчас % d год %d месяц\n",
year, month);
      }     // Конец тела главной функции
```

Рассмотрим эту программу. Первая строка `#include<stdio.h>` подключает стандартный файл ввода/вывода языка Си. Это так называемый заголовочный файл (header files). В файле `stdio.h` находится информация о стандартной функции вывода `printf ()`, которую мы используем.

Вторая строка /*Пример…*/ – комментарий. Строка main() – определяет имя функции. Строка: { содержит открывающую скобку, обозначающую начало тела функции main(). Строка: `int year, month`; объявляет (декларирует) переменные `year` и `month` и сообщает компилятору, что они целые (`int`).

Строка: `year` = 2017; является оператором присваивания.

Строка: printf ("Сейчас `% d` год `%d` месяц `\n", year, month`); является вызовом стандартной функции `printf()`, которая выводит на экран некоторую информацию. Она состоит из двух частей: имени функции `printf()` и двух её аргументов "сейчас `%d` год `%d` месяц `\n"` и `year, month` разделённых запятой. Первый аргумент называется управляющей строкой (Control String). Она может содержать любые символы или спецификации формата, начинающиеся с символа %. Спецификация `%d` указывает, что будет выводиться целое число. Комбинация символов `"\n"` сообщает функции `printf()` о необходимости перевода каретки на новую строку.

Последняя строка программы "}" содержит закрывающую фигурную скобку тела функции main().

Если программа не содержит ошибок, то после компиляции и выполнения её на экране появится сообщение: Сейчас 2019 год. Если была допущена ошибка, то после компиляции, строка, в которой эта ошибка была допущена, будет подсвечена. Переход к предыдущей ошибке

осуществляется комбинациями клавиш ALT+F7, а к последующей ALT+F8.

Усовершенствуем *пример 2*. Введем строковую переменную, например, «Имя» и инициализируем ее.

```
#include<stdio.h>
/* пример к лаб. работе №1*/
main ( )
{
int year, month;
year = 2019;
char *name="Ivan"; //указатель на строку
printf ("Сейчас % dгод\n", year);
printf ("Имя %s\n", name);
}
```

Для вывода новой переменной в printf() введен спецификатор вывода %s.

*Пример 3.*Требуется вычислить длину окружности.

Для вычисления нам потребуется ввести данные с клавиатуры, и мы будем использовать библиотечную функцию scanf(), которая позволит вводить данные во время выполнения программы.

```
#include<stdio.h>
/*Вычисление длины окружности */
main ()
{
int radius;
float length;
printf("Введите значение радиуса :\n");
scanf ("%d", &radius);
length = 3.1415*2*radius;
printf("Радиус-%d\n  длина  -  %f\n",  radius,
length);
}
```

Отличие этого примера от предыдущего заключается в том, что:

1) объявлены две переменные разных типов: `radius` – типа целое (`int`); `length` – типа с плавающей точкой (`float`);

2) используется функция `scanf()` для ввода с клавиатуры значения радиуса окружности.

Первый аргумент функции `scanf()` –“%d” указывает, что будет вводиться целое десятичное число, второй аргумент указывает адрес переменной, которой будет присвоено введённое значение.

В языке Си имеются специальные унарные и бинарные операторы, из которых наиболее хорошо известны положительное приращение (++) и отрицательное приращение (– –), позволяющие с помощью единственного оператора добавить 1 или вычесть 1 из любого значения. Сложение и вычитание допускаются в середине выражения. Кроме того, можно задавать операции приращения до, и после вычисления самого выражения. Например, запись

```
sum = a + b++;
sum = ++a + b;
```

означает следующее: первый оператор суммирует *a* и *b*, затем результат присваивается sum, и потом значение *b* увеличивается на 1; во втором выражении *a* увеличивается на 1, затем суммируются *a* и *b*, и далее результат присваивается *sum*.

Пример 4. Префиксная и постфиксная формы записи инкремента и декремента

```
int  i = 2 ;
printf ( "%d \n", ++i ) ;     // выводится 3
printf ( "%d \n",   i++ ) ;  // выводится 3
printf ( "%d \n", i ) ;       // выводится 4
printf ( "%d \n",  i- - ) ;  // выводится 4
printf ( "%d \n", i ) ;       // выводится 3
printf ("%d \n",- - i ) ;     // выводится 2
```

Задание на выполнение работы

1) Ввести программу примера №2, провести её компиляцию и выполнение.

2) Ввести программу примера №3, провести отладку и запустить на выполнение. Объявить символьную константную переменную, ввести и вывести ее на экран.

3) Модифицировать программу примера №3 таким образом, чтобы она вычисляла длину окружности и площадь круга для любых радиусов.

4) Составьте программу для вывода всех переменных, записанных в *примере* 1.

5) Ввести программу и проверить действие инкрементных операторов присваивания, используя круглые скобки:

```
#include <stdio.h>
#include <conio.h>
main ( )
{
int a = 5, b = 6, sum;
sum = a + b; printf (”a=%d  b=%d sum=%d\n”
, a, b, sum);
sum =a + + +b;printf(”a=%d  b=%d sum=%d\n”
, a, b, sum);
sum  =++a  +  b;  printf  ((”a=%d     b=%d
sum=%d\n” ,a,b, sum);
sum  =  -  -a  +  b;  printf((”a=%d     b=%d
sum=%d\n”,a,b, sum);
sum  =  a-  -  +b;  printf((”a=%d     b=%d
sum=%d\n”,a, b, sum);
sum = a + b;printf ((”a=%d  b=%d sum=%d\n”
, a, b, sum);
getch();
}
```

Отчёт должен содержать

1) Тексты выполняемых программ.
2) Результаты расчётов.

Контрольные вопросы

1. Какие стандартные функции содержатся в файлах stdio.h и conio.h ?
2.Что означает «инкремент»?
3. Что означает «декремент»?
4. Что означает операция присваивания?

Работа №2

Организация ветвящихся процессов: оператор if

Цель работы. Приобретение навыков программирования ветвящихся процессов.

Программа – это описание средствами языка программирования алгоритма решения задачи. Любой алгоритм можно разделить на следования, ветвления и циклы. В простейшем случае имеют место только следования.

Если имеет место проверка условий и выбор одного из возможных направлений продолжения вычислений, то имеет место ветвление, Пример ветвления представлен на рис.2.1.

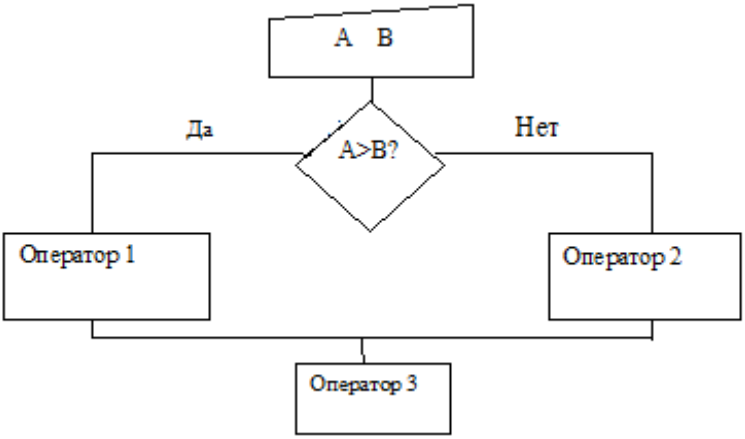


Рис.2.1

Ветвления в программах осуществляются с помощью оператора условия **if.** Этот оператор используется в двух формах:

а) в полной форме:

`if` (выражение) (`if` – если)

оператор1;

`else` (`else` – иначе)

оператор2;

Правило выполнения: Выполнение оператора `if` начинается с вычисления выражения. Если выражение в скобках (здесь некоторое условие) не равно нулю, т.е. “истина”, то выполняется оператор1. В противном случае выполняется оператор 2.

б) в сокращенной форме:

`if` (выражение)

оператор1;

Если выражение в скобках (…) не равно нулю, то выполняется оператор1, в противном случае управление передается следующему оператору программы (оператор3). Для проверки истинности выражения используют операции отношения, приведенные в таблице 2.1.

Таблица 2.1– Операции отношения

Название	Знак	Пояснение
Равенство	= =	Истина, если операнды равны, иначе – ложь
Неравенство	!=	Истина, если операнды не равны, иначе –ложь
Меньше	<	Истина, если левый операнд меньше правого, иначе – ложь
Больше	>	Истина, если левый операнд больше правого, иначе – ложь
Меньше или равно	<=	Истина, если левый операнд меньше или равен правому, иначе – ложь
Больше или равно	>=	Истина, если левый операнд больше или равен правому, иначе – ложь

Пример 1:

```
if (i< j) i++;
   else
       {
          j = i-3; i++;
        }
```

Этот пример иллюстрирует также и тот факт, что на месте «оператор1», так же как и на месте «оператор2» могут находиться сложные конструкции. Если за условием ставятся два и более операторов, то они заключаются в фигурные скобки. Допускается использование вложенных операторов `if`. Оператор `if` может быть включен в конструкцию `if` или в конструкцию `else` другого оператора `if`. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах `if`, используя фигурные скобки и отступы. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово `else` с наиболее близким к нему `if`, для которого нет `else`.

Пример 2.

```
int main ( )
     {
       int t=2, b=7, r=3;
         if (t > b)
            {
               if (b < r)  r=b;
             }
                 else r = t;
            return  0;
       }
```

В результате выполнения этой программы `r` станет равным 2.

Часто выражения в операторе `if` бывают составными, т.е. проверяется два или более условий. В этом случае их объединяют в виде связанной цепочки условий с помощью логических операторов **И** (&&) и **ИЛИ** (||).

Пример 3.

```
int main( )
  {
    int a, b, c;
      if ( a == x )
        b = a;
          if ( a != x  && a ==c)
              b=0;
                 if ( a != x  || a!=c)
                    b=1;
  return  0;
  }
```

Задание на выполнение работы

Составить алгоритм и его программную реализацию для сравнения двух математических выражений из таблицы 2.2. Считать, что выражения равны (или не равны) если их разность не превышает 0,001. Параметры тригонометрических функций ввести в градусах или радианах. Для работы с тригонометрическими функциями подключите директиву `#include <math.h>`.

Таблица 2.2 – Варианты расчетных выражений[4]

Ноиер	Формула 1	Формула 2
1	$Z1 = 2\sin^2(3\pi - 2\alpha)\ \cos^2(5\pi+2\alpha)$	$Z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$
2	$Z1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$	$Z_2 = 2\sqrt{2}\cos\alpha\sin\left(\frac{\pi}{4} + 2\alpha\right)$
3	$Z1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$	$Z_2 = 2\ \sin\alpha$
4	$Z1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha}$	$Z_2 = \mathrm{tg}3\alpha$
5	$Z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$	$Z_2 = \cos^2\alpha + \cos^4\alpha$

6	$Z_1 = \cos\alpha + \cos2\alpha + \cos6\alpha + \cos7\alpha$	$Z_2 = 4\cos\frac{\alpha}{2}\cos\frac{5}{2}\alpha\cos 4\alpha$
7	$Z_1 = \cos^2(\frac{3}{8}\pi - \frac{\alpha}{2}) - \cos^2(\frac{11}{8}\pi - \frac{\alpha}{4})$	$Z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$
8	$Z_1 = \cos^4\alpha + \sin^2\beta + \frac{1}{4}\sin^2 2\alpha - 1$	$Z_2 = \sin(\alpha+\beta)\sin(\alpha-\beta)$
9	$Z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$	$Z_2 = -4\sin^2(\frac{\alpha-\beta}{2})\cos(\alpha+\beta)$
10	$Z_1 = \frac{\sin(3\alpha+\frac{\pi}{2})}{1-\sin(3\alpha-\pi)}$	$Z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi - \frac{3}{2}\alpha\right)$
11	$Z_1 = \frac{1-2\sin^2\alpha)}{1-\sin(3\alpha-\pi)}$	$Z_2 = \frac{1-\operatorname{tg}\alpha}{1+\operatorname{tg}\alpha}$
12	$Z_1 = \frac{\sin4\alpha}{1+\cos4\alpha}\frac{\cos2\alpha}{1+\cos2\alpha}$	$Z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right)$
13	$Z_1 = \frac{\sin\alpha + \cos(2\beta-\alpha)}{\cos\alpha - \sin(2\beta-\alpha)}$	$Z_2 = \frac{1+\sin2\beta}{\cos2\beta}$
14	$Z_1 = \frac{\sin\alpha + \cos\alpha}{\cos\alpha - \sin\alpha}$	$Z_2 = \operatorname{tg}2\alpha + \sec2\alpha$
15	$Z_1 = \frac{\sqrt{2b+2\sqrt{bb-4}}}{bb-4+b+2}$, $bb=b^2$	$Z_2 = \left(\sqrt{b+2}\right)^2$
16	$Z_1 = \frac{x^2+2x-3+(x+1)\sqrt{x^2-9}}{x^2-2x-3+(x-1)\sqrt{x^2-9}}$	$Z_2 = \sqrt{\frac{x+3}{x-3}}$
17	$Z_1 = \frac{\sqrt{(3m+2)^2-24m}}{3\sqrt{m}-\frac{2}{\sqrt{m}}}$	$Z_2 = -\sqrt{m}$
18	$Z_1 = \left(\frac{a+2}{\sqrt{2a}} - \frac{a}{\sqrt{2a}+2} + \frac{2}{a-\sqrt{2a}}\right)$	$Z_2 = \frac{1}{\sqrt{a}+\sqrt{2}}$

Содержание отчета

1. Краткое содержание цели и задачи применения ветвящихся процессов вычислений.
2. Алгоритм вычисления заданного преподавателем варианта.
3. Таблицу переменных и трассировочную таблицу
4 Программу вычислений.
5. Распечатку результатов.

Контрольные вопросы

1. Для какой цели используются ветвящиеся процессы?
2. Назовите основные операторы ветвящихся процессов
3. В каких случаях используется конструкция **if…. else?**

Работа № 3

Организация циклических вычислений

Цель работы: Изучить формы изменения хода выполнения программы по условию или без него; формирование циклических процессов вычислений.

Операторы и конструкции организации циклов

Для программирования вычислительных процессов с известным числом повторений обычно используют оператор цикла. Циклы необходимы, когда повторяются одни и те же вычисления до тех пор, пока выполняется некоторое условие. В языке Си известно три вида операторов цикла: **`for, while, do- while.`**

Основная форма цикла `for` имеет следующий вид:

`for` (*инициализация; проверка условия; изменение*) *оператор;*

На самом деле в общем виде:

`for` (*выражение_1; выражение_2; выражение_3*) *оператор;*

В простейшем виде инициализация используется для присвоения начального значения параметру цикла. Проверка условия – обычное условное выражение, определяющее, когда цикл будет завершен. Изменение (приращение) обычно используется для изменения параметра цикла каждый раз при повторении цикла. Как только условие становится ложным, начинает выполняться следующий за циклом оператор.

Простейший пример оператора `for`:

1) `for( i= 0; i <10; i++) printf (“%d \n”, i);`

В результате выполнения этого оператора будут напечатаны цифры от 0 до 9 в столбик.

2) `for(i=9; i>=0; i- -) printf( “%d \n”, i);`

Будут напечатаны цифры от 9 до 0.

В качестве параметра цикла не обязательно использовать целочисленный счетчик. Приведем фрагмент программы, выводящий на экран буквы русского алфавита:

```
unsigned char ch;
for (ch='A'; ch<= 'Я' ; ch++) printf("%c", ch);
```

Следующий фрагмент программы

```
for (ch='o'; ch!='N' ; ch++) scanf ("%c", &ch);
```

будет выполняться до тех пор, пока с клавиатуры не будет введен символ ‘N’.

Заметим, что место, где должно стоять приращение, может оказаться пустым. Случайно или намеренно может получиться цикл, из которого нет выхода, это так называемый бесконечный цикл.

Приведем три примера таких циклов.

```
for( ; ;) printf(" Бесконечный цикл\n" );
for(i=1;1; i++) printf(" Бесконечный цикл \n");
for(i=10;i>6;i++)printf(" Бесконечный цикл \n");
```

Для избегания “зацикливания“ программы используется оператор `break`. Если оператор `break` встречается в составном операторе цикла, то происходит немедленное прекращение выполнения цикла и начинается выполнение следующего оператора программы.

Пример 1.

```
for( ;  ;)
{ ch = getchar();   /* Прочитать символ */
if (ch == 'Q')  break;   /* Проверка символа */
printf ( "%c", ch);  /* Печатать символ */  }
```

В этом цикле будут печататься введенные символы до тех пор, пока не будет введен символ ‘Q’.

Следующий оператор цикла – это цикл `while`. Основная его форма

`while` (*условие*) *оператор;*

где оператор может быть простым, составным или пустым оператором. *Условие,* как и во всех других операторах, является просто выражением. Цикл выполняется до тех пор, пока условие

принимает значение *истинно*. Когда же условие принимает значение *ложно*, программа передает управление следующему оператору программы. Так же как и в цикле `for` в цикле `while` сначала проверяется условие, а затем выполняется оператор. Это так называемый цикл с *предусловием*.

В отличие от двух предыдущих циклов в операторе цикла *do….while* условие проверяется в конце оператора цикла. Форма этого оператора следующая

do{

последовательность операторов

} while (*условие*);

Фигурные скобки не обязательны, если внутри них находится один оператор, но для лучшей читаемости их лучше ставить.

Оператор цикла *do–while* называется оператором цикла с *постусловием*.

Какое бы условие не стояло в конце оператора, набор операторов в фигурных скобках один (первый) раз выполняется обязательно. А в циклах `for` и `while` оператор может не выполниться ни разу.

Вложенные циклы. Когда один цикл находится внутри другого, то говорят, что это вложенные циклы. Часто такие циклы встречаются при заполнении таблиц, перемножении матриц и т.д.

Пример 2. Составить программу печати таблицы умножения целых чисел.

```
# include<stdio.h>
/*Пример  перемножения чисел от 1 до 10*/
  int main()
 {
   int  i, j;
   for( i=1; i<10 ; i++)
     {
       for(j=1; j<5; j++)
       printf ("% d*%d = %2d",i, j, i*j );
       printf ('\n');
     }
  }
```

Прерывание выполнения циклов может быть выполнено также с помощью двух следующих операторов: `continue` и `goto`. Если оператор `continue` встретился в операторе цикла, то он передает управление на начало следующей итерации цикла. В циклах `while` и do… `while` – на проверку условия, в цикле `for`–на приращение.

Для использования оператора `goto` надо ввести метку (label). Метка- это идентификатор, за которым следует двоеточие.

Пример 3. Использование оператора `continue`.

```
#include<stdio.h>
int main()
{   int i;
   for (i=1; i<100; i++)
    {
      if (i%7) continue;
      printf("%8d", i);
    }
}
```

Эта программа печатает натуральные числа, кратные семи.

Пример 4. Использование оператора `goto`.

```
for()
{while() {
for() {
.................................
goto exit; } } }
exit:  printf ("Быстрый выход из вложенных циклов");
```

Пример 5. Вычислить конечную сумму

$$S = \sum_{k=m}^{N} \ln(kx) / k^2$$

Общее слагаемое суммы выражается формулой
*ak = ln(kx)/k*k.*
Программа имеет следующий вид:

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

```
# include <math.h> /*Подключение математической библиотеки */
/*Вычисление суммы */
int main()
{
float x, sum;  /*Описание типов переменных */
int k, n, k1;
printf (“Введите число x\n”);
scanf (“%f”,&x);
printf (“Введите число n\n”);
scanf (“%d”,&n);
sum=0;
  for(k=1; k<=n; k++)
  { k1=k*k;
    sum+=log(k*x)/k1;
    printf(“k=%d сумма=%f\n”,k,sum);
  }
}
```

Задание на выполнение работы

1. Ввести программу примера 5, осуществить ее отладку и работу.

2. Составить программу и произвести вычисления для одного из следующих вариантов[4].

1) $s = \sum_{k=1}^{N} (x+1)k\cos(x/k)/(k+1)!$

2) $s = \sum_{k=1}^{N} (\sin x)^k / k!$

3) $s = \sum_{k=1}^{N} (\cos x)^k / k!$

4) $s = \sum_{k=1}^{N} (arctgx)^k /(k+2)!$

5) $s = \sum_{i=2}^{N} (e^x - i) \ /(i+1)!$

6) $s = \sum_{i=1}^{N} (e^{x/i} + e^{-x/i})/(i+2)!$

7) $S = \sum_{k=1}^{N} (\ln(x * i))/(i+2)$

8) $s = \sum_{k=1}^{N} (x)^{k+5} /(k+7)!$

9) $p = \prod_{i=1}^{N} (x^i + \frac{1}{\sqrt[4]{x}})$

10) $P = \prod_{i=1}^{N} (x+i)^{i+1} /(i-2)!$

11) $P = \prod_{i=1}^{N} (x^i + \frac{1}{arctg(x)})$, *x* –вещественное число

12) $S = \sum_{i=1}^{N} \prod_{j=1}^{i} \frac{j!}{i!}$, *N*<=5

13) $S = \sum_{k=m}^{N} k^2 \ln(k)$, $N>m$

14) $S = \sum_{i=1}^{N} \frac{i!}{(a+b)^i}$,

15) $F = \sum_{i=1}^{n} \frac{\cos(x)^i}{i!}$

16) $S = \sum_{i=1}^{N} \sum_{j=1}^{i} \sin(0{,}1 * i + 0{,}2 * j)$; N=4; S=6,212;

17) $S = \sum_{i=1}^{N} \sum_{k=0}^{i} (i+k)^2$,

18) $s = \sum_{k=1}^{N} \frac{(-1)^{k+1}}{k(k+1)}$,

19) $S = \sum_{i=1}^{N} \frac{x_i}{1+y_i}$ при x_i = (1+i), y = 1/i;

20) $s = \sum_{i=1}^{N} (\sin x)^i \; / \sum_{i=1}^{N} \sin x^i$;

Примечание: При вычислении факториалов необходимо помнить о максимально возможном целом числе, которое можно поместить в разрядной сетке.

Содержание отчета

1. Краткое содержание цели и задачи применения циклических процессов вычислений.
2. Алгоритм вычисления заданного преподавателем математического уравнения.
3. Программу вычислений.
4. Распечатку результатов.

Контрольные вопросы

1.Назовите основные операторы циклических процессов.
2. Назовите основные параметры цикла.
3. Как образуется бесконечный цикл и как выйти из него?

Работа №4

Условные операторы и операторы выбора

Цель работы. Изучение трех форм управления процессом выполнения программ:

1) выполнение последовательности операторов;

2) выполнение определенной последовательности операторов до тех пор, пока некоторое условие истинно;

3) использование проверки истинности условия для выбора между различными, возможными способами действия.

Основное задание

1. Составить программу решения квадратного уравнения вида:

$$AX^2 + BX + C = 0$$

с полным анализом возможных решений (дискриминант $D<0$, $D=0$, $D>0$) на основе конструкций `if` и `if-else`.

2. Разработать диалоговую программу, позволяющую получать решения квадратного уравнения (1) при различных значениях коэффициентов *A, B, C*, а также выхода из программы по запросу, используя конструкции while или do….while (по выбору).

3. Разработать диалоговую программу, позволяющую в зависимости от значений коэффициентов получать то или иное решение, используя оператор выбора `switch()`:

а) если $A=0$; B и C не равны нулю;

б) если $B=0$; A и C не равны нулю;

в) если $C=0$; B и A не равны нулю;

г) другие возможные сочетания коэффициентов *A, B, C*.

Программа должна вычислять как действительные, так и комплексные корни.

4. Вывести на экран сведения об авторе, исходные значения коэффициентов, значение дискриминанта и результаты решения.

5. Разработать алгоритм и программу вычисления функции F на интервале от $X_{нач.}$ до $X_{коч.}$ с шагом dx. Результаты вычислений вывести в виде таблицы значений функции. Варианты заданий приведены в приложении Б.

Рекомендации по программированию

1. При выполнении п.1 основного задания необходимо использовать:

– файл заголовка `math.h`, который позволит Вам вычислить корень квадратный из дискриминанта (`sqrt(D));`

– конструкцию вида: `if` (выражение) оператор, используемый, если выражение истинно.

Пример 1*:*

```
// подразумевается, что комплексных корней нет
if(D<0)
printf(“Решения нет \n”);
```

При необходимости в комбинации с `if` можно использовать ключевое слово `else`, позволяющее выполнить альтернативный оператор, или блок операторов, если условие неистинно.

Пример 2:

```
if (D<0)
printf (“Решения нет \n”)
else
printf (“Решение есть \n”);
```

Операторы `if` и `else` могут быть вложенными.

2. При выполнении п. 2 основного задания необходимо использовать дополнительно к предыдущей программе конструкцию вида:

`while` *(выражение) оператор.*

Ключевое слово `while` позволяет выполнять оператор или блок до тех пор, пока условие не перестанет быть истинным. Оператор или тело блока, связанного с `while`, не будет выполнять, если выражение изначально ложно.

Пример 3:

```
char vych='d';
          :
          :
    while(vych= ='d')
{
     :
     :        // группа операторов
  printf("Продолжать решение? (d/n)\n");
  cscanf("%c",&vych);
}
```

В этом фрагменте программы блок команд будет выполняться до тех пор, пока символьной переменной не будет присвоено значение 'd'.

3. При выполнении п. 3 основного задания необходимо использовать дополнительно к предыдущей программе конструкцию вида:

```
switch ( выражение_1)
    {
case константа 1: оператор или группа операторов
    блока 1;
break;
                                  :
                                  :
case константа 2: оператор или группа операторов
    блока 2;
break;
default: оператор или группа операторов;
    }
```

При выполнении оператора switch сначала вычисляется значение *выражения_1*, стоящего в скобках оператора switch. Тип значения должен быть одним из целых: *char, int, unsigned*

int, *long int* и *long unsigned*. Вычисленное значение сравнивается со значениями констант операторов case.

При совпадении значения *выражения_1* с *i*–й константой выполняется оператор или группа операторов *i*–го блока. Затем управление передается наследующий (после switch) оператор, если в *i*–й ветви присутствует оператор break.

Если значение *выражения_1* не совпало ни с одной из констант, выполняется оператор или группа операторов, помеченных default. При ее отсутствии выполняется следующий после switch оператор.

Пример 4:

```
int  var;
:
:
if (A1==0)
var=1;

else
var=2;
   :
   :
switch(var)

   {

  case 1: {
printf (“Решение уравнения \n”); …
     x1 = ……;
     x2 = ……;
       break;
         }
 case 2:  printf (“Решение  уравнения \n”);
     x1 = ……;
           :
                :
      break;
 default: puts (“ Ошибка \n”);
 }
```

Если значение $D<0$ (корни комплексные), то предусмотрите изменение знака D перед вычислением квадратного корня, иначе будет зафиксирована ошибка.

4. Выведите результаты расчетов и распечатайте программу.

Содержание отчета

1. Постановка задачи.
2. Формализация задачи.
3. Структурные схемы алгоритмов решения задачи.
4. Распечатки текстов программ с комментариями.
5. Ответы на контрольные вопросы.
6. Выводы по работе.

Контрольные вопросы

1. Какие управляющие средства выполнения программ существуют в языке Си? Объясните их синтаксис.
2. Объясните назначение всех функций, использованных в программах.
3. Какие операции используются в управляющих средствах выполнения программ.
4. Объясните различия между циклами while и do … while.
5. Для каких целей используется оператор switch ?

Работа №5

Обработка одномерных массивов

Цель работы. Изучить принципы ввода, инициализации и обработки одномерных массивов с использованием различных управляющих структур (`if, while, do… while, for`) при работе с массивами данных.

Основное задание

Составить программу обработки одномерного массива заданного типа произвольной длины по одному из вариантов представленному в таблице. Предусмотреть два типа ввода: ручной с клавиатуры и автоматический с помощью датчика случайных чисел. Для генерации псевдослучайных чисел в качестве элементов массива воспользуйтесь функцией `rand()`. Для ее инициализации включите заголовочные файлы `<time.h>` и `<stdlib.h>`. Инициализация случайными числами выполняется по фрагменту, приведенному в приложении А.

```
srand((unsigned)time(&t) ) ; // инициализацияГСЧ
for(i=0; i<= n; i++)
a[i] = rand();
```

В случае, если автоматически вводимые числа не удовлетворяют требованиям задания, следует их откорректировать или ввести вручную.

Рекомендации по программированию

В начале программы предусмотрите объявление массива, например mas[*N*], где *N* не более 20 и не менее 10.

Затем определите характер заполнения массива: автоматическое или ручное. Это можно сделать с помощью операторов:

```
printf ( “ Определите характер заполнения: 1– ручное, 2– автоматическое”);
cscanf ("%d", &w);
```
Далее используйте конструкцию if else :
```
if(w==1)
{
for(j=0; j<N; j++)
{
printf (“ Введите элемент массива mas[j]”);
scanf("%f", &mas [j]);
}
}
else if(w==2)
x [ j ] =1.2+rand()%10;
```
или *switch*(*w*), где *w* равно 1 или 2.

Воспользуйтесь указаниями работы №4.

Таблица– Варианты заданий[4]

№ варианта	Задание
1.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) сумму элементов массива; 2) произведение элементов массива, расположенных между максимальным и минимальными элементами.
2.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) сумму положительных элементов массива; 2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.
3.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) произведение элементов массива с четными номерами; 2) сумму элементов массива, расположенных

	между первым и последним нулевыми элементами.
4.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) сумму элементов массива с нечетными номерами; 2) сумму элементов массива, расположенных между первым и последним отрицательными элементами.
5.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) максимальный элемент массива; 2) сумму элементов массива, расположенных до последнего положительного элемента.
6.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) минимальный элемент массива; 2) сумму элементов массива, расположенных между первым и последним положительными элементами.
7.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) номер максимального элемента массива; 2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами.
8.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) номер минимального элемента массива; 2) сумму элементов массива, расположенных между первым и вторым отрицательными элементами.
9.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) максимальный по модулю элемент массива.

	2) сумму элементов массива, расположенных между первым и вторым положительными элементами.
10.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) максимальный по модулю элемент массива; 2) сумму элементов массива, расположенных между первым и вторым положительными элементами.
11.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) номер минимального по модулю элемента массива; 2) сумму модулей элементов массива, расположенных после первого отрицательного элемента.
12.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) номер максимального по модулю элемента массива; 2) сумму элементов массива, расположенных после первого положительного элемента.
13.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) количество элементов массива, лежащих в диапазоне от A до B; 2) сумму элементов массива, расположенных после максимального элемента.
14.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) количество элементов массива, равных 0; 2) сумму элементов массива, расположенных после минимального элемента.
15.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить:

	1) количество элементов массива, больших C; 2) произведение элементов массива расположенных после максимального по мо дулю элемента.
16.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) количество отрицательных элементов массива; 2) сумму модулей элементов массива, расположенных после минимального по модулю элемента.
17.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) количество положительных элементов массива; 2) сумму элементов массива, расположенных после последнего элемента, равного нулю.
18.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) количество элементов массива, меньших C; 2) сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.
19.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) произведение отрицательных элементов массива; 2) сумму положительных элементов массива, расположенных до максимального элемента.
20.	В одномерном массиве, состоящем из *n* вещественных элементов, вычислить: 1) произведение положительных элементов массива; 2) сумму элементов массива, расположенных после минимального элемента.

Содержание отчета

1. Постановка задачи.
2. Схема алгоритмов.
3. Распечатки текстов программы с комментариями.
4. Ответы на контрольные вопросы.
5. Вывод по работе.

Контрольные вопросы

1. С какой целью используется индексация массивов?
2. С какой целью используется тип данных?
3. Опишите примененный Вами алгоритм обработки массива.
4. Как определить адрес *i* –го элемента в памяти машины**?**

Работа № 6

Обработка двумерных массивов

Цель работы: изучить принципы работы различных управляющих структур (`while, do… while, for`) при работе с многомерными массивами данных.

Основное задание

Составить программу работы с матрицей, для этого: организовать ввод элементов `a[ i ][ j ]` где `i` – номер строки, `j` – номер столбца, N –размер матрицы (N<6); ввод матрицы осуществить двумя вариантами: вручную с клавиатуры и с помощью датчика случайных чисел.

Выполнить ввод, вывод и преобразование матрицы одним из следующих вариантов.

1. Транспонировать матрицу.
2. Возвести матрицу в квадрат.
3. Умножить *i*–ю строку исходной матрицы на сумму элементов главной диагонали.
4. Прибавить к *j*– му столбцу *i*– й столбец, умноженный на заданное число.
5. Прибавить к *i* – й строке *k* – ю строку.
6. Поменять местами два заданных столбца.
7. Поменять местами две заданные строки.
8. Поменять местами заданный столбец и заданную строку.
9. Сосчитать сумму элементов матрицы.
10. Сосчитать сумму элементов каждой строки матрицы.
11. Сосчитать сумму элементов каждого столбца матрицы.
12. Рассчитать среднее значение элементов матрицы.
13. Заменить знаки элементов матрицы на противоположные.
14. Сосчитать количество отрицательных элементов матрицы.

15. Рассчитать средние значения отрицательных и положительных элементов матрицы.

16. Сосчитать сумму элементов матрицы, лежащих ниже главной диагонали.

17. Сосчитать сумму элементов матрицы, лежащих выше главной диагонали.

18. Найти обратную матрицу.

19. Вычислить определитель матрицы.

20. Найти след матрицы.

Организовать вывод на экран дисплея сведения об авторе, номер варианта, результаты выполнения работы.

Рекомендации по программированию

1. В начале программы предусмотрите ввод значений количества строк, столбцов матрицы $M[N][N]$.

Пример:

```
m1: cprintf ("Введи значение количества строк и столбцов матрицы \n");
cscanf ("%d", &N);
```

В случае, когда $N<2$ или $N>6$, вывести сообщение об этом и повторить ввод N.

Пример:

```
if (N<2||N>6)
     {
      cprintf ("Неправильно введено значение N \n");
       goto m1;
     }
```

где оператор `goto m1` обозначает безусловный переход к оператору с меткой m1.

Затем определите характер заполнения матрицы: автоматическое или ручное. Это можно сделать с помощью операторов:

printf (“ Определите характер заполнения: 1– ручное, 2– автоматическое”); и

`switch(w)`, где w равно 1 или 2. Для генерации псевдослучайных чисел в качестве элементов матрицы

воспользуйтесь функцией `rand()`, инициализированной директивой препроцессора.

2. Для ввода элементов матрицы *M*[*N*][*N*] следует использовать конструкцию вида:

```
for ( i=0; i<N; i++)
for(j=0; j<N; j++)
scanf("%f", &M[i][j]);
```

В операторе цикла можно опустить одно или более выражений (но нельзя опустить символы точка с запятой). Необходимо только включить в тело цикла несколько операторов, которые, в конце концов, приведут к завершению его работы.

Пример 1:

```
ans=2;
for(n=3; ans<=15;)
     {
      ans = ans*n;
      cprintf ("+ans=%f"\n", ans);
     }
```

3. Для создания диалогового режима в программе можно применить конструкции операторов цикла: `while; do - while`. Операторы `while` и `do – while` рассмотрены в предыдущей работе.

Пример 2:

```
      char answer;  int N;
          do
            {
/*группа операторов */
             printf (“Хотите завершить задачу ?”);
            answer= getchar();
            }
        while ( answer != ‘Y’ || answer
!= ‘y’);
        printf  (“Сделано %d – раз вычислений
\n”,N);
```

В этом случае выход из цикла будет осуществлен при вводе с клавиатуры (операторы getch()) букв *Y* или *y*.

4. Для выполнения п.3 основного задания воспользуйтесь частью результатов, полученных в работах 4 и 5.

Содержание отчета

1.Постановка задачи.
2.Схема алгоритмов.
3.Распечатки текстов программы с комментариями.
4.Ответы на контрольные вопросы.
5.Вывод по работе.

Контрольные вопросы

1. Назовите операторы цикла, использованные Вами в программе.
2. Назовите операторы выхода из цикла и поясните их назначение.
3. Назовите условия возникновения бесконечного цикла.
4. Как образуется цикл с предусловием?
5. Что общего между циклом `while` и `for`.
6. Опишите примененный Вами алгоритм преобразования матрицы.

Работа №7

Функции пользователя и динамическое распределение памяти

Цель работы. Выработка навыков разработки функций и компоновки программы из нескольких функций (файлов) на примерах решения типовых задач линейной алгебры (операций с массивами). Знакомство с методами обработки динамических массивов.

Основное задание

Составить программу, состоящую из функций, реализующих операции над матрицами произвольного размера, представленных в виде динамических массивов, в соответствии с вашим вариантом задания. Для этого необходимо:

1. Познакомиться с функциями `input()` и `output()`, предназначенных для ввода и вывода соответственно элементов матриц произвольного размера.
2. Разработать функции, осуществляющие набор матричных операций в соответствии с вашим вариантом.
3. Дополнить функцию `main()` функцией `menu()` для селективного вызова сформированных Вами функций обработки. Создать диалоговый, интерактивный режим работы программы (см. пункт 3 работы № 6).

Рекомендации по программированию

Рассмотрим пример функции, которая вводит с клавиатуры матрицу размером m x n : m – количество строк, n – количество столбцов, выделяет необходимую область памяти для размещения ее элементов и осуществляет ввод их значений в память ЭВМ.

```
int ** input(int n, int m)
{
int i, j;
```

```
      int  **a;
      /*Выделение динамической памяти для  элементов
матрицы*/
      a= (int**)malloc(m*sizeof (int*));
      for(i=0; i<m; i++)
       {
         a[i]=(int*)malloc(n* sizeof(int*));
         for(j=0;j<n;j++)
         {
          a[i][j]=0;  //Обнуление ячеек памяти
         }
       }
      for(i=0; i<m; i++)
         for(j=0;j<n; j++)
         {
     printf("\n  Введите элемент  матрицы A(%d,%d) "
,i+1,j+1);
       scanf("%d",&a[i][j]);
      }
    return a;
   }
```

Для динамического выделения свободной памяти в данном фрагменте программы используется функция `malloc()`. Она возвращает указатель типа void. Для правильного использования значение этой функции надо преобразовать к указателю на соответствующий тип. При успешном выполнении операции `malloc()` возвращает указатель на первый байт свободной памяти требуемого размера. Если достаточного количества памяти нет, то возвращается значение 0 (нулевой указатель). Чтобы определить количество байт, необходимых для хранения переменной, используют операцию `sizeof`.

Для освобождения динамической памяти используют функцию `free` с прототипом void `*free(void *p)`; где *p – указатель на первый байт выделенной памяти. Прототипы обеих функций находятся в заголовочном файле `stdlib.h`.

Динамическое распределение памяти удобно тогда, когда заранее неизвестно количество используемых переменных (в нашем случае - это *m* и *n*).

Для выделения динамической памяти можно так же использовать функцию

new тип [*количество элементов*].

Рассмотрим теперь функцию, осуществляющую вывод матрицы на экран дисплея.

```
void output (int **z,  int m,  int n)
{
int  i, j;
printf ("\n Результирующая  матрица \n");
for(i=0; i<m; i++)
      {
      for(j=0; j<m; j++)
      printf("%8d",  z[i][j] ) ;

      printf("\n");
      }
}
```

Для того, чтобы начать выполнение работы составьте главную функцию main(), в которую вызовите описанные выше функции и функции обработки матриц в соответствии с вашим заданием. В начало программы нужно поместить директивы и декларации.

Примерный вид функции main().

```
#include<stdio.h>
#include<conio.h>
#include< stdlib.h >
//Объявление шаблонов функций обработки
int **input (int, int);
int  fproi (int**, int**, int,  int);
..... // другие функции
void  ouput(int**, int, int);

int  main(void)
{
```

```
int m,n;
int **p, **q;
puts(" Введите размер исходной матрицы");
printf("число строк=");
scanf("%d",&m);
printf("число столбцов=");
scanf("%d",&n);
//Вызов функций
p= input (m,n);
output (p,m,n);
q= input (m,n);
output (p,m,n);
fproi(p,q,m,n);
……..
}
```

При составлении программы воспользуйтесь разработками предыдущей работы № 6.

Варианты заданий

В вариантах заданий использованы следующие обозначения:

1. *T,U,S*
2. *MQB*
3. *S,M,T* *T*– транспонирование матрицы.
4. *B,U,C* *C*– вычисление следа матрицы.
5. *Q,S,N* *U*–умножение матрицы на число.
6. *C,Q,T* *M*– перемножение матриц.
7. *U,T,N* *Q*–возведение матрицы в квадрат.
8. *S,N,U* *S*– сложение матриц
9. *C,N,T* *B*– вычитание матриц.
10. *M,N,Q* *N*– вычисление нормы матрицы.
11. *C,N,T*
12. *B,N,M*
13. *C,S,T*
14. *Q,C,B*
15. Сравнить суммы элементов главных диагоналей. Переставить местами два столбца.

16. Сравнить суммы элементов двух столбцов. Переставить местами две строки.

17. Сравнить суммы элементов двух строк. Найти наибольшую сумму строк.

18. Проверить: является ли матрица магическим квадратом? Найти наибольший элемент в матрице.

Содержание отчета

1. Постановка задачи.
2. Схемы алгоритмов функций.
3. Распечатки текстов программы с комментариями.

Контрольные вопросы

1. В чем состоит отличие объявление функции от ее определения?
2. Как объявить функцию с переменным числом параметров?
3. Какие типы объектов могут быть использованы в качестве формальных параметров?
4. Что такое локальные объекты? Какова их область видимости и «время жизни»?
5. В чем состоит отличие автоматических переменных от статических?
6. Объясните механизм передачи параметров по значению, по указателю, по ссылке.
7. Какой массив называется динамическим?
8. Объясните назначение двойных указателей.

Работа № 8

Организация работы с файлами

Цель работы. Овладеть навыками работы с потоками данных, находящихся на внешних устройствах, функциями обработки текста.

Организация работы с файлами

Язык Си, кроме стандартного ввода данных с клавиатуры и вывода результатов на экран, предоставляет также возможность обмена данными с файлами, находящимися на диске. В Си не предусмотрены никакие предопределенные структуры файлов (такие как файлы последовательного или прямого доступа); все файлы рассматриваются как последовательные*, потоки битов*.

Для файла определен *маркер* (указатель чтения/ записи). Он определяет текущую позицию, к которой осуществляется доступ. С началом работы любой программы автоматически открываются некоторые стандартные потоки, например, стандартный ввод (его имя –***stdin***) и стандартный вывод (его имя –*stdout*). По умолчанию они связаны с клавиатурой и экраном соответственно. Поэтому в тех функциях ввода/вывода, которые использовались до сих пор, не указывалось, из какого потока берутся или куда помещаются данные: это известно по умолчанию.

Однако, в своей программе возможно открыть и другие потоки, связывать их либо с файлами на диске, либо с физическими устройствами (например, с принтером), записывать в них или считывать из них информацию. Для этого служат функции ввода/вывода *верхнего уровня*. Доступ к потоку осуществляется с помощью *указателя*. Указатель на файл описывается следующим образом:

```
FILE *fp;
```

Тип `FILE` – это структура, определенная в <*stdio*> с помощью средства *typedef* и содержащая некоторую информацию о файле: например, флаги состояния файла, размер буфера, указатель на буфер и др. Описанный указатель можно связать с конкретным файлом в момент открытия данного файла. Это осуществляется с помощью функции

fopen(*"путь к файлу","тип доступа"*);

которая возвращает указатель на файл или *NULL* в случае ошибки.

Например, в результате выполнения оператора

```
fp = fopen("ex1.txt","w");
```

файл `ex1.txt` будет открыт для записи, а в программе на этот файл можно сослаться с помощью указателя `fp`, т.е. функция `fopen()` берет внешнее представление файла – его физическое имя – и ставит ему в соответствие внутреннее логическое имя, которое далее будет использоваться в программе). В качестве типа доступа могут быть указаны следующие параметры:

"*w*" –существующий файл открывается для записи, если файл не существовал, он создается заново, если существовал, то старое содержимое стирается.

"r" – файл открывается для чтения с начала.

"*a*" – файл открывается для дозаписи в конец файла.

(О других типах доступа можно прочитать в пособии по языку Си[1]).

По окончании работы с файлом он должен быть закрыт. Для этого используется функция

fclose (*указатель_файла*).

Для чтения /записи данных в файл имеются функции, аналогичные уже известным функциям ввода/вывода

fprintf(), fscanf(), fputs(), fgets(), getc(), putc(),getc(), ffputc().

Функции *getc()/ fgetc(), putc()/fputc()* по своим действиям идентичны, отличие состоит только в том, что *getc()* и *putc()* реализованы как макроопределения, а *fgetc()* и *fputc()*– как настоящие функции.

Прототипы всех файловых функций, а также необходимые константы находятся в файле <*stdio.h*>.

Функция `fgets(str,50,fp)` имеет три параметра. Второй параметр – количество *N* считываемых символов, включая '\0'. Эта функция прекращает работу после чтения *N*– 1 символа, либо после чтения '\0'. Функция `fgets()` возвращает либо адрес прочитанной строки, либо NULL по исчерпании файла или в случае ошибки.

В случае исчерпания файла или ошибки функция *getc()* возвращает EOF (End of FILE) – конец файла.

Функция `fputs ()` возвращает код последнего прочитанного символа, если все в порядке, и EOF в случае ошибки. Эта функция не переводит строку автоматически.

Чтобы продемонстрировать работу с этими функциями, рассмотрим простой пример.

*Пример 1.*Форматированный ввод/вывод в файл числа и массива строк.

```
# include<stdio.h>
# include<conio.h>
#include<locale.h>
int main()
{
 setlocale(0,"rus");
 int n;
char str1[3][10];
FILE *fp;
 fp=fopen("D:\\ex.txt","w+a");
puts("Введите число") ;
scanf("%d",&n);
fprintf(fp, "%d\n" , n);//Запись числа в файл
for(int i=0;i<3;i++)
{
```

```
printf("Введите строку %d \n",i+1);
for(int j=0;j<10;j++)
{
scanf("%c", &str1[i][j]);
}
}
for(int i=0;i<3;i++)
{
for(int j=0;j<10;j++)
fprintf ( fp,"%c",str1[i][j]); //Запись строки
//в файл
}
fclose ( fp);
printf("Распечатка содержимого файла\n");
if(( fp= fopen ("D:\\ex.txt","r")) != NULL)
 {
fscanf (fp, "%d" , &n);
printf(  "n= %d\n" , n);
for(int i=0;i<3;i++)
{
 for(int j=0;j<10;j++)
fscanf(fp,"%c", &str1[i][j]);
}
for(int i=0;i<3;i++)
{
 for(int j=0;j<10;j++)
printf ("%c",str1[i][j]);
}
fclose ( fp );
  }
else printf ("\n Нельзя открыть файл для
чтения !");
getch();
  }
```

Все приведенные выше функции обрабатывали файл последовательно символ за символом. Язык Си предоставляет возможность работать с файлом как с массивом:

непосредственно достигать любого определенного байта. Для позиционирования файла служит функция.

fseek (указатель на файл, смещение позиции, начальная точка).

```
int fseek(FILE *stream, long int offset, int whence);
```

Второй аргумент имеет тип *long.* Его значение может быть больше и меньше нуля. Он показывает, как далеко (в байтах) следует продвинуться от начальной точки. Третий аргумент является кодом, определяющим положение начальной точки в файле. Установлены следующие значения кода: 0- начало файла, 1- текущая позиция, 2- конец файла. В случае успеха функция *fseek()* возвращает 0, а если была ошибка, то возвращается –1.

Пример 2. Неформатированный ввод и вывод строк через указатель.

```
       int main()
{
    setlocale(0,"rus");
int i;
char            *str1[]=            {"строка1",
"строка2","строка3"};
FILE *fp;
    // заполняем текстовый файл
fp=fopen("D:\\ee4.txt","wt");
for(i=0;i<3;i++)
fputs(str1[i],fp);
puts("распечатка содержимого файла");
 fp= fopen ("D:\\ee4.txt","r");
   for(i=0;i<3;i++)
   {
      fgets(str1[i],60,fp);
      puts(str1[i]);
   }
 fclose ( fp );
 getch();
}
```

Задание на выполнение работы

1. Создать текстовый файл в директории группы.

2. Введите программу *примера 1*. Выполните действия с заполнением и считыванием символов и строк. Убедитесь, что созданный файл не пуст.

3. Составить программу обработки текстового файла в соответствии с вариантом задания, представленного в таблице. Для формирования текстового файла используйте рассмотренные примеры. Для выполнения задания используйте функции заголовочных файлов `ctype.h` и `string.h`.

Таблица. Варианты обработки текстового файла

Номер варианта	Вид обработки
1	1.1 В текстовом файле подсчитать количество строк, которые оканчиваются заданной буквой (задается преподавателем). 1.2 Подсчитать количество слов в тексте 1.3 Посчитать количество символов нижнего регистра
2	2.1 В текстовом файле подсчитать количество строк, которые начинаются заданной буквой (задается преподавателем). 2.2 Посчитать количество строк в заданном тексте. 2.3 Подсчитать количество пробелов в тексте.
3	3.1 Найти максимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину. 3.2 Подсчитать количество печатных символов, отличных от пробела 3.3 Подсчитать количество пустых строк в файле.
4	4.1 Найти минимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину. 4.2 Подсчитать количество строк, начинающихся с определенной буквы латинского алфавита.

	4.3 Подсчитать количество символов в строке с минимальной длиной.
5	5.1 Подсчитать количество строк, начинающихся с букв верхнего регистра. 5.2 Подсчитать количество печатных символов, отличных от пробела 5.3 Подсчитать количество пустых строк в файле.
6	6.1 Найти длину строки, лежащей в заданном пределе. 6.2.Соединить две строки в одну. 6.3.Подсчитать количество печатных символов, отличных от цифр
7	7.1 Найти строку, начинающуюся с заданной буквы. 7.2 Сравнить две строки 7.3 Подсчитать количество слов в заданной строке.
8	8.1 Найти строку, оканчивающуюся на заданную букву. 8.2. Подсчитать количество слов в заданной строке. 8.3 К первой строке присоединить *n* символов другой строки.
9	9.1 Подсчитать количество строк, начинающихся с букв нижнего регистра. 9.2. Найти самую длинную строку в тексте. 9.3. Подсчитать количество слов в тексте.
10	10.1 Найти минимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину. 10.2 Найти две идентичные строки 10.3 Подсчитать количество символов, отличных от перевода строки.
11	11.1 Найти максимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину 11.1 Подсчитать количество пробелов в тексте 11.3 Подсчитать количество слов в тексте
12	12.1 В текстовом файле подсчитать количество строк, которые начинаются заданной буквой

	(задается преподавателем). 12.2 Посчитать количество строк в заданном тексте. 12.3 Подсчитать количество пробелов в тексте.
13	13.1 В текстовом файле подсчитать количество строк, которые имеющих одинаковую длину. 13.2 Посчитать количество слов в заданном тексте. 13.3 Подсчитать количество строчных букв в тексте.
14	14.1 Найти строку, оканчивающуюся на заданную букву. 14.2. Подсчитать количество слов в заданной строке. 14.3 К первой строке присоединить *n* символов другой строки.
15	15.1 Найти строку, начинающуюся с заданной буквы. 15.2 Найти строку идентичной заданной 15.3 Подсчитать количество слов в тексте.

Содержание отчета

1. Краткие теоретические сведения о методах обработки потоков данных.
2. Программа обработки, распечатки файлов с данными.
3. Комментарии к программе и полученным результатам

Контрольные вопросы

1. Что такое поток данных?
2. В чем состоит различие ввода данных в стандартном потоке и с внешнего устройства?
3. Напишите команды открытия файла для чтения и записи, закрытия файла.
4. В чем состоит различие команд *fputs()* и *fgets()*?

Работа № 9

Программирование графиков функций

Цель работы. Изучить методы построения графиков функций в графическом редакторе языка Си. Составить программу для расчета и вывода в графическом режиме заданной функции. Результаты расчета выводятся в графической форме на экран и принтер.

Исходные данные: Математическая функция, диапазоны изменения параметров функции.

Пример построения графика

Напишем программу, которая выводит на экран точечный график функции $y=0{,}5x^2+4x-3$. Диапазон изменения аргумента: от –15 до 5; шаг аргумента 0,1.

График вывести на фоне оцифрованной координатной сетки. Начало координат должно находиться в центре экрана. Предусмотреть вывод графика в виде точечной кривой и сплошной линии какого-либо цвета. Оси координат вывести в виде сплошной линии цвета фона. Линии координатной сетки вывести тонкими точечными линиями.

Программу целесообразно составить из двух функций: функции `grid()`, выводящей на экран оцифрованную координатную сетку, и функции `grafik()`, строящей график функции.

```
      void grid()
        {
      int x0,y0;  //начало координат
int   dx,dy;  //шаг  координатной  сетки  (  в
пикселах)
int h,w;     //высота и ширина области вывода
int x,y;
float lx,ly;  // метки линий сетки по X и  Y
```

```
float dlx,dly; // шаг меток линий сетки по X и
Y
char  st[8];       // изображение метки линии
сетки
      x0=50; y0=400; // начало осей координат
dx=40; dy=40; //шаг приращения по осям
dlx=0.5;
dly=1;
   h=300;
   w=400;
lx=0;
ly=0;
line (x0,y0,xo,y0-h); //ось  X
line (x0,y0,xo+w,y0); //ось Y
/*засечки, сетка и оцифровка по оси X*/
x=x0;
do
  {
      //засечка
setlinestyle(SOLID_LINE,0,1); //вид линии
line (x,y0-3,x,y0+3);
//оцифровка
sprintf(st,"%2.1f",lx);
outtextxy(x-8,y0+5,st);
      lx+=dlx;
        //линия сетки
      setlinestyle(DOTTED_LINE,0,1);
      line (x,y0-3,x,y0-h);
                  x+=dx;
}
while(x<x0+w);
  /*засечки, сетка и оцифровка по оси Y*/
y=y0;
do
{
   //засечка
setlinestyle(SOLID_LINE,0,1);
line (x0-3,y,x0+3,y);
```

```
//оцифровка
sprintf(st,"%2.1f",ly);
outtextxy(x0-40,y,st);
ly+=dly;
   //линия сетки
setlinestyle(DOTTED_LINE,0,1);
     line (x0+3,y,x0+w,y);
setlinestyle(SOLID_LINE,0,1);
y- =dy;
                    }
while(y>y0-h);
}   // конец  программы grid().
```

Построим точечный график для функции $y=0{,}5x^2+4x-3$.

```
    void grafik()
  {
  float x,dx;
  float x1,x2;
  float y;
  int mx, my;
  int x0,y0;
  int px,py;
  x0=320;y0=240;
  mx=20;my=20;
  line(10,y0,630,y0);//ось   x,строится,   если
нет сетки
  line(x0,10,x0,470);//ось   y,строится,   если
нет сетки
  x1=-25;
  x2=5;
  dx=0.1;
  x=x1;
  moveto(x,0);
  while(x<x2)
  {
      y=0.5*x*x+x*4-3;
px=x0+x*mx;
```

```
      py=y0-y*my;
putpixel(px,py,WHITE);//построение графика по
точкам
      setcolor(WHITE);
      //lineto(px,py);   в   случае   сплошного
//графика
      x+=dx;
      }
      }
      Запишем теперь главную функцию.
      #include <graphics.h>
     #include<math.h>
      #include<stdlib.h>
      #include <conio.h>
      #include<stdio.h>
     int main()
      {
      initgraph(); // загрузка графики
      grafik();  // ввод функции графика
      grid();   // ввод функции сетки
      getch();
      }
```

Задание на выполнение работы

Разработать программу построения графика одной из функций. Варианты заданий приведены в таблице.

Таблица. Виды функций

№ варианта	Функция	Диапазон изменения аргумента
1	$y= 1.3x^2-1.8+\log(x)$	[-1.2,1.2]
2	Окружность x=0.5+2cos(t); y=0.2+2sin (t)	$[0, 360^0]$
3	Степенная функция $y=x^3-2x^2+x$	[–1,–3]
4	Эллипс x=3cos(t); y=15sin(t)	$[0, 360^0]$
5	Показательная функция $y=\exp(x^2)$	[–1,1.3]

6	Кардиоида x=4cos(t) (1+cos t) y=4sin(t)(1+cos t)	[0, 2π]
7	Дробно-рациональная функция y= (1.5x +3) /(x-2)	[–4.2, 1.9]
8	Декартов лист x=3at /(1 + t^3) y=3at^2 /(1 + t^3)	[–0.5, 10] a=2
9	Функция синус y=2.5sin(x) + 0.5	[–2π , 2π]
10	Циссоида x=5t^2 /(1 + t^2) y=5t^2 /(1 + t^2) t=tg(f)	[–π /4, π /4]
11	Тригонометрическая функция y=cos(x^2)	[–2π , 2π]
12	Строфоида x=4(t^2-1) /(t^2+1) y=4t(t^2-1) /(t^2+1) t=tg(f)	[–π /2.5, π /2.5]
13	Тригонометрическая функция y=tg(x) – 2x	[–π /2.5, π /2.5]
14	Астроида x=3.5$\cos^3$ (t), y=3.5$\sin^3$ (t)	[0, 2π]
15	Арксинус y=arcsin (0.5x)	X=[–2, 2]
16	Эпициклоида x=(a+b)cos(t) – acos ((a+b)t/a), y=(a+b)sin(t) – asin ((a+b)t/a)	a=6, b=9 t = [π , 2π]
17	Логарифм y=ln(x+2)	–1.5, 5
18	Гопоциклоидах=2acos(f) + acos(2f), y=2asin(f) + acos(2f)	[–2π , +2π] a=1.5
19	Арктангенс y=3 arctg(x)	[–5, 5]
20	Эвольвента окружности x=acos(f) + afsin(f) y=a sin(f) – acos(f)	f=[–0^0,90^0] a=1.5

Содержание отчета

1. Краткие теоретические сведения о графических методах обработки данных.
2. Программа построения графика заданной функции.
3. Комментарии к программе и полученным результатам.
4. Распечатку графика функции.

Контрольные вопросы

1. ля чего включен в программу файл `graphics.h`?
2. Для чего включена в программу строка `initgraph()`

Раздел 2. Программирование на языке С++

Работа №10

Форматирование и вывод данных в С++

Цель работы. Приобретение практических навыков работы с форматами и флагами при выводе данных

Форматы и флаги языка С++

Для использования функций ввода/вывода и других в языке С++ необходимо подключить заголовочный файл iostream, а также указать пространство имен, в котором содержатся эти функции. Простейший вид программы должен содержать следующие строки:

```
#include <iostream>
using namespace std ;
int main ( )
/* тело функции */
{ int x,sum=0;
 cout<<"Введите x->\n"
  cin>>x;
  if ( x <= 0 )
  sum += x ;
  cout << sum << endl ;
 system("pause");
 return 0 ;
}
```

Если в программе используются функции языка Си, то нужно также включить и его заголовочные файлы, например : stdio.h, conio.h. В пространстве имен `namespace std` содержатся две функции: вывода (“*cout*<<”) и ввода (“*cin*>>”). Оператор `endl` аналогичен оператору `\n` в Си. Строка `system(«pause»);` останавливает выполнение программы для просмотра результата.

В отличие от языка Си при выводе расчетных данных с помощью оператора “*cout*<<” формат чисел устанавливается по

умолчанию, и он не всегда может удовлетворять пользователя. Для этой цели используются манипуляторы и флаги форматирования. С помощью флагов и манипуляторов можно установить ширину поля вывода, точность выводимого числа (количество знаков после запятой), установить или отменить вставку символов заменителей в пустые разряды, в какой системе счисления выводятся числа: в десятичной (dec) или в шестнадцатеричной (hex) и др.

Пример использования флагов форматирования:

```
1) long oldfmtflags = cout.flags();
2) cout.flags(ios::scientific|ios::hex);
3) cout<< '\n' <<1.2 << ";(dec) 110=(hex)"
   110;
4) cout.flags(oldfmtflags);
5) cout<< '\n' <<1.2 << ";(dec) 110=(hex)"
   <<110;
```

В строке 1 старые флаги (`oldfmtflags`) с помощью функции `flags()` будут заменены на новые.

В строке 2 устанавливаются флаги для вывода вещественных чисел в научном формате и целых чисел в шестнадцатеричном формате. Оба флага являются членами класса `ios`, на что указывает двойное двоеточие (`::`).

В строке 3 записан вывод числа 1.2 в научном формате и десятичного числа 110 в шестнадцатеричном формате.

В строке 4 снова устанавливаются старые флаги.

В строке 5 записан вывод числа 1.2 в научном формате и десятичного числа 110 в шестнадцатеричном формате по умолчанию. Обратите внимание, что `cout` пишется в строен один раз, тогда как `<<` столько раз, сколько выводится переменных.

В результате выполнения данного фрагмента на экран будет выведено следующее:

```
1.2 00000e+00; (dec) 110 = (hex) 6e

1.2; (dec) 110 = (hex) 110
```

Помимо флагов для управления выводом данных используются манипуляторы. Для их установки необходимо подключить заголовочный файл iomanip.
Пример использования манипуляторов:

```
cout.width(12);  //установка ширины поля вывода (12)

cout.precision(8);  //установка количества знаков после
```
запятой

```
cout.fill('%'); //установка символа заполнителя (%)

cout.setf(ios::showpoint|ios::showpos|ios::lef
t) ;//установка //позиции, с которой производится вывод

cout.setf(ios::internal); //вставка заполнителей между
```
знаком и //модулем числа (часто пробел).

```
Cout<< setw(sizeof(col_name))<<
setprecision(3)<<darray[0] <<endl;
```

Задания для выполнения работы

Задание №1. Дополнить фрагмент программы заголовочными файлами, объяснить назначение используемых манипуляторов. Ввести новые манипуляторы и объяснить, что изменилось при выводе.

```
Int main()

{  for(char capital = 'A', small = 'a';
capital <= 'Z'; capital++, small++)

    {  cout << endl;

   cout.fill('x');

   cout<<"\t"<< capital << hex << setw(10)<<
(int)(capital<< dec << setw(10) <<
(int)(capital)   << " " << small   << hex <<
```

```
setw(10) <<static_cast<int>(small) << dec <<
setw(10) << static_cast<int>(small);

     }

   cout << endl;

   system("pause");
   return 0;
}
```

Задание № 2. Задать два массива, инициализировать их вещественными числами. Поменять их содержимое друг на друга. Для вывода массивов используйте манипуляторы ширины поля вывода и научный формат представления чисел.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Сформированный текстовый файл и результаты.

Контрольные вопросы

1. В каком заголовочном файле содержатся манипуляторы и флаги?
2. С какой целью используется манипулятор `width(12)`?
3. Что определяет флаг `fill('%')`?

Работа №11

Сортировка массивов и списков

Цель работы. Освоить основные методы сортировки массивов и списков.

При работе со списками очень часто возникает необходимость перестановки элементов списка в определенном порядке. Такая задача называется сортировкой списка и для ее решения существуют различные методы. Простейшим из них является пузырьковая сортировка. При обменной сортировке упорядоченный список B' получается из B систематическим обменом пары рядом стоящих элементов, не отвечающих требуемому порядку, пока такие пары существуют. Метод систематического обмена соседних элементов с неправильным порядком при просмотре всего списка слева направо определяет пузырьковую сортировку: максимальные элементы как бы всплывают в конце списка.

Одной из наиболее эффективных является быстрая сортировка. Быстрая сортировка состоит в том, что список B=< K1,K2,…,Kn> реорганизуется в список B'< K1 >B'', где B' – подсписок B с элементами, не большими K1, а B''–подсписок B с элементами, большими K1. В списке B',< K1 >, B''элемент K1 расположен на месте, на котором он должен быть в результирующем отсортированном списке. Обычно K1 выбирается в середине списка. Определим два указателя *b* (begin) и *e* (end) на примере списка на рис.11.1 и рис.11.2.

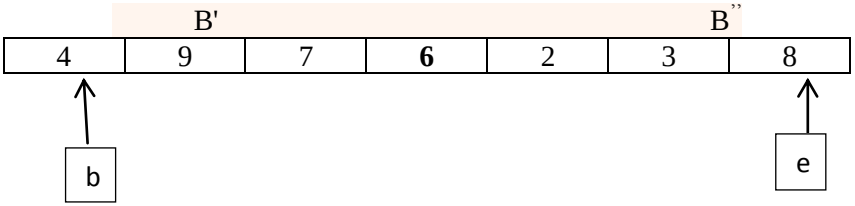


Рис.11.1

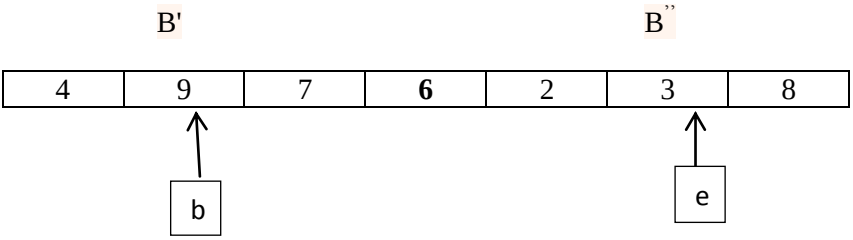


Рис.11.2 Сортируемый список

Указатель *b* указывает на первый элемент подсписка B', а указатель *e* – на последний элемент подсписка B", K1=6. Теперь двигаем наши указатели на встречу друг другу до тех пор пока не встретим число большее (если указатель *b*) или меньшее (указатель *e*) K1.Тогда мы должны поменять эти числа местами. Снова сдвигаем указатели на одну итерацию, повторяя те же действия до того, как только указатели встанут на позицию K1. Далее к спискам B' и B" снова применяется упорядочивание быстрой сортировкой, повторяя описанные действия.

Задания для выполнения работы

Задание №1. Дана программа, в которой использована функция пузырьковой сортировки
`swap(Array[place],Array[index].`

Раскройте ее содержимое и откомпилируйте программу.

```
#include<iostream>
#include<time.h>
 using namespace std;
 void RandArray(int size, int Array[])
    {
       for(int i=0; i<size; i++)
           {
```

```
            Array[i]=rand()%50;
          }
    }
void PrintArray(int size, int Array[])
    {
         for(int i=0; i<size; i++)
          {
             cout<<"M["<<i<<"]=
"<<Array[i]<<endl;
          }
    }
void InsertMinElement(int Array[],int size,int
place)
    {
        int index=place;
        int minElement=Array[place];
                for(int i=place;i<size;i++)
            {
                if(minElement>Array[i])
       {
            index=i;
            minElement=Array[i];
        }
       }
         swap(Array[place],Array[index]);
    }
void SelectSort(int Array[],int size)
    {
        for(int i=0;i<size-1;i++)
            InsertMinElement(Array,size,i);
    }
 int main()
    {   srand(time(0));
       const int size=10;
```

```
        int M[size]={0};
         RandArray(size,M);
        PrintArray(size,M);
                SelectSort(M,size);
                cout<<"otsortirov
massiv"<<endl;
                PrintArray(size,M);
        system("pause");
    }
```

Задание №2. Отсортировать вещественный массив по возрастанию функцией быстрой сортировки.

```
void qs(int* a, int first, int last)
{
    int i = first, j = last, x = a[(first +
last) / 2];

    do {
        while (a[i] < x) i++;
        while (a[j] > x) j--;

        if(i <= j) {
            if (a[i] > a[j]) swap(a[i], a[j]);
            i++;
            j--;
        }
    } while (i <= j);

    if (i < last)
        qs(a, i, last);
    if (first < j)
        qs(a, first, j);
}
```

Вызов функции `qs` для массива из n элементов будет иметь следующий вид.

`qs(a, first, n-1)`; где `a` – сортируемый массив, `first` – номер первого элемента (0), `last` – номер последнего элемента `(n-1)`.

Задание № 3. Найти максимальный (минимальный) элемент в массиве.

Задание № 4. Определить индексы элементов массива, делящихся без остатка на заданное целое число. Отсортировать массив, оставив в нем, только отрицательные элементы.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Сформированный текстовый файл.

Контрольные вопросы

1. Для каких целей используется сортировка?
2. В чем суть пузырьковой сортировки?
3. Чем отличается быстрая сортировка от пузырьковой?
4. Объясните алгоритм работы быстрой сортировки.

Работа № 12

Обработка структур данных

Цель работы. Приобрести практические навыки работы с простыми и вложенными структурами данных.

Структура – это совокупность взаимосвязанных элементов одного, либо разных типов. Взаимосвязь определяет порядок размещения элементов структуры. Элементы структуры объединены под одним именем. Поэтому структура трактуется не как множество отдельных элементов, а как единое целое.

Элементом структуры могут быть переменные базового типа, массивы, указатели, объединения, другие структуры. Однако элементом структуры не может быть структура того же типа, в которой он содержится. В то же время этот элемент может быть указателем на тип структуры, в которую он входит. Это позволяет создавать связанные списки структур. Структуры бывают простые и динамические.

Примеры простых структур: библиографические данные о книге; строка платежной ведомости; данные о некотором товаре; координаты точек на плоскости и т. п. Например, сведения о студенте

```
struct student { char name[25];
                 int  grup;
                 char  age;
                } st1;
```

Доступ к компонентам структуры осуществляется с помощью указания имени структуры и следующего через точку имени выделенного компонента, например:

```
st1.name="Иванов";
st1. grup =25;
st1.age = “2010 г.р.” ;
```

Если в качестве переменной структуры используется указатель, то обращение к компонентам структуры осуществляется символом –>.

Например : `struct student *p;`

```
p->name = "Иванов";
p-> grup = 25;
p->age = “2010 г.р.” ;
```

Можно создать массив структур. В этом случае в качестве переменной используется массив. Например, приведенной ниже структурой

```
struct student { char name[25];
                 int  grup;
                 char  age; }st[10];
```

будет сформирован массив из десяти структур типа `student`.

Задания для выполнения работы

Задание №1. Задать структуру на группу из *n* человек. Ввести список и вывести информацию обо всех введенных студентах.

Задание № 2. Вложенные структуры. Создать массив адресов. Дополнить программу интерфейсом (например, введите фамилию и т.д.). Вывести информацию на экран.

```
#include <iostream>
using namespace std ;
int main ( )
{
// простые структуры
struct Address {
      char city [ 20 ] ; char street [ 30 ] ;
int house ;
} ;
struct Person {
char Fname [ 15 ] ; char Lname [ 20 ] ;
} ;
// иерархическая структура
```

```
 struct Employee {
      Person p ;       // вложенная структура
Address addr ;         // вложенная структура
 } ;
 // объявление переменной указателя
 Employee *p = new Employee ;
 // ввод значений элементов
 cin >> p -> p . Fname ; cin >> p -> p . Lname
;
 cin >> p -> addr . city ;
cin >> p -> addr . street ;
 cin >> p -> addr . house ;
 system("pause");
 return 0 ;
}
```

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Сформированный текстовый файл.

Контрольные вопросы

1. К какому типу переменных относятся структуры?
2. Может ли структура быть переменной другой структуры?
3. Как образовать массив структур?
4. В каком случае используется обращение –>.

Работа № 13

Обработка списков

Цель работы. Освоение методов обработки линейных и дважды связанных списков, приобретение навыков составления и отладки соответствующих программ

Методические указания по обработке списков

Рассмотрим в качестве примера организацию систем учета вкладов в отделении сбербанка. Будем считать, что в простейшем случае информация о вкладе содержит фамилию вкладчика и сумму вклада в рублях. Запишем эту информацию в виде структуры[1]:

```
struct vklad
  {
   char family[12];
   float sum;
  }
```

Пусть стоит задача: разместить вклады в алфавитном порядке вкладчиков. Операции, которые надо произвести в программе, следующие: во – первых, нужно определить: имеется ли вкладчик с данной фамилией (НАЙТИ), во–вторых, включить новый вклад (ВКЛЮЧИТЬ), в – третьих, вычеркнуть вклад из системы (ИСКЛЮЧИТЬ).

Для обработки данных обо всех вкладах можно использовать массив вкладов:

```
strukt vklad array [1000],
```

тогда поиск вклада может быть выполнен быстро, например, при числе вкладов *K* при использовании метода деления пополам максимальное время поиска примерно пропорционально целой части двоичного логарифма *K*.

Однако, операция ВКЛЮЧИТЬ в общем случае требует сдвига части массива. Удаление вклада приводит либо к

незаполненным «окнам», когда информация стирается (например, поле family элемента массива array заполняется пробелами), либо опять требует сдвига части массива. Сдвиг массива – это нежелательное действие, которое занимает время, пропорциональное длине массива *K*.

Разрешить возникшую проблему выполнения операции ВКЛЮЧИТЬ и ИСКЛЮЧИТЬ можно в результате отказа от статического размещения элементов массива (вкладов) и организации динамического списка. Список состоит из элементов, каждый из которых в общем случае является структурой, в которой выделены содержательная и вспомогательная части. Вспомогательная часть используется для организации связей между элементами списка и содержит одно или несколько полей ссылочного типа.

В программе учета вкладов, использующей динамические списки, содержательная часть элементов остается такой же, как указано выше, и к ней добавляется в простейшем случае одно поле для указания следующего элемента списка (sled). Переменные - указатели sled должны быть переменными ссылочного типа. Элементы списка можно представить в виде следующего описания:

```
strukt vklad
{
char family[12];
float sum;
strukt vklad  sled;
}
```

Тогда информацию, например о четырех вкладах, можно представить в виде списка. Переменная top типа `strukt vklad *` указывает на первый элемент списка, а поле sled в последнем элементе имеет значение NULL.

Сформировать такой список можно с помощью программы включающей следующий фрагмент:

```
strukt  vklad *top;

strukt  vklad *p;
```

```
……..
top=NULL;
for(i=0; i<4; i++)
{
p = (strukt vklad*) malloc((sizeof(strukt
vklad)));
      p-> sled=top;
      printf("Введите фамилию вкладчика:")
;
      gets(p->family);
      printf("Введите сумму вклада:");
      scanf("%f",p->sum);
      top=p;
}
```

Рассмотрим поиск элемента списка с заданным значением одного из полей. Пусть, например, необходимо увеличить сумму вклада кладчика А. Для этого рассмотрим еще одну переменную типа `strukt vklad *pt`, которая будет перемещаться по списку до обнаружения нужной фамилии:

```
pt = top;

while(pt->  family !=A)

pt = pt->sled;
```

Этот фрагмент можно пояснить так. Сначала `pt` указывает на первый элемент списка. До тех пор пока фамилия вкладчика, на которую указывает переменная `pt`, не будет совпадать с заданной фамилией А, нужно передвигать pt на переменную, задаваемую полем структуры, на которую в данный момент выполнения указывает `pt`.

Приведенный фрагмент будет решать задачу ПОИСК только в том случае, когда можно гарантировать, что человек с фамилией А имеет в системе учета некоторый вклад.

Если это не так, то, «подойдя» к последнему элементу списка, переменная pt приобретает значение NULL, и на следующем шаге выполнения цикла обращение к полю pt –> family вызовет аварийное окончание программы, так как требуемого элемента просто не существует.

Опознать конец списка можно попытаться так:

```
pt=top;

while( pt!=0  &&  (pt->  family !=A))

pt=pt->sled;
```

Это решение в общем случае не устранит аварийное окончание по той же самой причине: в конце списка pt ==NULL и элемента pt –> с полем family, который требуется для выполнения операции сравнения с A в программе учета не создано.

Два варианта правильного решения приведены ниже:

```
1)  pt=top;
   while(pt->sled  !=NULL)      &&    (pt->
family!=A))
           pt=pt->sled;
if ( pt-> family != A)
pt=NULL;
2)  pt=top;
    int b=1;
    while(pt !=NULL   &&  b)
    b=0;
     else
    pt=pt->sled;
```

В обоих случаях после выполнения фрагмента справедливо положение: если человек с фамилией A является вкладчиком, то pt указывает на его вклад, иначе pt ==NULL.

Время поиска элемента с заданным полем пропорционально длине списка, так как при поиске просматривается последовательно весь список.

Следующая типовая задача состоит в том, чтобы вставить новый элемент в список. Для нашего примера это соответствует появлению нового вклада. Вставка возможна в начало списка (перед элементом, на который указывает переменная top) или после любого элемента, например, после элемента, на который указывает `pt`.

```
strukt vklad *new ;

new=( strukt vklad *) malloc(sizeof (strukt vklad ));

new->sled=pt -> sled ;

pt->sled=new;
```

Существуют два вида списков: связанный и кольцевой. Связной список –это простой *однонаправленный* список, в котором каждый элемент (кроме последнего) имеет ссылку на следующий элемент и поле информации. Можно организовать также *кольцевой* список (в нем последний элемент будет содержать ссылку на первый) или *двунаправленный* список, когда каждый элемент, кроме первого и последнего, имеет две ссылки : на предыдущий элемент и следующий элемент и т.д.

Кроме того, можно помещать в начале списка дополнительный элемент, называемый заголовком списка, который может использоваться для хранения информации обо всем списке. В частности, он может содержать счетчик числа элементов списка.

Элемент дважды связанного списка содержит два поля указателей, один из которых указывает на следующий элемент, а другой – на предыдущий. Такая организация позволяет перемещаться списку в двух направлениях: влево и вправо. Циклический список представляет собой «кольцо» элементов и является простой модификацией рассмотренного выше

линейного списка: последний элемент в поле связи вместо значения NULL содержит ссылку на первый элемент списка.

Рассмотрим пример оформления функции, формирующей дважды связанный линейный список. В «содержательной» части элементов списка записан один символ. Формирование списка заканчивается при вводе символа ‘*’.

```
struct element
{
   char  inf;
   struct element  *lev;
   struct element  * prav;
        }
          …….
void form (struct element* perv)
{
struct element  *tek;
char ch;
perv = NULL;
do
{
tek=(struct        element*)malloc(sizeof
(struct element));
tek -> prav = perv;
tek -> lev = NULL;
scanf("%c" , ch);
tek -> inf = ch;
perv = tek;
if (tek -> prav !=NULL)
tek -> prav ->lev = tek;
}
 while(ch=='*');
 }
```

Ниже приведен заголовок функции, осуществляющий удаление элемента, содержательное поле которого совпадает с заданным значением. Параметрами функции является ссылочная переменная- указатель головы списка и заданный символ.

```
void udalen(strukt element* top, char ch );
```

Задание к работе

Задание № 1. Ввести и откомпилировать тестовую программу обработки списка, приведенную в приложении В.

Задание № 2. Составить программу обработки списка в соответствии с номером задания. Вид списка определяется номером строки табл.13.1, а вид обработки – номером столбца, в котором стоит номер задания по табл.13.2. Например, для задания 12 следует рассматривать линейный дважды связанный список и произвести проверку: входит ли в список заданный элемент с заданным значением информационного поля?

В программе предусмотреть функцию формирования списка, функцию его вывода и функцию обработки. Тело программы представляет собой последовательность четырех обращений к функциям: ввод списка, его контрольный вывод, обработку, вывод результата.

Таблица 13.1. Вид списка

Вид списка	1	2	3	4	5	6	7	8	9	10
Линейный			20		25	23	4	21	24	19
Линейный дважды связанный		18	6	8	10	14	12	1	5	9

Таблица 13.2. Вид обработки списка

Номер списка	Вид обработки
1	Подсчитать число элементов списка
2	Добавить новый элемент. Элемент задан ссылочной переменной
3	Добавить новый элемент после заданного. Элемент задан значением информационного поля.
4	Удалить заданный элемент из списка. Элемент задан ссылочной переменной.
5	Удалить заданный элемент. Элемент задан значением поля; удаляется первый элемент
6	Удалить заданный элемент. Элемент задан значением поля; удаляются все такие элементы.
7	Проверить: входит ли заданный элемент в список?
8	Подсчитать: сколько имеется элементов с заданным содержимым одного из полей
9	Объединить два списка – второй добавить в хвост первого (циклические списки разрываются в произвольном месте)
10	Найти элемент с заданным значением информационного поля (из функции обработки возвращается значение ссылочной переменной, указывающей на первый элемент).

11	Подсчитать число элементов списка
12	Добавить новый элемент. Элемент задан ссылочной переменной
13	Добавить новый элемент после заданного. Элемент задан значением информационного поля.
14	Удалить заданный элемент из списка. Элемент задан ссылочной переменной.
15	Удалить заданный элемент. Элемент задан значением поля; удаляется первый элемент
16	Удалить заданный элемент. Элемент задан значением поля; удаляются все такие элементы.
17	Проверить: входит ли заданный элемент в список?
18	Подсчитать: сколько имеется элементов с заданным содержимым одного из полей

Для сохранения списка и последующего вывода его на печать в программе предусмотреть файл для чтения и записи. Файл должен содержать все необходимые поля, представленные в виде таблицы.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.

3. Рисунки, поясняющие процессы обработки списка.
5. Разработанную программу.
6. Сформированный текстовый файл.

Контрольные вопросы

1. Какой список называется линейным?
2. Какой список называется кольцевым?
3. Как проверить: входит ли заданный элемент в список?
4. Как найти список с заданным значением информационного поля?

Работа № 14

Шаблонные (перегружаемые) функции

Цель работы. Освоить принципы программирования перегружаемых шаблонных функций языка С++.

Основные сведения

Шаблонные функции относятся к классу перегружаемых функций и отличаются от обычных тем, что имеют один и тот же код независимо от типа параметров. Например, при работе с массивами различных типов, можно переставлять элементы или копировать содержимое одного массива в другой. В языке С++ такие функции называются шаблонными (template function). Объявление шаблона функции такое же, как и обычной функции, но первой строкой в объявлении пишется строка следующего вида[6,7]:

```
template < class имя_типа, … , class имя_типа >
```

В угловых скобках < > список формальных параметров шаблона. Например:

```
template <class T> void view(const T *p,const size_t size)
template < class T,class S>T* del( T x,T *in,S &size )
```

Каждый параметр типа должен иметь уникальный идентификатор, который может быть использован в разных шаблонах. Каждому параметру типа должно предшествовать ключевое слово `class`.

Задание на выполнение работы

Задание №1. Ознакомиться с назначением и действием шаблонных перегружаемых функций. Протестировать тестовый пример, объяснить результат.

// Тест

```
#include <iostream>
using namespace std ;
// шаблонная функция
template <class T> T abs (const T n ){return n
< 0? -n:n;}
// главная функция
int main ( )
{
      int i ( -5 ); float f( -1.125e-5 ) ; long
l (123L );
      cout << abs ( i ) << endl ;
      cout << i << endl ;
      cout << abs ( f ) << endl ;
      cout << abs ( l ) << endl ;
      cout << abs ( -123L ) << endl ;
     system("pause");
      return 0 ;
}
```

Задание №2. Используйте шаблонную функцию

```
template <class T> void view(const T *p,const
size_t size)
{
   for(int ni=0;i<size;i++)
    cout << p[i]<<'\t';
    cout << endl;
}
```

для вывода трех массивов (int, float,double) и строкового литерала заданной длины.

Задание № 3. Для трех массивов (`int, long, float`) пересортировать массивы: в начало массива записываются отрицательные элементы, после отрицательных элементов записываются положительные элементы. Используйте шаблонную функцию

```
template < class T >void reorg (T *p, const
int size )
{
      T tmp ;
      for ( int i = 0, j = 0 ; i < size ; i++ )
      {
            if ( p [ i ] > 0 )    continue ;
            tmp = p [ i ] ;
      for ( int k = i ; k > j ; k-- )p [ k ] =
p [ k - 1 ] ;
            p [ j ] = tmp ;
            j++ ;
      }
}
```

Вывести не отсортированные и отсортированные массивы, используя для этого функцию из предыдущего задания №2.

`template <class T> void view(const T *p,const int size).`

Задание № 4. Для трех массивов (`int, gouble, char`) пересортировать массивы, удалив из них одинаковые элементы. Пример шаблона для удаления вхождений указанного элемента из массива:

```
template < class T, class S> T* del( T x, T
*in, S &size )
{
      T *out = in ; T *p = in ; T *end = in +
size ;
      while ( in < end )
      {
```

```
            if ( *in != x ) *p++ = *in ;
            else --size ;
            in++ ;
      }
      return out ;
}
```

Вывести не отсортированные и отсортированные массивы, используя шаблонную функцию вывода из задания № 2.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Расчетные данные.

Контрольные вопросы

1. Какая функция называется шаблонной и почему?
2. Какие параметры имеет шаблонная функция?
3. Объяснить работу функции `del()`.

Работа № 15

Классы и объекты в С++

Цель работы. Освоить принципы объектно-ориентированного программирования языка С++. Назначение и работу конструктора и деструктора.

Основные сведения

Понятие и определение класса является определяющим в объектно-ориентированном программировании. Объявление класса во многом совпадает с определением структуры и представляет собой описание членов класса: данных и методов. Члены–данные объявляются согласно правилам объявления переменных и могут иметь любой тип, включая тип класса и указателя на тип класса.

Спецификация класса имеет следующий формат:

```
class
  {
private:
//закрытые члены класса
protected:
//защищенные члены класса
public:
// открытые члены класса
};
```

Внутри класса данные и методы могут быть частными (приватными, `private`), защищенными (`protected`) и общими (`public`). Объектно–ориентированные языки обладают четырьмя важнейшими характеристиками: инкапсуляцией, наследованием, полиморфизмом и абстракцией типов. Понятие инкапсуляции означает, что в качестве единицы целого рассматривается объединение некоторой группы данных и некоторой группы функций. Наследование позволяет одним объектам приобретать атрибуты и свойства других объектов. Полиморфизм означает, что одно и то же имя может

использоваться для логически связанных, но разных целей. Свойства объектов хранятся в структурах данных, напоминающие структуры языка С, а поведение объектов реализуется в виде функций, называемых функциями–членами. В языке С++ реализована защита данных и функций. Если они в объекте объявлены приватными (частными), то к ним нет доступа извне. Зато, если они объявлены общими, то они доступны любому внешнему объекту.

Класс описывает множество объектов с общими свойствами, поведением и семантикой. Каждый класс имеет уникальное имя, определяющее название нового типа данных. Например, для множества книг можно определить класс `CBook[7]`.

```
class CBook
{
private:
      char m_author [ 50 ] ;      // автор
      char *m_pTitle ;       // указатель на
название
      int m_year ;                // год издания
public:
      // методы установки значений
      void setAuthor ( const char* ) ;
      void setTitle ( const char* ) ;
      void setYear ( const int ) ;
      // методы возврата значений
      char* getAuthor ( void ) ;
      char* getTitle ( void ) ;
      int getYear ( void ) ;
} ;
```

В этом примере объявлены три защищенных члена – данных и шесть членов – методов обработки данных. Как правило, объявление класса записывается вне главной функции.

При создании объекта автоматически вызывается специальный метод, который называется *конструктором*. Конструктор управляет построением объекта в оперативной памяти. Конструктор отличается от других членов – методов

тем, что он имеет имя класса и никаких действий по обработке данных не выполняет. Назначение конструктора – инициализация переменных. Конструкторы бывают трех видов: по умолчанию, инициализации, копирования.

```
public:  CBook (); // по умолчанию,
CBook  ()  :  :  CBook  ()  :  m_year(0),
m_pTitle(**)
{  m_Autor [0] = '\0' ; }
```

Конструкторы инициализации и копирования являются конструкторами с параметрами и предназначены для инициализации объектов. Прототип такого конструктора имеет формат:

```
CBook (char*, char*, int );
```

В классе с конструктором используется специальный метод класса *деструктор*, предназначенный для разрушения объекта класса в оперативной памяти. Деструктор имеет открытый спецификатор доступа. Его имя совпадает с именем класса, которому предшествует символ тильда. Для нашего примера

~ CBook ();

Для работы с членами класса необходимо объявить переменную класса – объект. По правилам объявления переменных для класса CBook, объявим объект данного класса как CBook book.

Задание на выполнение работы

Задание №1. В приведенной ниже программе ввести объект, вывести строку "student ". Объяснить результаты вывода, распечатать адреса введенных объектов. Объяснить назначение указателя *this*. Объяснить вид и назначение конструктора.

```
#include <iostream>
using namespace std ;
class C
{
      string m_s ;
public:
      C ( ) : m_s ( "student") { };
      C ( const string& s ) : m_s ( s ) { };
      ~ C ( ) { }
      string getS ( ) const { return m_s ; } ;
      C* operator -> ( ) { return this ; }
      C& operator * ( ) { return *this ; }
      C& operator [ ] ( const unsigned int i )
{ return this [ i ] ; }
} ;
int main ( )
{           C o ( "Ivanov") ;
      cout << "Overloaded operator ->\t" << o -
> getS ( ) << endl ;
      cout << "Overloaded operator *\t" << ( *o
).getS ( ) << endl ;
      cout << "Overloaded operator [ ]\t" << o
[ 0 ].getS ( ) << endl ;
      cout << "Standard operator .\t" << o.getS
( ) << endl ;
      return 0 ;
```

```
}
```

Задание №2. Вывести объекты o1 и o2 в ниже приведенной программе.

```
#include <iostream>
#include<string.h>
using namespace std ;
class C
{
public:
      // конструкторы и деструктор
      C ( char* s ){ strncpy ( m_s,  s,40 ) ;
incObj ( ); }
      C ( ){ m_s [ 0 ] = '\0' ; incObj ( ) ; }
      ~C ( )     { --objAmount ; }
      // открытые статические методы
static void initAmount ( const int v ) {
objAmount = v ;}
      static int getAmount ( ) { return
objAmount ; }

   private:
      // закрытые статические метод и член-
данное
      static int incObj ( ) { return
++objAmount ; }
      static int objAmount ;
      // закрытые нестатический метод и член-
данное
      int lenS ( ) {  return strlen ( m_s ) ;
}
      char m_s [ 40 ] ;

} ;
// объявление статической переменной
int C :: objAmount ;
// главная функция
int main ( )
```

```
{
      C :: initAmount ( 0 ) ;     //
инициализация objAmount
      // объявление и создание объектов
     C o1 = "St. Petersburg",o2 = "Yalta";
      C o [ 12 ] , o3 ;
      C* p = new C [ 25 ] ;
      // вывод состояния статической переменной
      cout << C :: getAmount ( ) << endl ;  //
выводится 40
      // разрушение объектов
      delete [ ] p ;
      // вывод состояния статической переменной

      cout << C :: getAmount ( ) << endl ;  //
выводится 15

       system("pause");
      return 0 ;
}
```

Для вывода используйте дружественную функцию.

```
char *ret().
{return m_s;}
```

Объяснить результаты работы программы.

Задание № 3. Объявить объект класса *C* с числом членов не более 10. Объявить и инициализировать целочисленный массив с положительными и отрицательными членами. Присвоить члены массива объекту класса и сделать их все положительными. Вывести значение любого элемента массива. Запустить программу и объяснить результаты. Распечатать содержимое, в том числе * m_p .

```
class C
{
public:
```

```
      int* m_p ; int m_n ;
      C ( int n = 10 ) : m_n ( n )
      {
        m_p = new int [ m_n ] ;
        for ( int i = 0; i < m_n; i++ )
       m_p [ i ] = 0 ;
      }
      ~ C ( ) { delete [ ] m_p ; }
      int& operator [ ](const unsigned int i )
const
  { return m_p [ i ] ; }
   } ;
   #include <iostream>
   using namespace std ;
   int main ( )
   {
      C o ( 5 ) ;
     for ( int i = 0; i < 5; i++ )
           cout << o [ i ] << '\t' ;
      cout << endl ;

      cout << endl ;
      system("pause");
      return 0 ;
   }
```

Задание № 4. Создать класс с функциями вывода состояния объекта класса CBook. Протестировать программу, инициализировать своими данными (не менее пяти авторов).

```
#include <string.h>
#include <iostream>
using namespace std;

class CBook
{
private:
      char m_author [ 50 ] ;     // автор
      char *m_pTitle ;      // указатель на
название
```

```
     int m_year ;     // год издания
public:
   CBook ( char *author, char *title =
NULL,int year = 0 ):
           m_year ( year ), m_pTitle ( title )
    {
         strncpy ( m_author, author, 50 ) ;
         if ( strlen ( author ) > 49 )
           m_author [ 49 ] = '\0' ;
     cout << "with parameters
CONSTRUCTOR\nthis = "  ;
    }
        CBook(void):m_pTitle(NULL), m_year(0)
    {
        strcpy(m_author, "None");
    }

     CBook ( CBook &o ) : m_year ( o.m_year )
     {
       strcpy( m_author,  o.m_author ) ;
        m_pTitle = new char [strlen (
o.m_pTitle ) + 1 ] ;
            strcpy( m_pTitle, o.m_pTitle ) ;
     cout << "CONSTRUCTOR of copying\nthis = "
<< this ;
     }
    ~CBook ( )
    { delete [ ] m_pTitle ;      }
     // методы установки значений
     void setAuthor ( const char* author )
    {
     strncpy ( m_author,  author, 50) ;
     if ( strlen ( author ) > 49 ) m_author [
49 ] = '\0' ;
     }
     void setTitle ( const char* title )
     {
      delete [ ] m_pTitle ;
```

```
     m_pTitle = new char [strlen ( title )+ 1]
;
      strncpy ( m_pTitle, title, strlen (
title ) + 1 ) ;
    }
     void setYear ( const int year )
     {
         m_year = year ;
     }
     // методы возврата значений
     char* getAuthor ( void )
      {
         return m_author ;
     }
     char* getTitle ( void )
     {
          return m_pTitle ;
    }
     int getYear ( void )
     {
        return m_year ;
    }
} ;
void view (char* s, CBook &o );

int main ( )
{CBook book  ;
int god;
char nazv[20];
char auth[20];
 cout<<"Enter the name of Author of the book
"<< "\n";
 cin >>auth;
       book.setAuthor ( auth ) ;
       cout<<"Enter the name of the book "<<
"\n";
     cin >>nazv;
    book.setTitle (nazv);
```

```
     cout<<"Enter the Year of the book\n ";
     cin >> god;
 book.setYear (god);


 view ( "book", book) ;
     system("pause");
     return 0;
}
// функция вывода состояния объекта класса
CBook
  void view (char *s, CBook &o )
  {
     cout << "\nState of object \' " << s << "
\'\n" ;
     cout << "Author:\t" << o.getAuthor ( ) <<
endl ;
     cout << "Title:\t" << o.getTitle ( ) <<
endl ;
     cout << "Year:\t" << o.getYear ( ) <<
endl << endl ;

  }
```

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Расчетные данные.

Контрольные вопросы

1. Что представляет собой класс?
2. Назовите виды конструкторов в примере 1.
3. Какой параметр используется в конструкторе копирования?

Работа № 16

Наследование в среде С++

Цель работы. Приобретение практических навыков работы с объектно-программируемым языком С++. Освоить основные элементы механизма наследования.

Понятие класса и объекта

Одним из главных понятий языка С++ является понятие класса (`class`). Чтобы определить объект надо сначала определить его форму с помощью ключевого слова `class`. Рассмотрим пример базового класса и на его примере сформируем производный класс[2].

Пример 1. Объявление класса

```
//Объявим класс queue (очередь)
Class queue {
private: //режим доступа частный
int q[100];
int  i, j;
public: //режим доступа открытый
void  func1(void);
int  func2(inti);
protected:
int a,d; //режим доступа защищенный
} [ список объектов ];
```

Список объектов может быть задан непосредственно при формировании класса или же в функции `main()` по аналогии с переменными структур в языке Си. Например, для рассматриваемого класса: `queue` *a,b* задаются два объекта *a* и *b*. Когда же требуется описать функцию – член класса, то необходимо указать к какому классу она принадлежит. Например, функция `func1(void)` принадлежит классу queue. Это записывается форматом:

```
queue :: func1(int i)
{ i = 0;
……..
}
```

Чтобы вызвать функцию – член класса в той части программы, которая не является частью класса, надо использовать имя – объекта и операцию доступа (.). Например, если объявлен объект *a* класса `queue` (`queue` *a*), то для вызова функции `func1()` нужно записать:

a. `func1();`

Основная форма наследования

`Class`< имя_наследующего_класса>: < режим_доступа> < наследуемый_класс>.

Пример 2 . Объявление наследуемого класса

```
Class queue1: public queue
{  int sum;
public:
int get_sum(void);};
```

В этом примере объявлен класс `queue1` как наследователь класса `queue`. Он наследует члены базового класса и дополнительно может содержать свои члены (`get_sum(void)`).

Задание на программирование

1. Ввести программу, реализующую очередь (Пример 3). Найти и исправить ошибки.

2. Создать наследующий класс и включить его в программу примера 3.

Пример 3.

```
#include <iostream >
class queue {
int q[10];
int sloc,rloc;
public:
void  init();
void  qput(inti);
int  qget(void);
   };
void queue::init(void)
{ sloc=rloc=0;
   }
int queue::qget()  {
if(sloc==rloc)
   {cout<<"cohered pusta"<<"\n";
return 0;
   }
return q[rloc++];
   }
void queue::qput(inti)
    {
```

```
if(sloc==10)
cout<<"ocheredpolna"<<"\n";
return;
     }
q[sloc++]=i;
   }
main()
   {
queue a,b;
a.init();
b.init();
a.qput(7);
a.qput(9);
a.qput(11);
cout<<a.qget()<<"\n";
cout<<a.qget()<<"\n ";
cout<<a.qget()<<"\n" ;
cout<<a.qget()<<"\n";
for(inti=0;i<12;i++)
b.qput(i*i);
for(i=0;i<12;i++)
cout<<b.qget()<<" ";
cout<<"\n";
```

```
getch();
return 0; }
```

Указания для выполнения

1. Создаем наследующий класс

```
Class queue1:public queue{
int sum;
public:
int get_sum(void);
void show_sum(void);
     };
```

2.Создаем объект наследующего класса: `queue1` **`obj`**.

3. Описываем функции наследующего класса.

```
int queue1::get_sum(void)
{
sum=0;
for (inti=rloc;  i<sloc;i++)
sum+=q[i];
return sum;
}
void queue1::show_sum(void)
 {
  cout<<"summa ochered="<<sum<<"\n";
 }
```

4. Дополняем главную функцию программы фрагментом вызова функций.

```
queue1 obj;
obj.init();
for(inti=0;i<5;i++)
   {
     obj.qput(100+i);
```

```
      obj.get_sum();
      obj.show_sum();
    }
for(i=0;i<6;i++)
     {
      obj.get_sum();
      obj.show_sum();
      obj.qget();
     }
```

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Расчетные данные.

Контрольные вопросы

1. Дайте определение класса и объекта.
2. Дайте определение инкапсуляции и наследования.
3. Какие функции называются перегружаемые и почему?
4. Что общего между классом и структурой в языке Си?
5. Какие члены базового класса наследует производный класс?

Работа № 17

Дружественные функции

Цель работы. Изучить назначение и работу дружественных функций.

Как следует из определения класса, в нем содержатся закрытые и защищенные члены класса. Возникает вопрос: Как получить доступ к этим членам? Для этого создается механизм дружественных функций[6]. Дружественная функция – это функция, которая не является членом класса, но имеет доступ к членам класса, объявленных в полях *private* или *protected*. Такие функции имеют перед именем модификатор *friend*. Дружественные функции будут получать данные об объекте, обрабатывать их как обычные методы класса.

В приведенном ниже примере запрограммируем следующее: создадим класс Woman25, который, используя дружественные функции и обычные методы класса, будет получать данные об объекте (имя и вес) и выводить их на экран. Методы и *friend*–функции будут выполнять аналогичные действия. В этом и есть особенная польза данного примера — вы сможете посмотреть отличия в объявлении и определении дружественных функций от обычных методов класса. На основании полученных данных, программа даст пользователю совет относительно корректировки его веса.

Задание на выполнение

1.Скопировать и запустить программу с дружественными функциями., приведенную ниже

2. Ввести еще одно имя в программу и дополнить объекты параметром «рост».

```
//Программа с дружественными функциями
#include <iostream>

#include <string.h>

using namespace std;

class Woman25

{

private:

har *name;//имя

nt weight;//вес

ъявление дружественных функций

end void setData(char *, int, Woman25&);

end void getData(Woman25&);

ic:

oman25()//конструктор


   name = new char [20];

   strcpy(name, "Норма");

   weight = 60;


Woman25()//деструктор
```

```
    delete [] name;
    cout << "!!! Деструктор !!!" << endl;
}
   void setData(char*, int);//объявление
етодов класса
void getData();
void advise();

ределяем friend-функцию setData
 setData(char *n, int w, Woman25& object)

    strcpy(object.name, n);
    object.weight = w;
}
пределяем friend-функцию getData
d getData(Woman25& object)
{cout << object.name<< "\t:"<< object.weight
 кг" << endl;
```

```
} //определяем set-метод класса
void Woman25::setData(char *n, int w)
    {
        strcpy(name, n);
        weight = w;
    }
     void Woman25::getData()
//определяем get-метод класса
    {
     cout << name << "\t: " << weight
 << " кг" << endl;
    }
     void Woman25::advise()
//определяем метод класса Совет (advise)
    {
   if(weight < 55) //если вес меньше 55 кг
{cout << "Вам надо потреблять больше калорий!"
      << endl;
       cout << "==================" << endl
```

```
<< endl;
        }
  else if(weight >= 55 && weight <= 65)
{
    cout << "Ваш вес в норме!" << endl;
   cout << "========================" <<
endl << endl;
        }
   else { //если вес > 65 кг
cout << "Вам надо ограничивать себя в еде!"
 << endl;
cout << "================" << endl << endl;
        }
    }
 int main()
{    setlocale(LC_ALL, "rus");
     Woman25 Norm;
//создаем объект Norm, и weight
будет = 60, name - Норма
```

```
    Norm.getData(); //вызов метода класса
    cout << "============" << endl << endl;
     Woman25 Anna; //второй объект
    Anna.setData("Анна", 100);
    Anna.getData();
    Anna.advise();
     Woman25 Inna; //третий объект
    setData("Инна", 50, Inna);
    getData(Inna);
    Inna.advise();
   return 0;
}
```

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Расчетные данные.

Контрольные вопросы

1. Что отличает дружественную функцию от обычных методов класса?

Работа № 18

Виртуальные функции в наследовании

Цель работы. Изучить принципы программирования виртуальных функций при наследовании базового класса.

Чистая виртуальная функция (pure virtual function) – это функция, которая не имеет реализации в базовом классе. Она представляет собой прототип для определения функций, которые располагаются в производных классах.

Класс, в котором объявлена хотя бы одна виртуальная функция, называется абстрактным (abstract class). Для таких классов невозможно создать объекты. Абстрактные классы могут использоваться только в качестве базовых для других производных классов. Если в производном классе, который наследуется от абстрактного базового класса, отсутствует реализация чистой виртуальной функции, то этот производный класс тоже будет являться абстрактным.

Объявление чистой виртуальной функции в базовом классе имеет следующий формат:

virtual <тип_функции>< имя_функции> (<список_параметров>) = 0;

В производных классах прототип указывается как обычно, однако он не может быть изменен. Это значит, что тип возвращаемого значения (тип функции), типы и количество ее параметров должны строго соответствовать описанию, которое задано в базовом классе.

Задание на выполнение

Задание № 1. Провести разбор программы, код которой приведен ниже[7]. В программе используется абстрактный класс *cFigure*, который описывает геометрические фигуры и содержит три виртуальные функции. Протестировать программу при новых значениях параметров.

Задание № 2. Дополнить программу новым объектом, например, ромбом. Вычислить его площадь: $S=1/2\ d_1\ d_2$.

Задание № 3. Извлечь из программы какой– либо объект. Протестировать программу.

//Листинг программы использования чистой виртуальной функции и //наследования

```
// Specifications.h - Спецификации классов
// спецификация CFigure - геометрическая
фигура (базовый класс)
class CFigure
{
protected:
      double m_a, m_b ;// две стороны фигуры
public:
      CFigure ( ) ; CFigure ( const double,
const double ) ;
virtual ~CFigure ( ) ;// виртуальные функции
      virtual void setFigure (double, double) ;
      virtual void getFigure ( double&, double&
) const ;
// чистая виртуальная функция - вычисление
//площади фигуры
virtual double getArea ( ) const = 0 ;} ;
// спецификация CTriangle - треугольник
//(производный класс)
class CTriangle : public CFigure
{
protected:
      double m_c ;// третья сторона
public:
      CTriangle ( ) ; CTriangle ( double,
double, double ) ;
      ~CTriangle ( ) ;
```

```
      void setFigure(double, double, double );
      void getFigure ( double&, double&,
double& ) const ;
      double getArea ( ) const ;// вычисление
//площади треугольника
} ;
// спецификация CRectangular - прямоугольник
// (производный класс)
class CRectangular : public CFigure
{
public:
      CRectangular ( ) ; CRectangular ( double,
double ) ;
      ~CRectangular ( ) ;
      double getArea ( ) const ; // вычисление
//площади прямоугольника
} ;
// спецификация CParallelogram -
//параллелограмм (производный класс)
class CParallelogram : public CTriangle
{
public:
      CParallelogram ( ) ; CParallelogram (
double, double, double ) ;
      ~CParallelogram ( ) ;
      double getArea ( ) const ;// вычисление
//площади параллелограмма
} ;
// спецификация CTrapeze - трапеция
// (производный класс)
class CTrapeze : public CTriangle
{
protected:
```

```
double m_d, m_h ;// четвертая сторона и высота
public:
     CTrapeze ( ) ;
CTrapeze ( double, double, double, double,
double ) ;
     ~CTrapeze ( ) ;
     void setFigure ( double, double, double,
double, double ) ;
     void getFigure ( double&, double&,
double&, double&, double& )  const ;
     double getArea ( ) const ;// вычисление
//площади трапеции
} ;
// Realizations.cpp - Реализации классов
// реализация CFigure - (базовый класс)
CFigure :: CFigure ( ) : m_a ( 0.0 ), m_b (
0.0 ) { }
CFigure :: CFigure ( double a, double b ) :
m_a ( a ), m_b ( b ) { }
CFigure :: ~CFigure ( ) { }
void CFigure :: setFigure ( double a, double b
) { m_a = a ; m_b = b ; }
void CFigure :: getFigure(double& a, double &b
) const
{     a = m_a ; b = m_b ;   }
// чистая виртуальная функция не имеет
//реализации в базовом классе
// реализация CTriangle - треугольник //
(производный класс)
CTriangle :: CTriangle ( ) : CFigure ( ), m_c
( 0.0 ) { }
CTriangle :: CTriangle ( double a, double b,
double c )
```

```
: CFigure ( a, b ), m_c ( c ) { }
CTriangle :: ~CTriangle ( ) { }
void CTriangle :: setFigure ( double a, double
b, double c )
{     CFigure :: setFigure ( a, b ) ; m_c = c ;
      }
void CTriangle :: getFigure ( double& a,
double& b, double& c ) const
{     CFigure :: getFigure ( a, b ) ; c = m_c ;
      }
// реализация чистой виртуальной функции
// площадь треугольника вычисляется по формуле
//Герона
#include <cmath>
using namespace std ;
double CTriangle :: getArea ( ) const
{
      double p = ( m_a + m_b + m_c ) / 2 ;
      return sqrt ( p * ( p - m_a ) * ( p - m_b
) * ( p - m_c ) ) ;
}
// реализация CRectangular - прямоугольник
// (производный //класс)
CRectangular :: CRectangular ( ) : CFigure ( )
{ }
CRectangular :: CRectangular ( double a,
double b )
: CFigure ( a, b ) { }
CRectangular :: ~CRectangular ( ) { }
// реализация чистой виртуальной функции
double CRectangular :: getArea ( ) const {
return m_a * m_b ; }
```

```
// реализация Romb – параллелограмм
// (производный класс)
// член-данное m_c базового класса CTriangle -
//это высота параллелограмма
CParallelogram :: CParallelogram ( ) :
CTriangle ( ) { }
CParallelogram :: CParallelogram ( double a,
double b, double h )
: CTriangle ( a, b, h ) { }
CParallelogram :: ~CParallelogram ( ) { }
// реализация чистой виртуальной функции
double CParallelogram :: getArea ( ) const {
return m_a * m_c ; }
// реализация CTrapeze - трапеция (производный
//класс)
CTrapeze :: CTrapeze ( ) : CTriangle ( ), m_d
( 0.0 ), m_h ( 0.0 ) { }
CTrapeze :: CTrapeze ( double a, double b,
double c, double d, double h )
: CTriangle ( a, b, c ), m_d ( c ), m_h ( h )
{ }
CTrapeze :: ~CTrapeze ( ) { }
void CTrapeze ::
setFigure ( double a, double b, double c,
double d, double h )
{     CTriangle :: setFigure ( a, b, c ) ; m_d
= d ; m_h = h ; }
void CTrapeze ::
getFigure ( double& a, double& b, double& c,
double& d, double& h ) const
{     CTriangle :: getFigure ( a, b, c ) ; d =
m_d ; h = m_h ; }
// реализация чистой виртуальной функции
```

```
double CTrapeze :: getArea ( ) const
{     return 0.5 * ( m_a + m_b ) * m_h ;    }
// main.cpp - главная функция
#include <iostream>
using namespace std ;
//#include "Specifications.h"
void view ( double, double ) ;
void view ( double, double, double ) ;
void view ( double, double, double, double,
double ) ;
//#pragma once
int main ( )
{
// объявление массива указателей на базовый
//класс
     CFigure* pF [ 4 ] ;
// объявление массива указателей с
//инициализацией //названиями фигур
     char* name [ ] =
{ "Triangle", "Trapeze", "Parallelogram",
"Rectangular" } ;
     // создание объектов разных типов
     CTriangle triangle ( 3.0, 4.0, 2.0 ) ;
     CTrapeze trapeze ( 5.0, 2.0, 1.9, 2.8,
1.8 ) ;
     CParallelogram parallelogram ( 15.2,
28.75, 18.3 ) ;
     CRectangular rectangular ( 100.5, 24.13 )
;
// инициализация массива pF адресами созданных
//объектов
     pF [ 0 ] = &triangle ;     pF [ 1 ] =
&trapeze ;
```

```
      pF [ 2 ] = &parallelogram ;pF [ 3 ] =
&rectangular ;
// цикл вывода состояния созданных объектов и
//их площадей
      double a, b, c, d, h ;
      for ( int i = 0; i < 4; i++ )
      {
      cout << '\n' << name [ i ] << " ( ";
            switch ( i )
            {
            case 0:
            triangle.getFigure ( a, b, c ) ;
      view ( a, b, c ) ; cout << " )\t\t\t" ;
break ;
      case 1:
      trapeze.getFigure ( a, b, c, d, h ) ;
      view ( a, b, c, d, h ) ; cout << " )\t\t"
; break ;
      case 2:
      parallelogram.getFigure ( a, b, h ) ;
      view ( a, b, h ); cout << " )\t"; break ;
      case 3:
      rectangular.getFigure ( a, b ) ;
      view ( a, b ) ; cout << " )\t\t" ;break ;
            }
      cout << "AREA = " << pF [ i ] -> getArea
( ) << endl ;
      }
      // цикл изменения данных объектов, вывода
//их состояния //и площадей
      for ( int i = 0; i < 4; i++ )
      {
      cout << '\n' << name [ i ] << " ( ";
```

```
       switch ( i )
       {
       case 0:
       triangle.setFigure ( 3, 3, 3 ) ;
       triangle.getFigure ( a, b, c ) ;
       view ( a, b, c ) ; cout << " )\t\t\t" ;
break ;
       case 1:
      trapeze.setFigure (5, 2, 1.8, 1.8, 1.9);
      trapeze.getFigure ( a, b, c, d, h ) ;
      view ( a, b, c, d, h ); cout << " )\t\t"
; break ;
       case 2:
      parallelogram.setFigure ( 14, 15, 14 ) ;
      parallelogram.getFigure ( a, b, h ) ;
      view ( a, b, h ); cout << " )\t\t" ;
break ;
      case 3:
      rectangular.setFigure ( 1005, 2413 ) ;
      rectangular.getFigure ( a, b ) ;
      view ( a, b ) ; cout << " )\t\t"; break ;
           }
      cout << "AREA = " << pF [ i ] -> getArea
( ) << endl ;
      }
 system("pause");
      return 0 ;
}
// определение перегруженной функции вывода
//переменных
void view ( double a, double b )
{     cout << a << ", " << b ;   }
void view ( double a, double b, double c )
```

```
{     view ( a, b ) ; cout << ", " << c ;   }
void view (double a, double b, double c,double
d,double h)
{view ( a, b, c ) ; cout << ", " << d << ", "
<< h ;}
```

Пояснения к программе

Программа содержит виртуальные функции класса, которые начинаются с префикса *set* и *get*. Эти функции предназначены для установки и получения значений данных объектов производных классов. Установка и получение данных осуществляется по именам объектов: CTriangle, CTrapeze, CParallelogram, CRectangular. Виртуальная функция getArea() предназначена для вычисления площади фигуры. Вызов чистого виртуального метода производится через указатель на базовый тип pF[i] , в котором задается адрес производного объекта.

В программе используется перегружаемая функция view() для вывода разного числа переменных с плавающей точкой и вывода состояния объектов производных классов.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Разработанную программу.
4. Расчетные данные.

Контрольные вопросы

1. Что отличает виртуальную функцию от обычных методов класса?

2. С какими членами класса работает виртуальная функция?

3. Почему функция view() называется перегружаемой?

Список рекомендуемых источников

1. Шишкин А.Д. Программирование на языке СИ. Учебное пособие. СПб.: Изд. РГГМУ, 2003. –104 с.

2. Березин Б. И., Березин С. Б. Начальный курс С и С++. – М.: ДИАЛОГ – МИФИ, 2009.–288 с.

3. Крячков А. В., Сухинина И. В., Томшин В.К. Программирование на С и С++. Практикум: Учеб. пособие для вузов / Под ред. В.К. Томшина .– 2–е изд., испр. – М.: Горячая линия – Телеком, 2000.

4. С/С++. Структурное программирование: Практикум/ Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2003. –240 с.

5. Культин Н.Б. Самоучитель С++ Builder. – СПб. : БХВ – Петербург, 2006. – 320 с.

6. Довбуш Г.Ф. Visual C++ на примерах / Г.Ф. Довбущ, А.Д. Хоменко / Под ред. проф. А.Д. Хоменко. СПб. : БХВ – Петербург, 2007.– 528 с.:ил.

7. Хортон , Айвор. Visual C++ 2010, Полный курс. : Пер. с англ.– М. : ООО «И.Д. Вильямс». 2011, 1216 с.

ПРИЛОЖЕНИЕ А

Программа с датчиком случайных чисел

```
// консольное приложение программы «Угадай число»[5]
#include <stdio.h>
# include <conio.h> // длядоступакgetchf)
#include <stdlib.h> // длядоступакsrand(), rand()
#include <time.h> // длядоступактime_tutime()
char* rus (char* st) ; // преобразует ANSI-строкувстроку ASCII
#pragma argsused
main()
{
int comp, // число, "задуманное" компьютером
igrok, // вариант игрока
n=0; // число попыток
// ГСЧ — генератор случайных чисел
time_t  t; // текущее время (для инициализации ГСЧ)
srand((unsigned) time (&t) ) ; // инициализацияГСЧ
comp = rand () % 10 + 1;
puts( rus ( " \n Компьютер \"задумал\" число от 1 до 10."));
puts ( rus ( "Вы должны его угадать за три попытки.")));
do
{
printf ("->");
scanf("%i",&igrok) ;
while(igrok != comp && n < 3) ;
if (igrok == comp)
printf ( rus ( " Выпроиграли! " ) ) ;
else{
puts ( rus ( "Выпроиграли . " ) ) ;
printf( rus ( "Компьютер \"задумал\" число %d") , comp) ;
}
printf (rus ("\пДля завершения нажмите любую клавишу. " ) ) ;
getch( );
return 0;
}
```

ПРИЛОЖЕНИЕ Б

Задания к лабораторной работе № 4

Вычислить и вывести на экран в виде таблицы значения функции F на интервале от $X_{\text{нач.}}$ до $X_{\text{коч.}}$ с шагом dx при заданных условиях.

№	Функция	Условия
1	$F = \begin{cases} ax^2 + b, & \text{при } x < 0 \text{ и } b \neq 0 \\ \dfrac{x-a}{x-c}, & \text{при } x > 0, b = 0, c \neq 0 \\ \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения a, b, c, $X_{\text{нач.}}$, $X_{\text{кон.}}$, dx ввести с клавиатуры
2	$F = \begin{cases} \dfrac{1}{ax} - b, & \text{при } x + 5 < 0 \text{ и } c = 0 \\ \dfrac{x-a}{x}, & \text{при } x + 5 > 0 \text{ и } c \neq 0 \\ \dfrac{10x}{c-4}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения a, b, c, $X_{\text{нач.}}$, $X_{\text{кон.}}$, dx ввести с клавиатуры
3	$F = \begin{cases} ax^2 + bx + c, & \text{при } a < 0 \text{ и } c \neq 0 \\ \dfrac{-a}{x-c}, & \text{при } a > 0 \text{ и } c = 0, x \neq c \\ \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения a, b, c, $X_{\text{нач.}}$, $X_{\text{кон.}}$, dx ввести с клавиатуры
4	$F = \begin{cases} -ax - c, & \text{при } c < 0 \text{ и } a \neq 0 \\ \dfrac{x-a}{-c}, & \text{при } c > 0 \text{ и } a = 0 \\ \dfrac{bx}{c-a}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения a, b, c, $X_{\text{нач.}}$, $X_{\text{кон.}}$, dx ввести с

		клавиатуры
5	$F = \begin{cases} a - \dfrac{x}{10 + b}, \text{при } x < 0 \text{ и } b \neq 0 \\ \dfrac{x - a}{x - c}, \text{ при } x > 0 \text{ и } b = 0 \\ 3x + \dfrac{2}{c}, \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
6	$F = \begin{cases} ax^2 + b^2x, & \text{при } a < 0 \text{ и } b \neq 0 \\ \dfrac{x + a}{x + c}, & \text{при } c > 0 \text{ и } a = 0 \\ \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
7	$F = \begin{cases} -ax^2 - b, & \text{при } x < 5 \text{ и } b \neq 0 \\ \dfrac{x - a}{x - c}, & \text{при } x > 5 \text{ и } c \neq 0 \\ 1 - \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
8	$F = \begin{cases} ax^2 + c, & \text{при } c < 0 \text{ и } a \neq 0 \\ \dfrac{x - a}{xc}, & \text{при } c > 0 \text{ и } a = 0 \\ \dfrac{x - c}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
9	$F = \begin{cases} ax^2 + b^2x, & \text{при } a < 0 \text{ и } x \neq 0 \\ x - \dfrac{a}{x - c}, & \text{при } a > 0 \text{ и } c \neq x \\ 1 + \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры

10	$F = \begin{cases} ax^2 - bx, & \text{при } x < 3 \text{ и } b \neq 0 \\ \dfrac{x-a}{x-c}, & \text{при } x > 3 \text{ и } a = 0 \\ \dfrac{x}{c}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
11	$F = \begin{cases} ax^2 + \dfrac{b}{c} & \text{при } x < 1 \text{ и } c \neq 0 \\ \dfrac{x-a}{(x-c)^2}, & \text{при } x > 1 \text{ и } c = 0 \\ \dfrac{x^2}{c^2}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
12	$F = \begin{cases} ax^2 + b^2 + c, & \text{при } x < 0{,}5 \text{ и } b \neq 0 \\ \dfrac{x-a}{x-c}, & \text{при } x > 0{,}6 \text{ и } b + c = 0 \\ \dfrac{x}{c} + \dfrac{x}{a}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
13	$F = \begin{cases} ax^2 + b, & \text{при } x - 1 < 0 \text{ и } b \neq 0 \\ \dfrac{x-a}{x}, & \text{при } x - 1 > 0 \text{ и } b = 0 \\ \dfrac{x}{c-x}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
14	$F = \begin{cases} ax^2 + b^2, & \text{при } x + c < 0 \text{ и } a \neq 0 \\ \dfrac{x-a}{x-c}, & \text{при } x + c > 0 \text{ и } a = 0 \\ \dfrac{x}{c} + \dfrac{c}{x}, & \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры

15	$F = \begin{cases} ax^2 + b, \text{при } x < 0 \text{ и } b \neq 0 \\ \dfrac{x}{x-c} + 8, \text{ при } x > 0 \text{ и } b = 0 \\ \dfrac{x}{-c}, \quad \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
16	$F = \begin{cases} a(x+c)^2 - b, \text{при } x = 0 \text{ и } b \neq 0 \\ \dfrac{x-a}{c}, \text{ при } x = 0 \text{ и } c \neq 0 \\ a + \dfrac{x}{x+c}, \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
17	$F = \begin{cases} ax^2 - cx + b, \text{при } x + 5 < 0 \\ \dfrac{x-a}{x-c}, \text{ при } x + 5 > 0 \text{ и } x \neq c \\ \dfrac{-x}{a-c}, \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
18	$F = \begin{cases} ax^2 + bx^2, \text{при } x < 0 \text{ и } b \neq 0 \\ \dfrac{x-a}{x-c}, \text{ при } x > 0 \text{ и } x \neq c \\ \dfrac{x+5}{c(x-10)}, \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры
19	$F = \begin{cases} a(x+7)^2 - b, \text{при } x < 5 \text{ и } b \neq 0 \\ \dfrac{x-cb}{c+5}, \text{ при } x > 5 \text{ и } b = 0 \\ \dfrac{x}{c}, \quad \text{в остальных случаях} \end{cases}$	a, b, c – действительные числа. Значения $a, b, c, X_{\text{нач.}}, X_{\text{кон.}}, dx$ ввести с клавиатуры

20	$F = \begin{cases} -\dfrac{2x-c}{cx-a}, \text{при } x < 0 \text{ и } c \neq 0 \\ \dfrac{x-a}{x-c}, \quad \text{при } x > 0 \text{ и } (x-c) \neq 0 \\ \dfrac{-x}{c} + \dfrac{-c}{2x}, \text{в остальных случаях} \end{cases}$	*a*, *b*, *c* – действительные числа. Значения *a*, *b*, *c*, $X_{\text{нач.}}$, $X_{\text{кон.}}$, *dx* ввести с клавиатуры

Приложение В

Программа обработки списка

```
#include <iostream>
#include <list>
#include <stdio.h>
#include <Windows.h>
#include <time.h>
 using namespace std;
struct List
{   int data;
    double a;
    List *next;
};
List *Head=0;
 List* Read(List *L)
{
    List *a = new List;
    a->next = NULL;
   cout << "Введите элемент: ";
    cin >> a–>a;
    if (L == NULL)
```

```
    {
      L = a;
    }
    else
    {
        List *c = L;
        while (c->next != NULL) c = c->next;
        c->next = a;
    }
    return L;
}
 void Printlist(List *L)
{
    cout<<"\nВаш список: ";
    while (L != NULL)
    {
        cout << L->a << "\t";
        L = L->next;
    }
    cout << endl;
}
```

```
List* End(List *L)
{
   while (L != NULL)
   {
      L = L->next;
   }
   return L;
}
void add(List *&top, int pos, int x) //Добавление элемента в любое место списка
{
   int j = 1;
   int i; List *pnew, *p = top;
   pnew = new List; pnew->a = x;
   if (pos <= 1 || top == NULL)
   {
      pnew->next = top; top = pnew;
   }
   else
   {
      for (i = 1; i<pos - 1; i++)
      if (p->next != NULL)
```

```
        {
            p = p->next;
            j++;
        }
        if (pos <= j + 1)
        {
            pnew->next = p->next;
            p->next = pnew;
        }
    }
}
void Delete(int p, List *L) //Удаление элемента по его позиции в списке
{
    List*& a = L;
    List* b = a->next;
    if (p == 1) {
        a = a->next;
        cout << "Элемент удален" << endl;
    }
    else{
```

```
        while (p>2)
        {
            p--;
            a = a->next;
            b = b->next;
            if (b == NULL) break;
        }
        if (b != NULL) {
            a->next = b->next;
            cout << "Элемент удален" << endl;
        }
        else cout << "Такой позиции нет" << endl;
    }
}
void sizeL(List *L)
{
    int i;
     while (L != NULL)
    {
        i++;
        L = L->next;
```

```
    }
    cout <<"\nВ спике "<<i<<" элементов"<< endl;
}
void searchL(List *L, int x)
{
     List *a=L;
    while (a->next != NULL)
      {
    a->next;
    if (L->a==x)
    {
    cout<< "element takoy"<<'\t'<<x<<'\t'<< "ect"<<endl;
   break;
     }
    else
    {cout<<"no element"<<endl;
    break;    }
        }
  }
  void menu(List *L)
{
```

```
    int choice;
    do
    {
        cout << "\n Выберите необходимое действие:\n1-Ввод элементов в список\n2-Ввод в любое место списка\n3-Вывод списка\n4-Удаление элементов списка\n5-Определение количества элементов в списке\n6-Поиск элемента в списке\n\n9-Выход из программы\n" << endl;
 cin >> choice;
        switch (choice)
        {
        case 1:
            L = Read(L);
            break;
         case 2:
          int pos, x;
          cout << "Введите номер позиции для вставки: ";
          cin >> pos;
          cout <<"Введите значение элемента для вставки: ";
            cin >> x;
            add(L, pos, x);
             break;
        case 3:
```

```
      Printlist(L);
      break;
    case 4:
      int p;
      cout << "Введите p: ";
      cin >> p;
      Delete(p, L);
      break;
    case 5:
      sizeL(L);
      break;
    case 6:
      cout << "Введите элемент для поиска: ";
      cin >> x;
      searchL(L,x);
      break;
    case 9:
      choice = 0;
      break;
    }
} while (choice != 0);
```

```
}

 int main()

{

setlocale(LC_ALL, "Rus");

int choice;

system("color A");

cout << "Здравствуйте, хотите создать новый список?\n1-да\n0-нет\n" << endl;

cin >> choice;

 if (choice == 1){

List* list = NULL;

menu(list);

 }

cout <<"\nДо свидания!";

getchar();

return 0;

}
```

ПРИЛОЖЕНИЕ Г

Образец выполнения отчета по лабораторной работе

Федеральное бюджетное государственное учреждение
Высшего образования
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине «Языки программирования»

Преподаватель, доцент

Студентка гр. ИБ-С16
Е.И.Семенова

Санкт-Петербург
2017

Цель работы: приобретение навыков программирования ветвящихся процессов.

Основное задание:

1. Разработать алгоритм, написать программу расчета по двум формулам и сравнить их значения.

$Z_1 = 2 \sin^2(3\pi - 2\alpha) \cos^2(5\pi + 2\alpha)$

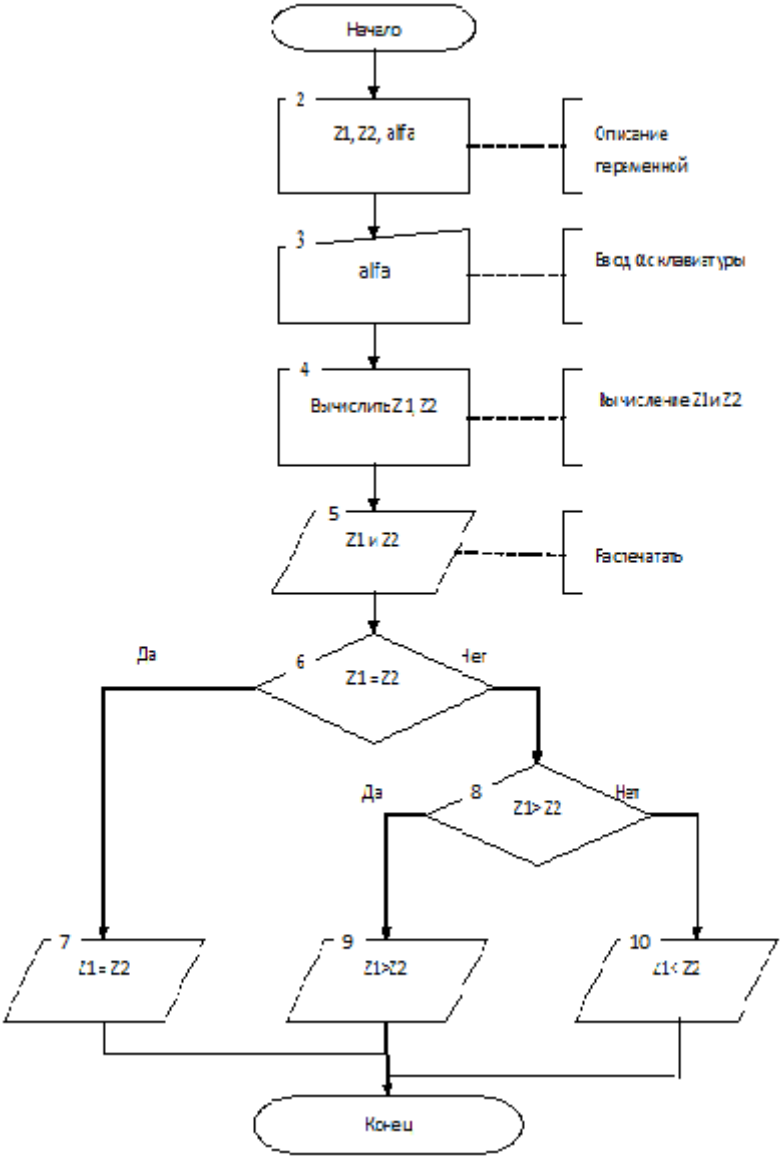
$Z_2 = ¼ - ¼\sin(5\pi/2 - 8\alpha)$

2. Произвести сравнение расчетных переменных для выявления равенства или неравенства этих переменных.

3. Произвести контрольный расчет для $\alpha = \pi/4$.

Таблица имен переменных

Имя в задаче	Имя в программе	Тип данного	Содержательный смысл
Z_1	Z1	Вещественное	Расчетная переменная
Z_2	Z2	Вещественное	Расчетная переменная
α	alfa	Вещественное	Исходное данное


Начало
2
Z1, Z2, alfa
Описание переменной
3
alfa
Ввод α с клавиатуры
4
Вычислить Z1, Z2
Вычисление Z1 и Z2
5
Z1 и Z2
Распечатать
6
Z1 = Z2
Да
Нет
8
Z1> Z2
Да
Нет
7
Z1 = Z2
9
Z1>Z2
10
Z1< Z2
Конец

Трассировочная таблица

№	№ блока	Результат вычислений
1	3	Ввод α = π/4
2	4	$Z_1 = 2 \sin^2(3\pi - 2\alpha) \cos^2(5\pi + 2\alpha) = 0$ $Z_2 = ¼ - ¼ \sin(5\pi/2 - 8\alpha) = 0$
3	5	Вывод Z1 и Z2
4	7	Z1 = Z2
5	9	Z1>Z2
6	10	Z1<Z2
7	11	Конец

Код программы

```
#include "math.h"
#include <stdio.h>
#include <conio.h>
#include <locale.h>
#define pi M_PI
int main()
{
setlocale (0,"rus");
float Z1, Z2, alfa;
printf (" Введите значение альфа :\n");
scanf ("%f",&alfa);
   Z1 = 2*pow(sin(3*pi - 2*alfa),2)*pow(cos(5*pi + 2*alfa),2);
   Z2 = 1/4 - (sin(5*pi/2 - 8*alfa))/4;
```

```
printf ("Z1 = %f\n Z2 = %f\n",Z1,Z2);
if (Z1 == Z2)
printf ("равны");
if(Z1 > Z2)
printf ("Z1 > Z2");
else
printf ("Z2 > Z1");
getch();
return 0;
}
```

Результаты и выводы

Так как результат трассировки совпадает с результатом программы, алгоритм разработан верно.

Учебное издание

ПРАКТИКУМ

по дисциплине

«Информатика и программирование»

Программирование на языке Си и C++

А.Д. Шишкин

Е.А. Чернецова

Печатается в авторской редакции

Подписано в печать 02.04.2020. Формат 60×90 1/16. Гарнитура Times New Roman.

Печать цифровая. Усл. печ. л. 9,9. Тираж 30 экз. Заказ № 872.

РГГМУ, 192007, Санкт-Петербург, Воронежская, 79.