Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

А.Д. ШИШКИН, Е.А. ЧЕРНЕЦОВА

ПРАКТИКУМ

по дисциплине

“ЯЗЫКИ ПРОГРАММИРОВАНИЯ”

Раздел: Программирование С и С++

РГГМУ
Санкт-Петербург
2012

УДК 681.3.657.1

Шишкин А.Д., Чернецова Е.А. Практикум по дисциплине “Языки программирования”. Раздел: Программирование на языке Си и С++. Издание 2-е испр. и доп .- СПб., изд. РГГМУ, 2012.- 72 с.

Рецензент В.В. Фомин, проф. РГГУ им. А.И. Герцена

В лабораторный практикум включены десять лабораторных работ, которые охватывают разделы, относящиеся к языку Си и С++. Работы ориентированы на приобретение студентами навыков программирова-0ния на языке Си и С++ в пакетах TURBO C, Borland C++ и C++Builder.

Содержащиеся в практикуме сведения по теории программирования, методические указания и рекомендации по выполнению работ позволяют использовать его в качестве пособия для закрепления курса лекций. Включен материал по технологии работы в пакете Borland C++ Builder. Дополнительно включены две новые работы, расширены задания по выполнению работ.

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

ISBN 978-5-86813-319-0

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ В ИНТЕГРИРОВАННУЮ СРЕДУ ПРОГРАММИРОВАНИЯ

Назначение и режимы работы ИС TURBO C

Интегрированная среда (ИС) программирования TURBO C предназначена для создания, редактирования и запуска в работу программ, написанных на языке программирования C и C++. В ней пользователь получает ряд услуг, упрощающих процесс создания и отладки программ. В системе имеется развитая система помощи, которая позволяет получить справочную информацию по всем вопросам программирования. Она предоставляет такие возможности, как многооконный режим работы, поддержка работы с мышью, возможность быстрого перехода к другим программам и быстрого возврата, наличие макроязыка редактора и др.

Система имеет два режима работы. Первый, наиболее важный, который используется практически всегда - это режим работы с интегрированной средой TURBO. В ней работа осуществляется с помощью меню. Второй режим работы – использование традиционного метода, когда в начале применяется какой-либо текстовый редактор для создания текстового файла с программой, затем, набираются в командной строке соответствующие команды для компиляции, компоновки и, наконец, выполнения программы.

Запуск среды TURBO

Войти в ИС программирования можно двумя путями:

1) найти на рабочем столе ярлык системы Turbo C и щелкнуть по нему левой кнопкой мыши два раза;

2) на диске найти каталог TURBO C, в подкаталоге BIN выбрать команду

tc.exe

и нажать клавишу <Enter>.

После загрузки ИС она представляется как, представленный на рис. 1, графический образ, состоящий из трех компонентов: строки меню, оконного пространства, строки состояния.

Строка меню предоставляет доступ к командам интегрированной среды. Выбор команд меню, а при их активации, подменю осуществляется с помощью мыши.

Окно редактирования предназначено для ввода и редактирования текста программы. Система TURBO C позволяет держать в памяти несколько открытых окон. При этом активным является только одно. Новое окно открывается с помощью пункта меню File/New строки меню. Для

редактирования уже имеющегося файла необходимо его открыть командой File/Open. В открывшемся окне указать путь и имя файла. Каждое окно имеет рамку, в верхней части которой расположен заголовок окна (имя файла).

ИС позволяет держать в памяти несколько открытых окон, при этом активным является только окно, на которое установлен так называемый фокус ввода.

Окно сообщения предназначено для вывода сообщений о результатах компилирования.

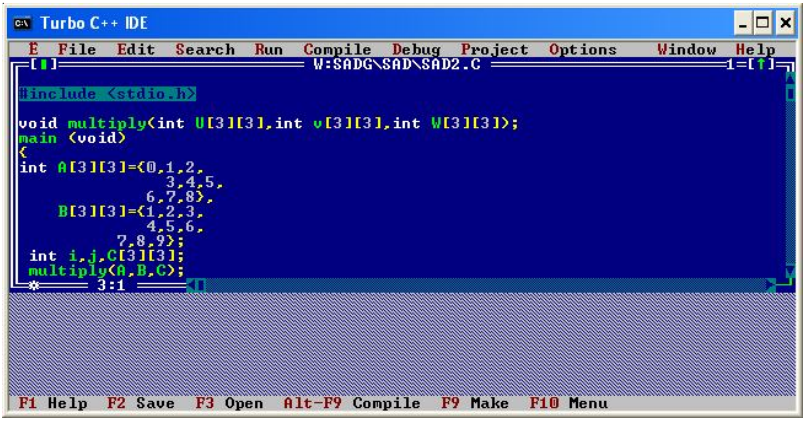

Рис. 1. Строка меню

Приведём краткое описание назначения элементов меню рис. 1:

– E - Системное меню.

– File - загрузка и создание файлов, сохранение внесенных в программу изменений, выход из системы.

– Edit - реализация различных режимов редактирования текста программы (копирование, вставка, удаление) в активном окне.

– Search - поиск фрагментов текста, объявление функций, местоположения ошибок.

– Run – компиляция, компоновка и запуск программы на выполнение.

– Compile – компиляция программы.

– Debug – управление возможностями отладки и запуска программы.

– Project – организация проектов (многофайловых программ).

– Options – настройка интегрированной среды.

– Window – управление окнами.

– Help – Обращение к системе оперативной подсказки.

Выход из системы

Чтобы окончательно покинуть систему можно воспользоваться командой Quit в меню File или нажать комбинацию клавиш ALT+X. Для временного выхода в операционную систему (чтобы выполнить какую-либо команду DOS), оставив при этом программу в памяти машины, используется опция DOS Shell меню FILE. Для возврата в систему требуется набрать в командной строке DOS команду Exit.

Работа с окнами

ИС является многооконной. Главную роль играют окна редактора, но используются также и окна других видов: справочной системы, диалога, контроля, сообщений, наблюдений, проектов и вывода.

Каждое окно имеет свой номер. Переход в другое окно (когда работа ведется с несколькими окнами) осуществляется нажатием клавиш ALT+N, где N – цифра, определяющая номер окна.

Операции с окнами могут выполняться с помощью меню, мыши, либо с помощью «горячих» клавиш. Описание команд меню приведено в [2].

Технология работы в TURBO C

1.Загрузите интегрированную среду Turbo C. Ознакомьтесь с окнами и пунктами главного меню. Откройте новое окно с помощью пункта меню File/New.

В появившееся окно наберите реализуемую программу. Набор текста осуществляется обычным набором средств, знакомых Вам по работе с текстовыми редакторами. Запишите набранную программу под своим оригинальным именем в каталог группы. Для этого выберите в меню команду File/Save as. В появившемся окне наберите путь, имя файла и нажмите клавишу <Enter>.

2. Редактирование теста программы. Откройте следующее окно. Используя директивы редактора для работы с блоками текста, выделите часть текста программы. Для этого установите курсор в начало блока текста и, используя директиву Ctrl+KB, выделите начало блока, а затем, установив курсор в конец блока текста директивой Ctrl+KK – конец блока. Сразу же после выделения текст высветится в инверсном изображении. С выделенным блоком текста можно осуществлять операции копирования, перемещения, удаления и т. д.

Скопируйте выделенный блок в подготовленное Вами пустое окно редактора. Для этого следует выбрать пункт меню Edit/Copy, с помощью которого выделенный текст будет скопирован в карман. Затем следует

перейти в нужное окно редактора и, используя пункт меню Edit/Paste, вставить текст из кармана в окно на позицию, указанную курсором.

Отмените выделение текста директивой Ctrl+KH (повторное выполнение директивы снова выделит текст).

3. Повторите описанный выше процесс копирования с различными участками исходного текста для закрепления навыков копирования. Попробуйте переместить фрагменты текста в выбранную Вами позицию в режиме копирования и переноса. Удалите часть текста, а затем вставьте его.

Выделение блока происходит также в том случае, если при нажатой клавише Shift нажимаются клавиши: стрелки перемещения курсора и клавиши Home, End, PgUp, PgDn. Кроме команд редактирования, удобно использовать следующие директивы для работы с выделенными блоками текста.

Директива	Функция
Ctrl+Del	удаление выделенного блока
Ctrl+Ins	копирование выделенного блока в карман
Shift+Del	перемещение выделенного блока в карман
Shift+ Ins	вставка выделенного блока из кармана

Проведите выделение текста и манипуляции с текстом с использованием указанных клавиш.

4. Использование мыши для работы с текстом. Использование мыши значительно упрощает работы с окнами и текстом. Для перехода из одного окна в другое необходимо щелкнуть левой кнопкой мыши в площади нужного окна. Изменить размеры окна можно «протаскиванием» правого нижнего угла окна. Перемещение окна осуществляется «протаскиванием» в нужное место поля заголовка.

Расположите с помощью мыши два окна таким образом, чтобы они занимали одинаковую площадь и располагались вертикально по всей длине экрана.

Для фиксации курсора в нужной позиции установите в неё указатель мыши и нажмите её левую кнопку. Протаскивание мыши с нажатой левой кнопкой приводит к такому же выделению блока текста, как и использование рассмотренных ранее директив.

Повторите действия выделения, копирования и перемещения текста с использованием мыши и меню Edit.

5. Компиляция и выполнение программы. Откройте окно с исходной программой. Если программа подверглась модификации, то загрузите её снова с помощью меню File/Open. Для компиляции программы необходимо, чтобы текст находился в активном окне. Компиляция осуществляется с помощью пункта меню Compile и может производиться в режимах Compile, Make, Build All. Компиляция в режиме Compile завершается созданием файла с расширением .obj, а компиляция в режимах

Make и Build All заканчивается формированием файла с расширением .exe (выполняемый файл).

Откомпилируйте программу. Если программа не содержит ошибок, то она сразу же может быть запущена на выполнение. В случае наличия ошибок, в нижнем окне будут выданы сообщения об ошибках. Здесь указывается номер строки и характер ошибки. Исправьте допущенные ошибки.

6. Запуск в работу программы. Выберите меню Run и откройте его подменю. Выбор команды Run приведет к немедленному выполнению всей программы. Сначала осуществляется компиляция программы в режиме Make , а затем её запуск на выполнение. Нажатие клавиш Ctrl+F9 из активного окна вызывает аналогичные действия.

Выполнение программы приводит к активизации окна программы, в котором вы видите результаты её работы. Для возвращения в окно программы необходимо нажать клавиши Alt+F5. Повторное нажатие клавиш вернет вас в окно редактора.

Для отладки программы удобно использовать пошаговое выполнение команд программы. Для этого в меню Run выбрать пункт Step Over. Нажимая клавишу F8, можно по шагам выполнить программу. При завершении работы необходимо сохранить файл. Для сохранения нового файла надо выбрать команду File/Save As, при этом необходимо указать путь и имя файла (не более восьми символов). При повторных обращениях и внесении изменений в файл следует набрать команду File/Save.

Технология работы в Borland C++ 5.02

Всем, привыкшим работать в среде Widows, предоставляется возможность программировать в удобной для себя среде Borland C++ 5.02. Запуск среды программирования производится аналогично, описанному выше. После запуска появляется рабочий экран BC, содержащий четыре основные части, показанные на рис. 2: строка меню, окно редактирования, окно сообщений, строка состояния.

Назначение элементов меню аналогично описанному выше, за исключением того, что отсутствует в меню окно Run. Текст программы может создаваться в любом текстовом редакторе, затем обычными опциями копирования и вставки может быть перенесено в поле редактирования. В остальном - технология работы сводится к компиляции, исправлению ошибок и запуску на выполнение.

Для компиляции программы нужно открыть окно меню Project и выбрать одну из команд: Compile, Make all, Build all.

Для запуска программы на выполнение нужно открыть окно меню

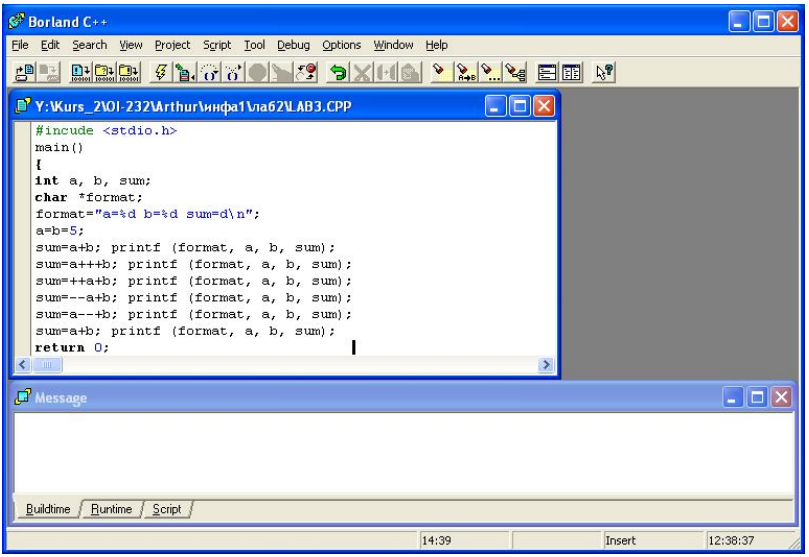

Рис. 2. Вид рабочей области экрана

Debug, в котором следует выбрать опцию Run (сквозное выполнение) или Run to (пошаговое выполнение). При этом будет сформирован исполняемый файл (с расширением .exe).

Для вывода результатов расчётов следует:

- активизировать кнопку «Пуск»;
- выбрать команду «Выполнить» и нажать кнопку «ОК»;
-перевести протягиванием левой кнопкой мыши сформированный исполняемый файл в поле команды «Выполнить».
- запустить на выполнение;
- появившиеся результаты расчетов выделить кнопкой «Выделить все» и кнопкой «Вставить»;
- открыть программу «Блокнот» и командой «Вставить» ввести результат;
- распечатать результаты.

Примечание. Последние три пункта так же можно выполнить после запуска команды Run.

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

Среда Borland C++Builder (читается си плюс плюс билдер) 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 распределёнными базами данных. В качестве языка программирования в среде Borland C++Builder используется C++.

Процесс создания программы в C++Builder состоит из двух шагов: сначала нужно создать форму программы (диалоговое окно), а затем функцию обработки *событий*. Форма *приложения* создаётся путём добавления в неё *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 Radio Button. Результат программа может вывести в поле вывода текста (компонент Label) или в окно сообщения (функции ShowMessage, MessageDlg).

Расчётные 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. 3.

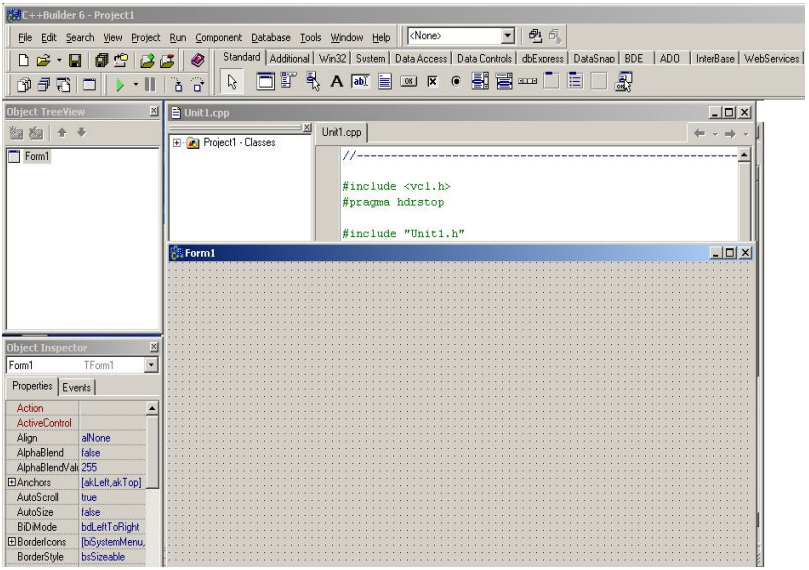

Рис. 3. Основная форма

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. 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 | **Other Application** и на вкладке New появившегося диалогового окна New **Items** щелкнуть на значке **Console Wizard**, показанное на рис.4.

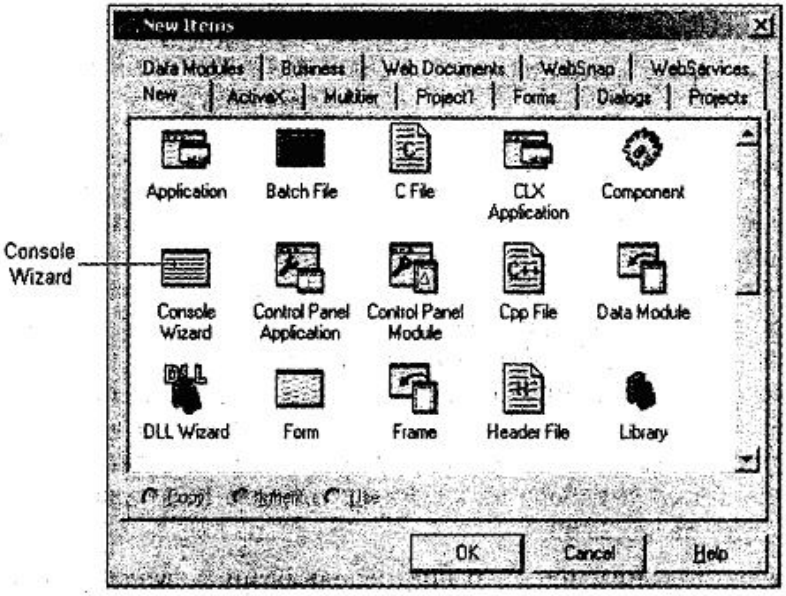

Рис. 4. Задание консольного приложения

В результате этих действий на экране появится окно **Console Wizard**, представленное на рис.5.

В этом окне можно выбрать язык программирования и указать, бу-

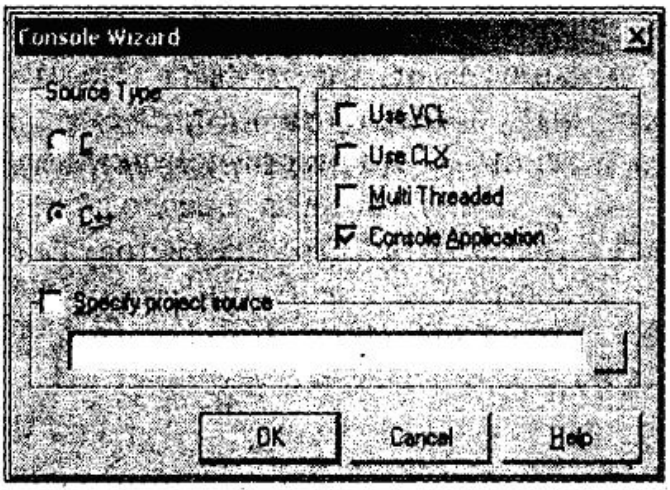

Рис. 5. Задание характеристик консольного приложения

дет ли использоваться та или иная библиотека. После того как будут заданы параметры создаваемого консольного приложения, надо щёлкнуть 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.6, в котором находится шаблон консольного приложения - функция main ().

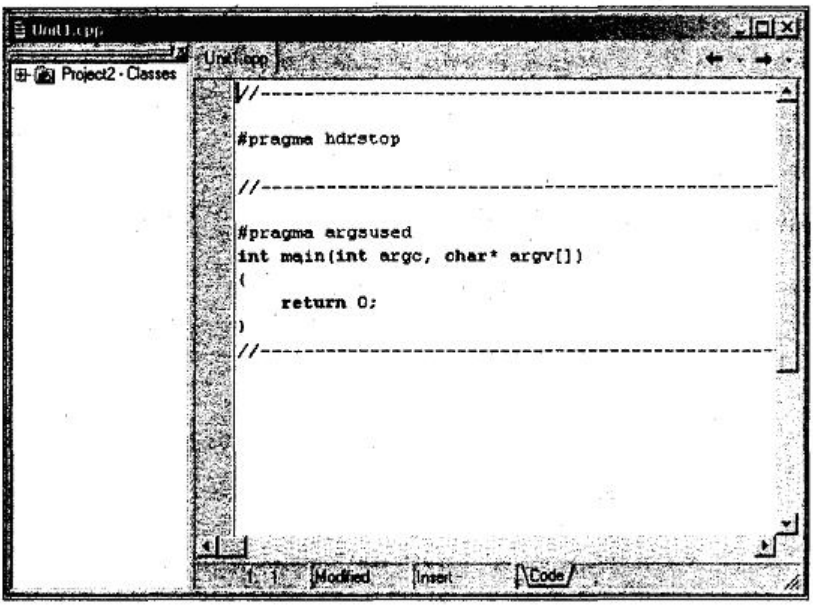

Рис. 6. Окно ввода и редактирования программы

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 [5].

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).

Консольное приложение подобного типа можно создать также в среде программирования **Visual C++**. Заготовка файла реализации представляется в виде:

```
    //Console.cpp: Defines the entry point for the
console fpplication.
    #include “stdafx”
    int main(int argc, char *argv[])
    {
      return 0;
    }
```

Данное простое консольное приложение создаётся мастером при выборе опции **A simple application.** Компиляция, запуск и выполнение выполняется аналогично, описанным выше действиям.

Лабораторная работа №1

Базовые операции языка Си

Цель работы: приобретение навыков программирования линейных процессов; освоение функции ввода/вывода данных, оператора присваивания.

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. Алгоритм программы с линейной структурой

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

Тип Имя_Переменной;

Объявление переменной можно совместить с её 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». 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 её адреса в качестве параметра функции scanf() является типичной ошибкой начинающих программистов).

Для иллюстрации действия указанных операций рассмотрим следующий пример.

Пример № 1: Введём программу, печатающую на экране: "Сейчас 2012 год".

```
#include<stdio.h>
/* пример к лаб. работе №1*/
main ( )
{     //  Начало тела главной функции
int year, month;   // Объявление переменных
year = 2012; //Инициализация переменных
month=10;
printf("Сейчас %d год %d месяц\n",year,month);  // вывод
}      // 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. С помощью этой конструкции можно выделить целый блок программы. 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 = 2012; 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(). В программе также использованы четыре комментария. Двойной слеш комментирует не более одной строки.

Если программа не содержит ошибок, то после компиляции и выполнения её на экране появится сообщение: Сейчас 2000 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 1. Введем строковую переменную, например, «Имя» и инициализируем ее.

```
#include<stdio.h>
/* пример к лаб. работе №1*/
main()
{
int year, month;
year = 2012;
char name= "Ivan";
printf("Сейчас %d год\n",year);
}
```

Используйте для вывода новой переменной оператор printf() и спецификатор вывода %s. С помощью функции scanf () введите новое имя и распечатайте его.

Пример № 2. 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.

```
/* ПРОГРАММА
Вычисление длины окружности */
   #include <stdio.h>
    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 имя 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.

Пример № 3.

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

Первый оператор суммирует a и b, затем результат присваивает 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.

Задание для выполнения работы

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

2) ввести программу примера № 2, провести отладку и запустить на выполнение.

3) модифицировать программу примера № 2 таким образом, чтобы она вычисляла длину окружности и площадь круга для любых радиусов.

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

Пример № 4:

```
#include <stdio.h>
#include <conio.h>
main ( )
{
 int a, b, sum;
    a=b=5;
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();
}
```

5) скопировать пример 4, открыть ИС Borland C++ 5.0, вставить файл, произвести запуск и вывести результат в блокнот.

Содержание отчёта:

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. Как выполняется операция присваивания?
5. Какой спецификатор применяется для ввода и вывода строки символов?
6. В каком заголовочном файле содержится функция getch()?

Лабораторная работа № 2

Организация циклических вычислений

Цель работы: Изучение 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.

Простейшие примеры 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 необязательно использовать целочисленный счетчик. Приведем фрагмент программы, выводящий на экран буквы русского алфавита:

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

Следующий фрагмент программы

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

будет вводить символы 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 избежания "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*/
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>
    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

$$S = \sum_{i=1}^{n} \ln(kx)/k^2$$

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 */
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:

1) $S = \sum_{k=1}^{5} (x+1)k * \cos(x/k)/(k+1)!,\ x=\pi/2;$

2) $S = \sum_{k=1}^{5} (\sin x)^k / k\ !,\quad x=\pi/4;$

3) $S = \sum_{k=2}^{8} (\cos x)^k/k!,\quad x=\pi/4;$

4) $S = \sum_{i=1}^{7} (\sin x)^i/ i!,\quad x=\pi/4;$

5) $S = \sum_{i=2}^{7} (e^x - i)/(i+1)!$, x=1,0;

6) $S = \sum_{i=1}^{20} (e^{x/i} + e^{-x/i})/(i+2)^4$, x=1,0;

7) $S = \sum_{i=1}^{10} \ln(x*i)/(i+2)$, x=2,0;

8) $S = \sum_{i=-5}^{5} x^{i+5}/(i+7)!$, x=2,0;

9) $P = \prod_{i=5}^{10} x^i/i!$, x=5,0;

10) $P = \prod_{i=5}^{10} (x+i)^{i+1}/(i-2)!$, x=4,0;

11) $P = \prod_{i=1}^{N} (x^i + \frac{1}{\sqrt[4]{i}})$, 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) $S = \sum_{i=1}^{N} \frac{i!}{(a+b)^i}$, a=2, b=3,5;

15) $F = \sum_{i=1}^{n} \frac{\cos(x)^i}{i!}$, $x = \pi/3$;

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) $S_1 = \sum_{i=1}^{N} (\sin x)^i \; / \; \sum_{i=1}^{N} \sin x^i$,

21) $S = \sum_{k=1}^{N} a_k$, где, $a_k = \sqrt{x^2 + \sin^2(kx)/4}$

N – натуральное, а x – вещественное;

22) $P = \prod_{k=1}^{N} (1 - \frac{f^2}{2k+1})$, N – натуральное, f – вещественное;

23) $S = \sum_{k=2}^{N} k^2 \ln(1 + k^2)$, N >2;

24) $S = \sum_{i=1}^{N} \sum_{k=1}^{M} \sin(\frac{\pi}{4} i + \frac{\pi}{8} k^2)$,

25) $S = \sum_{i=1}^{N} \prod_{k=1}^{N} \frac{i + x}{k}$, N - натуральное, x - вещественное;

26) $S = \sum_{k=N-5}^{N} (k - \ln(1 + 2k))$, N >5;

27) $P = \prod_{i=1}^{N} \frac{1}{\sum_{k=0}^{I} (f + k)}$, N – натуральное, f – вещественное;

28) $S = \sum_{k=1}^{N} \frac{\sum_{i=1}^{N} \sin(0.01 * k * i)}{k!}$;

29) $S = \sum_{k=1}^{N} (k^3 * \sum_{i=1}^{k} (k - i^2))$;

30) $S = \sum_{i=1}^{N} (-1)^{i+1} \frac{x^{2i}}{i!}$, N – натуральное, x- вещественное;

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?

Лабораторная работа № 3

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:

$$AX^2 + BX + C = 0 \qquad (3.1)$$

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 (3.1) 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:

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.

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:
// подразумевается, что комплексных корней нет
```
if(D>0)
printf(“Решение есть \n”);
```
При необходимости в комбинации с *if* можно использовать ключевое слово *else*, позволяющее выполнить альтернативный оператор, или блок операторов, если условие неистинно.
Пример:
```
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:
```
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 n: оператор или группа операторов блока n;
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 на следующий (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:

```
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”);
                        :
                        :
```

```
break;
        default: puts (“ Ошибка \a \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* ?

Лабораторная работа № 4

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 4.1. 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*(), инициализированной директивой препроцессора

```
#define   RND((float)rand()/32768.0)
                .....
                x[i][j]=RND*( целое число);
```

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]=RND*( целое число);
```

или *switch(w),* где *w* равно 1 или 2.

Воспользуйтесь указаниями лабораторной работы № 3.

Таблица № 4.1

Варианты одномерного массива

Вариант	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 чётными 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 нечётными 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.

Продолжение табл. № 4.1

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 модулю 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 мас-сива; 2) сумму элементов массива, расположенных после пос-леднего 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.

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,j) - того элемента в памяти машины?

Лабораторная работа № 5

Обработка двумерных массивов

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

Основное задание

Составить программу работы с матрицей, для этого:

- организовать ввод элементов `a[i][j]` матрицы `M[N][N]`, 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 преобразование матрицы, одним из следующих вариантов.

1.Транспонировать матрицу.

2. Возвести матрицу в квадрат.

3. Умножить *к*-ю строку исходной матрицы на сумму элементов главной диагонали.

4. Прибавить к к- 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. Вычислить *ij*- минор матрицы.

21. Найти след матрицы.

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

Дополнительное задание

Дополните Вашу программу функциями, определяющими активное текстовое окно.

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;
}
```

Здесь 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

```
#define   RND ((float) rand () /32768.0)
  .....
x[ i ][ j ] =RND* (целое число);
```

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

for (инициализация_1; условное выражение_1; конечное выражение_1)

for (инициализация_2; условное выражение_2; конечное выражение_2)

оператор или блок операторов // двойной цикл.

В круглых скобках содержится три выражения, разделенные символом точка с запятой. Первое из них служит для инициализации счета. Она осуществляется только один раз, когда цикл for начинает выполняться. Второе выражение – для проверки условия (она производится перед каждым возможным выполнением цикла). Когда выражение становится ложным (или в общем случае равным нулю), цикл завершается. Третье выражение вычисляется в конце каждого выполнения тела цикла.

Пример:

```
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:

```
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:

```
Char answer;  int N;
       do
          {
           :
         /*группа операторов */
          printf ("Хотите завершить задачу ?");
           answer= getch();
           }
        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 лабораторных работах 3 и 4.

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. Опишите примененный Вами алгоритм преобразования матрицы.

Лабораторная работа № 6

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 текст программы, состоящей 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 ()*, предназначенными 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()* необходимыми операторами и операндами.

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 её элементов и осуществляет ввод их значений в память ПЭВМ.

```
int ** input(int n, int m)
{
int i,j;
int **a;
//Выделение динамической памяти для элементов матрицы
a= (int**)malloc(n*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("\nВведите элемент матрицы  A(%d,%d) элемент
массива:" ,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 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 также 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 внесите 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<alloc.h>
int **input (int,int);
//Объявление функций обработки, использующих функцию input
int fproi(int**,int**,int,int);
……
void ouput( int**,int,int);
void main(void)
{
int m,n;
int **p, **q;
clrscr();
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 лабораторной работы № 5.

Варианты заданий:

1. T,U,S
2. MQB
3. S,M,T
4. B,U,C
5. Q,S,N
6. C,Q,T
7. U,T,N

8. S,N,U
9. C,N,T
10. M,N,Q
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.

В вариантах заданий использованы следующие обозначения:

T- транспонирование матрицы.
C- вычисление следа матрицы.
U- умножение матрицы на число.
M- перемножение матриц.
S - сложение матриц.
B- вычитание матриц.
N- вычисление нормы матрицы.

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?

Лабораторная работа № 7

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()* берёт внешнее представление файла - его физическое имя - и ставит ему в соот-

ветствие 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*>.

Чтобы продемонстрировать работу с этими функциями, рассмотрим простой пример.

Пример 1.

```
# include<stdio.h>
void main()
{
     int n;
     char *str, str1[50],ch;
     FILE *fp;
     // Заполняем файл
     fp=fopen("D:\ex.txt","w+a");
     puts("Введите целое число") ;
     scanf("%d",&n);
     fprintf(fp, "%d\n" , n);
     puts("Введите символ") ;
     ch= getch ();
     putc ( ch,fp);
     puts("Введите строку\n") ;
     scanf("%s", str);
     fputs ( str, fp);
     fclose ( fp);
     // Читаем из файла
```

```
printf("распечатка содержимого файла");
  if(( fp= fopen ("D:\ex.txt","r")) != NULL)
     {
       fscanf (fp, "%d" ,&n);
       printf(  "n= %d\n" , n);
       ch=getc ( fp ) ;  putchar (ch );
       fgets ( str1, 50 , fp); puts(str1 );
       fclose ( fp );
       }
else printf ("\n Нельзя открыть  файл для чтения !");
}
```

Второй параметр функции *fgets()* – 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.

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 от начальной точки, начальная точка****)***.

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.

Так как язык Си связан с операционной системой *UNIX*, то в системе Borland C++ создана вторая система ввода/вывода. Эта система соответствует стандарту *UNIX*. Прототипы функций находятся в файле IO.H. Этими функциями являются

read() – читает буфер данных;
write() – пишет в буфер данных;
open() – открывает файл;
close() – закрывает файл;
lseek() – поиск определенного байта в файле;
unlink() – уничтожает файл.

Пример 2. Написать функцию, которая читает любое число байтов из любого места файла.

```
int get (int fd, long pos, char *buf, int n);
{
  if ( lseek(fd, pos, 0)>=0) return read (fd,
duf, n);
   else return –1;
}
```

Задание на выполнение работы

1. Создайте текстовый файл в директории группы.
2. Введите программу примера 1. Выполните действия с заполнением и считыванием символов и строк.
3. Составить программу обработки текстового файла в соответствии с вариантом задания, представленного в табл. 7.1.

Таблица 7.1

Варианты обработки текстового файла

Вариант	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.

Продолжение табл. 7.1

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 начина-ются 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 чём состоит различие команд *fputs()* и *fgets()*?

Лабораторная работа №8

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 учёта 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:

```
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 К при использовании метода деления пополам максимальное время поиска примерно пропорционально целой части двоичного логарифма К.

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 К.

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 вспомогатель-

ная части. Вспомогательная часть используется для организации связей между элементами списка и содержит одно или несколько полей ссылочного типа.

В программе учёта 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 вкладчика 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 A, 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 A имеет в системе учёта 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 учёта 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 показан на рис.8. 1. 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 (рис. 8.2), когда каждый элемент, кроме первого и последнего, имеет две ссылки : на предыдущий элемент и следующий элемент и т.д.

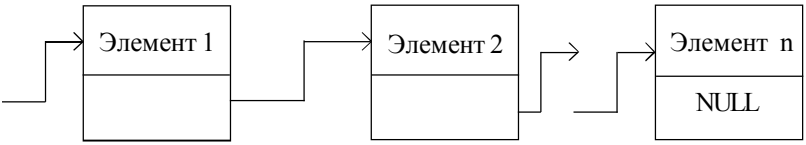

Рис.8.1 Связной список

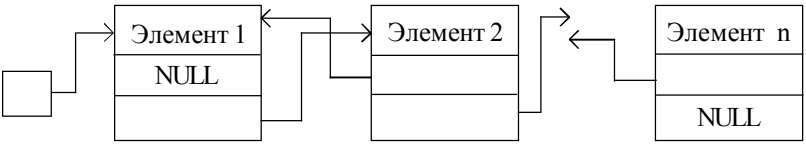

Рис.8.2. Двунаправленный список

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 счётчик 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 спис-

ка 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 (structelement));
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 для выполнения работы

Составить программу обработки списка в соответствии с номером задания. Вид списка определяется номером строки табл. 8.1, а вид обработки – номером столбца, в котором стоит номер задания в табл. 8.2. 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 8.1

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
Линейный циклический	16	4	7	17	22	11	13	2	15	3

Таблица 8.2

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.

Продолжение табл. 8.2

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.
19	Объединить два списка - второй добавить в хвост первого (циклические списки разрываются в произвольном месте).
20	Найти элемент с заданным значением информационного поля (из функции обработки возвращается значение ссылочной переменной, указывающей на первый элемент).
21	Удалить заданный элемент из списка. Элемент задан ссылочной переменной.
22	Удалить заданный элемент. Элемент задан значением поля; удаляется первый элемент
23	Удалить заданный элемент. Элемент задан значением поля; удаляются все такие элементы.
24	Проверить, входит ли заданный элемент в список?
25	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.
4. Разработанную программу.
5. 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?

Лабораторная работа № 9

Вывод графиков функций

Цель работы: изучение методов построения графиков функций в графическом редакторе языка Си; составление программы для расчёта и вывода в графическом режиме заданной функции. Результаты расчёта 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 $y=0{,}5x^2+4x-3$.

```
#include<graphics.h>
#include<math.h>
```

```
#include<stdlib.h>
#include <conio.h>
#include<stdio.h>
    void zaggraf();
   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.

```
   void main(void)
   {
    zaggraf(); // загрузка графики
    grafik();  //  ввод функции графика
      grid();              //  ввод функции сетки
    getch();
    closegraph();
   }
```

Задание для выполнения работы

Разработать программу построения графика одной из функций. Варианты заданий приведены в табл. 9.1.

Таблица 9.1

Варианты 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 $y=\exp(x^2)$	[-1.3- 1.3]
6	Кардиоида x=4cos(t) (1+cos t) y=4sin(t)(1+cos t)	$[0, 2\pi]$
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	$[-2\pi, 2\pi]$
10	Циссоида $x=5t^2 /(1 + t^2)$ $y=5t^2 /(1 + t^2)$ t=tg(f)	$[-\pi/4, \pi/4]$
11	Тригонометрическая функция $y=\cos(x^2)$	$[-2\pi, 2\pi]$
12	Строфоида $x=4(t^2-1) /(t^2+1)$ $y=4t(t^2-1) /(t^2+1)$ t=tg(f)	$[-\pi/2.5, \pi/2.5]$
13	Тригонометрическая функция y=tg(x) – 2x	$[-\pi/2.5, \pi/2.5]$
14	Астроида $x=3.5\cos^3(t)$, $y=3.5\sin^3(t)$	$[0, 2\pi]$
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 $t = [\pi, 2\pi]$
17	Логарифм y=ln(x+2)	-1.5, 5
18	Гопоциклоида x=2acos(f) + acos(2f), y=2asin(f) + acos(2f)	$[-2\pi, +2\pi]$ a=1.5
19	Арктангенс y=3 arctg(x)	[–5, 5]
20	Эвольвента окружности x=a cos(f) + af sin(f) y=a sin(f) – acos(f)	$f=[-90^0,90^0]$ a=1.5
21	Дробно- рац. нелинейная $y=ax+c/ bx+cx^2$	[0.18, 3] a=1, b=2, c= -0.5

Продолжение таблица 9.1

22	Лемниската x=rcos(f) y=rsin(f), r=asqrt(2cos(2f))	$[-\pi, \pi]$
23	Локон Аньези y= $a^3/x^2 + a^2$	[–5, 5], a=2
24	Архимедова спираль x=rcos(f) y=rsin(f) ,r=af	$[-60^0, 60^0]$ a=1.5
25	Трохоида (Удлиненный цикл) x=a(1+b sin(f)) y=a(1-b cos(f))	$[-2\pi, 4\pi]$ a=1.5, b=1.3
26	Гиперболическая спираль x=(a cos(f))/f y=(a sin(f))/f	$[0, \pi]$ a=3
27	Удлиненная эпициклоида x=5cos(f) – 2cos(5f) y=5sin(f) – 2sin(5f)	$[0, \pi]$
28	Логарифмическая спираль x=rcos(f), y=rsin(f) r=aexp(bf)	f =$[0, 4\pi]$ a=1.3, b=0.5
29	Удлиненная гипоциклоида x=4cos(f) + 2cos(4f) y=4sin(f) – 2sin(4f)	f = $[0, 2\pi]$
30	Конхоида Никомеда x=a+bcos(f) y=atg(f) + bsin(f)	[-1.5, 1.5] a=1, b=2
31	Улитка Паскаля $x=2\cos^2(t) + 3\cos(t)$, y=2cos(t)sin(t) + 3sin(t)	$[0, 2\pi]$

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. Распечатка графика функции.

Лабораторная работа № 10

Освоение работы в среде С++

Цель работы: приобретение практических навыков работы с объектно-программируемым языком С++

Введение в основы языка C_{++}

Объектно-ориентированное программирование включает элементы структурного программирования, дополняет его новыми идеями, которые переводят в новое качество подход к созданию программ.

Наиболее важное понятие языков объектно-ориентированного программирования - это понятие объекта. Обьект - логическая единица, которая содержит данные и методы обработки этих данных - функции. Внутри объекта данные и функции могут быть частными (приватными, *private*), защищенными (*protected*) и общими (*public*). Можно сказать, что объект - это переменная определенного пользователем типа. 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 Си, 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.

В языке С++ для ввода и вывода данных используются, так называемые, перегружаемые функции. Они имеют прототипы в заголовочном файле *iostrem.h*.

Пример 1.

```
#include< iostrem.h>
void main(void)
{
cout<<" Язык программирования С++"; // оператор вывода данных работает как функция printf().
cin>>m; // работает как функция scanf().
cout<<m<<"\n"; //вывод нескольких операндов        }
```

Обе функции не требуют задания спецификаторов полей данных. Можно использовать стиль комментариев языка Си. Можно использовать функцию *printf()* для вывода и функцию *scanf()* для ввода чисел.

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*.

Пример 2. Объявление класса

```
//Объявим класс queue
class queue {
private: //режим доступа частный
int q[100];
int i,j;
public: //режим доступа открытый
void func1(void);
int func2(int i);
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 формой

```
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 3. Объявление наследуемого класса

```
class queue1: public queue {
 //Объявлен класс queue1 наследователь queue
int sum;
public:
int get_sum(void);
};
```

Задание на программирование

1. Составить программу ввода и вывода матрицы на базе функций языка С++.
2. Ввести программу, реализующую очередь (пример 4). Найти и исправить ошибки.
3. Дополнить класс queue защищенной функцией для вычисления факториала и вызвать её в главную функцию.
4. Создать наследующий класс и включить его в программу примера 4.

Пример 4.

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

В наследуемом классе

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

Создаем объект: queue1 obj.

Описываем функции наследующего класса.

```
int queue1::get_sum(void)
   {
   sum=0;
   for(int i=rloc;i<sloc;i++)
   sum+=q[i];
   return sum;
   }
   void queue1::show_sum(void)
```

```
{cout<<"summa ochered="<<sum<<"\n"; }
```

Дополняем главную функцию программы фрагментом вызова функций.

```
queue1 obj;
  obj.init();
  for(int i=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();
   }
```

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?

Приложение

```
// консольное приложение программы «Угадай число»[5]
#pragma hdrstop
#include <stdio.h>
# include <conio.h> // для доступа к getchf)
#include <stdlib.h> // для доступа к srand(), rand()
#include <time.h> // для доступа к time_t и time()
char* rus (char* st) ; // преобразует ANSI-строку в строку 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) ) ; // инициализация ГСЧ
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 ( “ Вы проиграли! “ ) ) ;
else {
puts ( rus ( “Вы проиграли . “ ) ) ;
printf ( rus ( “Компьютер \”задумал\” число %d”) , comp) ;
}
printf (rus (“\nДля завершения нажмите любую клавишу. “ ) ) ;
getch( );
return 0;
}
/* Функция rus преобразует ANSI -строку в строку ASCII и может
использоваться для вывода сообщений на русском языке в консольных
программах. */
char* rus (char* st)
{
unsigned char* p = st;
```

```
/* при объявлении символов как char русские буквы кодируются
отрицательными числами */
while ( *p)
{
if (*p >= 192) // здесь русская буква
if (*p <= 239) // А, Б, ... Я, а, б, ... п
*p -=64;
else // р ... я
*p -= 16;
p++;
 } return st;
}
```

Литература

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, 2001.-288 с.
3. Культин Н. Б. С/С++ в задачах и примерах – СПб.: БХВ- Петербург, 2011.
4. 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., исправ.- М.: Горячая линия – Телеком, 2000.
5. Культин Н. Б. Самоучитель С++ Builder.- СПб.: БХВ-Петербург, 2004.- 320 с.
6. С/С++. Структурное программирование: Практикум/ Т.А. Павловская, Ю.А. Щупак.- СПб: Питер, 2003.-240 с.
7. Секунов Н.Б. Самоучитель Visual C++.- СПб.: БХВ - Петербург, 2004.- 960 с.

Учебное издание

Анатолий Дмитриевич Шишкин,
Елена Анатольевна Чернецова

П Р А К Т И К У М

по дисциплине

“ЯЗЫКИ ПРОГРАММИРОВАНИЯ”

Раздел: Программирование С и С++

Редактор О.С. Крайнова
Компьютерная вёрстка К.П. Ерёмин

ЛР № 020309 от 30.12.96

Подписано в печать 17.05.2012. Формат 60x90 1/16. Гарнитура “Таймс”
Печать цифровая. Усл. печ. л. 4,5. Тираж 300 экз. Заказ № 92
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр. 98.
Отпечатано в ЦОП РГГМУ