**КОСТАНАЙСКИЙ ГОСУДАРСТВЕННЫЙ**

**ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ**

**ЕСЛЯМОВ С.Г.**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ
ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ С++**

УЧЕБНОЕ ПОСОБИЕ

**Костанай**

**2016**

УДК 004.438

Е82

*Допущено Ученым советом Костанайского государственного педагогического института в качестве учебного пособия для бакалавров, обучающихся по образовательным программам физико-технических специальностей и специальностей информационных технологий*

*Рецензенты:*

***Баймухамедов М.Ф.****, доктор технических наук, профессор, проректор по науке и международным связям Костанайского социально-технического университета им. ак.З.Алдамжар.*

***Сухов М.В.****, кандидат технических наук, декан естественно-математического факультета Костанайского государственного педагогического института.*

**Еслямов С.Г.**

**Е82** Лабораторный практикум по программированию на языке C++: учебное пособие / С.Г.Еслямов. – Костанай: КГПИ, 2016. – 90 с.

ISBN 978-601-7839-64-2

В учебном пособии изложены базовые сведения по программированию на языке высокого уровня С++. Также излагаются классические модели, методы и алгоритмы языка программирования высокого уровня С++, дается строгое изложение основ теории программирования на этом языке, приводятся примеры программ на языке программирования С++. Главное внимание уделяется объяснению того, как использовать на практике полученные знания. Курс разбит на лекции, включающие теоретический материал и лабораторные работы.

Для работы с учебным пособием требуется знание основ алгоритмизации и навыков программирования на языке программирования Паскаль. Пособие рассчитано на обучающихся высших учебных заведений, специализирующихся по физико-техническим специальностям. Предложенные методики программирования полезны и для обучающихся высших учебных заведений, изучающих дисциплины образовательных программ педагогического направления.

УДК 004.438

Все права защищены: Никакая часть данной книги не может быть воспроизведена в любой форме без письменного разрешения владельцев авторских прав.

ISBN 978-601-7839-64-2

 © С. Еслямов, 2016

**ВВЕДЕНИЕ**

*Цель лабораторного практикума*

Лабораторный практикум выполняется при изучении курса «Программирование на языке C++» и имеет целью выработку у студентов навыков в трех направлениях:

* Обучение основам языка программирования C++;
* Освоение технологии разработки программ и выработки у студентов навыков программирования с использованием языков программирования С и С++;
* Применение языков программирования С/C++ как инструмента для системного программирования.

**Язык С++ как инструмент системного программирования**

Главным качеством языка C+, которое делает его именно языком системного программиста, является то, что «C++ – это язык относительно «низкого уровня». Это означает, что C++ имеет дело с объектами того же вида, что и большинство ЭВМ, а именно, с символами, числами и адресами. Они могут объединяться и пересылаться посредством обычных арифметических и логических операций, осуществляемых реальными ЭВМ». Система программирования C++ при представлении данных не вносит никаких дополнительных структур памяти, которые не были бы «видны» программисту. Так, например, внутреннее представление массивов в языке C++ полностью совпадает с внешним: массив – это только последовательность слотов в памяти. Отсутствие специального дескриптора массива, с одной стороны, делает невозможным контроль выхода индексов за допустимое границы, но с другой, уменьшает объем памяти программы и увеличивает ее быстродействие за счет отсутствия в памяти дескриптора и команд проверки индекса при каждом обращении к элементу массива. Это общий принцип C-программ: программист имеет максимальные возможности для разработки эффективных программ, но вся ответственность за их правильность ложится на него самого. Поэтому отладка программ на языке C++ – непростой процесс, C-программы чаще «зависают» и выдают результаты, которые не всегда воспроизводятся и которые труднее объяснить, чем программы на других языках.

Чрезвычайно важным свойством языка C++, которого нет в других языках, является адресная арифметика. Над данными типа «указатель» возможны арифметические операции, причем в них могут учитываться размеры тех объектов, которые адресуются указателем. Другое свойство указателей – их явная связь с конструкциями интеграции данных (массивы, структуры, объединения) и возможность подмены операций индексации и квалификации операциями адресной арифметики. За счет указателей программист имеет возможность удобным для себя способом структурировать адресное пространство программы и гибко изменять это структурирование.

Свойством, которое вытекает из общих принципов построения языка C++, является слабая защита типов. В языках с сильной защитой типов (Pascal) для каждого типа данных определен набор доступных операций и компилятор запрещает применение к типу непредусмотренных для него операций и смешивание в выражениях данных разных типов. В C++ определен богатый набор правил преобразования типов по умолчанию, поэтому почти любая операция может быть применена к почти любому типу данных и выражения могут включать данные самых разных типов.

Еще некоторые средства языка не ориентированы непосредственно на низкоуровневое системное программирование, но могут быть очень полезны при разработке системных программ:

* обязательной составной частью языка является препроцессор. C++ не поддерживает сложных структур данных, но позволяет программисту определять свои типы. Включение в программу описания таких типов средствами препроцессора позволяет обеспечить однозначную интерпретацию типов во всех модулях сложного программного изделия;
* процедурно-ориентированный язык C++ вместе с тем представляет все кодовые составляющие программы в виде функций. Это дает возможность применять язык C++ как инструмент функционально-ориентированного программирования, как, например, язык LISP.

Важное качество языка C++ – высокая эффективность объектных кодов C-программы как по быстродействию, так и по объему памяти. Хотя это качество обеспечивается не столько свойствами самого языка, сколько свойствами системы программирования, традиция построения систем программирования C++ именно такова, что они обеспечивают большую, эффективность, чем, например, Pascal. Это связано еще и с «родословной» языков. Pascal возник как алгоритмический язык, предназначенный в своей первой версии не для написания программ, а для описания алгоритмов. Отсюда Pascal-трансляторы строились и строятся как синтаксически-ориентированные трансляторы характерно компилирующего типа: транслятор выполняет синтаксический анализ программы в соответствии с формально представленными правилами, а объектные коды формируются в основном в виде обращений к библиотечным процедурам, которые реализуют элементарные функции. Язык C++ ведет свое происхождение от языка BCPL, который был языком Макроассемблера. Отсюда и объектный код C-программы строится как последовательность машинных команд, которая оптимизируется для каждого конкретного выполнения данной функции.

***Общие указания к выполнению лабораторных работ***

**Порядок выполнения работ**

Для выполнения всех лабораторных работ предлагается единый порядок, предусматривающий следующие шаги.

* Ознакомиться с постановкой задачи и исходными данными.
* Определить вариант индивидуального задания.
* Сконструировать структуру программы.
* Составить текст программы.
* Набрать текст программы.
* Выполнить компиляцию программы.
* Провести анализ и исправление обнаруженных синтаксических ошибок в тексте программы.
* Получить решение (изображение) и, в случае обнаружения логических ошибок, определить и устранить их.

**Содержание отчета**

Отчет оформляется по каждой лабораторной работе и состоит из следующих разделов.

* Лекция по теме лабораторной работы.
* Цель работы.
* Индивидуальное задание.
* Описание структур данных и алгоритмов
* Результаты работы программы.
* Интерпретация результатов.

Лабораторная работа №1

“Операторы ввода и вывода в языке программирования С++”

**Цель работы:** изучить операторы ввода и вывода, форматы, используемые в этих операторах. Оформить законченную программу с применением этих операторов.

1.Теоретические сведения

* 1. Структура программы

## В языке С++ любая программа состоит из одной или более функций, задающих действия, которые нужно выполнить. Выполнение любой программы начинается с функции main. Далее идет текст программы, заключенный в фигурные скобки. Таким образом, структура программы имеет вид:

**main ( )**

**{**

 **// Тело программы**

**}**

В самом простом случае функция main не имеет аргументов, поэтому в скобках ничего не содержится. Для работы программы, обеспечивающей ввод и вывод информации, перед функцией main необходимо поместить строку:

**#include <stdio.h>**

* 1. Алфавит языка и типы данных

Алфавит языка включает латинские прописные и строчные буквы, цифры и специальные знаки. К последним относятся: . (точка), , (запятая), ‘ (апостроф), : (двоеточие) и др.

Важным понятием языка является идентификатор, который используется в качестве имени объекта, например, переменной, функции и т.п. Идентификатор может содержать до 32 символов и состоит из букв и цифр, но начинается обязательно с буквы. Строчные буквы отличаются от прописных, поэтому идентификаторы SIGMA и sigma считаются разными.

В языке С++ существует несколько типов данных. Каждый тип данных определяется одним из следующих ключевых слов:

1. int (целый) – задает значения, к которым относятся все целые числа. Диапазон возможных целых значений лежит в пределах от –32768 до 32767, переменная типа int занимает 16 бит;
2. short (короткий целый) – соответствующие объекты не могут быть больше, чем int, переменные этого типа занимают 16 бит;
3. long (длинный целый) – соответствующие объекты не могут быть меньше, чем int. Переменная типа long занимает 32 бита и позволяет представить целые числа от –2147483648 до 2147483647;
4. char (символьный) – задает значения, которые представляют различные символы;
5. unsigned (беззнаковый) – в языке C++ можно объявлять некоторые типы (char, short, int, long) беззнаковыми с помощью модификатора unsigned (например, unsigned short). Это значит, что соответствующие переменные не будут иметь отрицательных значений. В результате они могут принимать большие положительные значения, чем переменные знаковых типов. В случае типа int объявления вида «unsigned int a;» можно записать «unsigned a;»;
6. float (вещественный) – задает значения, к которым относятся вещественные числа, имеющие дробную часть, отделяемую точкой. Вещественные числа могут быть записаны также в экспоненциальной форме. Например, -1.58е+2 (что равно -1,58·102 ). В языке C++ переменная типа float занимает 32 бита. Она может принимать значения в диапазоне от +3.4е-38 до +3.4е+38;
7. double (двойная точность) – определяет вещественные переменные, занимающие в два раза больше места, чем переменная типа float. Переменная типа double занимает 64 бита. Она может принимать значения в диапазоне от +1.7е-308 до +1.7е+308.

1.3. Ввод и вывод информации

1.3.1. Форматный вывод

Вначале рассмотрим функцию, определяющую форматный вывод:

**printf("управляющая строка", аргумент1, аргумент2, ... );**

Управляющая строка содержит объекты трех типов: обычные символы, которые просто выводятся на экран дисплея, спецификации преобразования, каждая из которых вызывает вывод на экран значения очередного аргумента из последующего списка и управляющие символы-константы.

Каждая спецификация преобразования начинается со знака % и заканчивается некоторым символом, задающим преобразования.

Символ преобразования связан с типом переменных. приведем символы преобразования:

1. d – значением аргумента является десятичное целое число;
2. o – значением аргумента является восьмеричное целое число;
3. x – значением аргумента является шестнадцатеричное целое число;
4. c – значением аргумента является символ;
5. s – значением аргумента является строка символов;
6. e – значением аргумента является вещественное число в экспоненциальной форме;
7. f – значением аргумента является вещественное десятичное число с плавающей точкой;
8. u – значением аргумента является беззнаковое целое число;
9. p – значением аргумента является указатель (адрес).

Если после знака % записан не символ, то он выводится на экран. Функция *printf* использует управляющую строку, чтобы определить, сколько всего аргументов и каковы их типы.

Например, в результате работы программы получены переменная i, имеющая значение 100, и переменная j, имеющая значение 25. Обе переменные целого типа. Для вывода этих переменных на экран в виде

**i=100 j=25**

необходимо применить функцию

printf(“i=%d j=%d”,i,j);

Как было описано выше, в кавычках задается формат вывода. Перед знаком % записываются символы, которые будут непосредственно выданы на экран. После знака % применена спецификация d, т.к. переменные i и j имеют целый тип. Сами i и j приведены через запятую в списке аргументов. Если результат должен быть представлен в виде

**i=100; j=25**

необходимо применить функцию

printf(“i=%d; j=%d, i, j);

Если после знака % стоит цифра, то она задает поле, в котором будет выполнен вывод числа. Приведем несколько функций *printf*, которые будут обеспечивать вывод одной и той же переменной S целого типа, имеющей значение 336.

Функция printf(“%2d”, S);

выдает на экран:

**336**

В этом примере ширина поля ( она равна двум) меньше, чем число цифр в числе 336, поэтому поле автоматически расширяется до необходимого размера.

Функция printf(“%6d”, S);

выдаст на экран:

**\_ \_ \_336**

(6 позиций)

То есть, в результате работы функции число сдвинуто к правому краю поля, а лишние позиции перед числом заполнены пробелами.

Функция printf(“%-6d”, S);

выдаст на экран:

**336\_ \_ \_**

(6 позиций)

Знак «минус» перед спецификацией приводит к сдвигу числа к левому краю поля.

Рассмотрим вывод вещественных чисел.

Если перед спецификацией f ничего не указано, то выводится число с шестью знаками после запятой. при печати числа с плавающей точкой перед спецификацией f тоже могут находиться цифры.

Рассмотрим на конкретном примере три возможные ситуации:

%6f – печать числа с плавающей точкой в поле из шести позиций;

%.2f – печать числа с плавающей точкой с двумя цифрами после десятичной точки;

%6.2f – печать числа с плавающей точкой в поле из шести позиций и двумя цифрами после десятичной точки.

Например, в результате работы программы получены переменные вещественного типа а=3,687 и b=10,17.

Если для вывода значений использована функция

printf(“%7f %8f”,a,b);

то результат будет представлен в виде строки:

**\_ \_ 3.687 \_ \_ \_ \_10.17**

(7 поз.) (8 позиций)

Как видно из примера, лишние позиции заполняются пробелами. Если для вывода значений использована функция

printf(“%.2f %/2f”, a, b);

то результатом будет строка:

**3.69 10.17** ,

из которой следует, что в первом числе третья цифра после десятичной точки отброшена с округлением, т.к. указан формат числа с двумя цифрами после десятичной точки.

Если для вывода значений использована функция

printf(“%7.2f e”,a,b);

то будет выведена строка:

**\_ \_ \_ 3.68 1.010000е+01**

(7 позиций)

Поскольку для вывода значения переменной b применена спецификация е, то результат выдан в экспоненциальной форме. Следует отметить, что , если ширина поля меньше, чем число цифр в числе, то поле автоматически расширяется до необходимого размера.

Как было отмечено выше, в управляющей строке могут содержаться управляющие символьные константы. Среди управляющих символьных констант наиболее часто используются следующие:

1. \a – для кратковременной подачи звукового сигнала;
2. \b – для перевода курсора влево на одну позицию;
3. \n – для перехода на новую строку;
4. \r – для перевода курсора в начало текущей строки;
5. \t – для горизонтальной табуляции;
6. \v – для вертикальной табуляции.

Предположим, в результате работы программы переменная i получила значение 50. В результате записи инструкции вызова функции

printf(“\t ЭВМ\n%d\n”,i);

сначала выполнится горизонтальная табуляция (\t), т.е. курсор сместится от края экрана на 8 позиций, затем на экран будет выведено слово “ЭВМ”, после этого курсор переместится в начало следующей строки (\n), затем будет выведено целое значение i по формату d, и окончательно курсор перейдет в начало новой строки (\n). Таким образом, результат работы этой функции на экране будет иметь вид:

**\_ \_ \_ \_ \_ \_ \_ \_ ЭВМ**

**50**

1.3.2. Ввод данных

Для форматного ввода данных используется функция

scanf(«управляющая строка», аргумент1, аргумент2,...);

Если в качестве аргумента используется переменная, то перед ее именем записывается символ &.

Управляющая строка содержит спецификации преобразования и используется для установления количества и типов аргументов. спецификации для определения типов аргументов такие же, как и для функции printf. Перед символами d, o, x, f может стоять буква l. В первых трех случаях соответствующие переменные должны иметь тип long, а в последнем double.

Рассмотрим пример. Требуется ввести значения для переменных i (целого типа) и a (вещественного типа). Эту задачу выполнит функция:

scanf(“%d%f”,&i,&a);

В управляющей строке спецификации трех типов могут быть отделены друг от друга различными знаками, в том числе и пробелом. Следовательно, при занесении значений переменных необходимо использовать указанный разделитель. Если спецификации не отделены одна от другой никакими значениями, то значения переменных заносятся через пробел.

В языке C++ есть две очень удобные функции puts и gets, позволяющие вводить и выводить строку символов. Пример их использования показан ниже:

**#include<stdio.h>**

**main()**

**{**

**char q[40]; /\*объявление строки символов\*/**

**puts(“Введите строку символов”);**

**gets(q); /\*ввод строки символов\*/**

**puts(q); /\*вывод строки символов\*/**

**}**

В результате работы программы вначале на экране появится текст:

**Введите строку символов**,

после чего следует ввести какую-либо строку символов. Эта информация при помощи оператора gets будет присвоена элементам символьного массива q. Оператор puts выведет строку символов.

**1.3.3. Операторы и выражения**

Выражения широко используются в программах на языке C++ и представляют собой формулы для вычисления переменных. Они состоят из операндов (переменные, константы и др.), соединенных знаками операций (сложение, вычитание, умножение и др.). Порядок выполнения при вычислении значения выражения определяется их приоритетами и может регулироваться с помощью круглых скобок. Наиболее часто арифметические выражения используются в операторе присваивания. Этот оператор заменяет значение переменной в левой части оператора на значение выражения, стоящего в правой части, и имеет следующую форму:

переменная = выражение;

В языке C++ может быть использован модификатор const, запрещающий какие бы то ни было переопределения константы: ее уменьшение, увеличение и т.п. Модификатор const, используемый отдельно, эквивалентен const int. Приведем примеры:

const float a=3.5;

const j=47;

В таблице 1 приведены арифметические операции, используемые в языке C++.

Таблица 1

|  |  |
| --- | --- |
| Знакоперации | Выполнение действия |
| + | Сложение |
| – | Вычитание |
| \* | Умножение |
| / | Деление |
| % | Деление по модулю |

Результатом деления по модулю является остаток от деления. Например, если b=5, c=2, то при выполнении операции

а=b%c,

переменная а получит значение 1.

Широкое распространение находят также выражения с еще одной нетрадиционной терпарной операцией ?: . В выражении

у=х?а:b,

у=а, если х не равно нулю, и у=b, если х равно нулю. Следующее выражение

у=(а>b)?a:b;

позволяет присвоить переменной y значение большей переменной (а или b), т.е. y=max(a,b).

В таблице 2 приведены некоторые функции, применяемые при программировании на C++.

 Таблица 2

|  |  |
| --- | --- |
| Математическая запись | Запись на языке C++ |
| | X | | int abs(int X) |
| | X | | float fabs(float X) |
| arccos X | double acos(double X) |
| arcsin X  | double asin(double X) |
| arctg X | double atan(double X) |
| cos X | double cos(double X) |
| sin X | double sin(double X) |
| tg X  | double tan(double X) |
| eX | double exp(double X) |
| ln X | double log(double X) |
| log X | double log10(double X) |
| √X | double sqrt(double X) |
| XY | double pow(double X, double Y) |

Перед аргументом и функцией указан допустимый тип (при программировании эта запись типа опускается).

В программах на языке C++ важная роль отводится комментариям, которые повышают наглядность и удобство чтения программ. Они могут быть записаны в любом месте программы и обрамляются символами /\* и \*/.

Рассмотрим пример программы на языке C++.

Требуется вычислить:

, где

Для работы с математическими функциями необходимо перед функцией main поместить строку:

**#include <math.h>**

Программа на C++ имеет вид:

**#include <stdio.h>**

**#include <math.h>**

**main()**

**{**

**float z,f,k; /\*объявление вещественных переменных z,f,k\*/**

**double y,a,b,c,d,x; /\*объявление переменных y,a,b,c,d,x
 переменными двойной точности\*/**

**scanf(“%f %f %f %lf %lf”, &z, &f, &k, &d, &x); /\*ввод с клавиатуры
 переменных z,f,k,d,x \*/**

**a=log(x)+(z+f)/k;**

**b=sin(x)+tan(x);**

**c=pow(d+exp(x),1./5);**

**y=(a+b)/c;**

**printf(“%lf %lf %ef %lf”, a, b, c, y); /\*вывод на экран значений
 переменных a,b,c,y\*/**

**}**

Следует обратить внимание на то, что при вычислении переменной с, выражение, стоящее в правой части, представлено как 5√(d+ex), поэтому применена функция pow. Еще одно замечание. Следует осторожно подходить к делению целых чисел. Если оба операнда целые, то результат тоже будет целым, а дробная часть отбрасывается. таким образом, при выполнении операции 1/5, результат будет равен нулю. Для того чтобы сохранить дробную часть, хотя бы один из операндов должен быть вещественным. Это условие выполнено при вычислении 1./5 .

2. Задания

Из таблицы 3 взять задание по варианту и написать программу для вычисления выражения на языке C++.

Таблица 3

|  |  |  |
| --- | --- | --- |
| № варианта | Задание | Исходныеданные |
| d =c = sin2 (x + y) – ln x;b = | i – j 3|;, гдеd + bcr = |  | x = 4,5y = 8,5i = 3j = 6 |
| d = | i2 – m|z =ac – da = sin3 ( 2 + n ), где |  | n = 10,2x = 4,3i = 5m = 2 |
| s =xa + y+ ln2 zq =c – bdb = | x2 + y |, где |  | m = 8,1w = 4,2x = 10y = 4 |
| a = | k + m2 |, где |  | t = 4,7z = 0,8p = 6,2k = – 4m = 6 |
| d = | j3 – i2 |, гдеm =x2 + yd |  | t = 4,1p = 3,2w = 8,7j = 3i = 7 |
| f = | k2 + m3 |t = ln x2 + sin2 y, гдеp =t – qf2 |  | x = 5,7y = 1,9k = 8m = 2 |
| x = sin e2m, гдеz = ln2 p –x – y2t |  | m = 5,6n = 9,4i = 4j = 9 |
| y = tg2 q – 4p, гдеs =x2 + y2t |  | q = 1,7p = 2,3i = 9j = 4 |
| 1.

y = sin2( n + m ), гдеr = x∙ay+ t∙ln2 m |  | n = 2,6m = 3,7p = 6k = 2 |
| y = sin(p2+t), гдеs =x + ym + 5 |  | t = 1,1p = 2,4k = 4i = 5 |
| t = log2 (x + y), гдеz =tp+ f |  | x = 3,7y = 2,1i = 3j = 4 |
| s =q + pp+ tg2 zq =hc + dc = tg2 (t+p) + sin p2, где |  | t = 1,5p = 4,8i = 2m = 6 |
| , где |  | z = 2,4m = 5,8n = 4j = 5 |
| r =hp – dh = sin (tg x2), где |  | x = 9,5y = 3,6j = 2i = 5 |
| a = cos2 (x+y), гдеca + b2d = |  | x = 6,4y = 1,7k = 5m = 7 |

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьтер, 1993. - 160 с.
5. Уинер Р. Язык Турбо C . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке C. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №2
“Условные и безусловные операторы в языке программирования С++”

**Цель работы:** познакомиться с работой условного оператора и оператора перехода. Изучить оператор выбора варианта. Научиться применять их при составлении программ.

1. Теоретические сведения

1.1 Условный оператор *if*

Рассматриваемая группа операторов позволяет организовать ветвление в программе. Часто, например, необходимо в зависимости от того или иного результата реализовать одну либо другую группу операторов (инструкций). В языке С для этих целей используются операторы if (если) – else (иначе), switch (переключатель) и goto (идти к).

Оператор *if* имеет вид:

if (проверка условия) инструкция1; else инструкция2;

Если условие в скобках принимает истинное значение (см. Рис. 1), выполняется Инструкция1, а если ложное – Инструкция2.

Рис.1.

Вход

Выбор

Инструкция1

Инструкция2

Условие истинно

Условие ложно

Выход

Например:

**if (a>b)**

**z=a;**

**else**

**z=b;**

Необходимо обратить внимание на точку с запятой после z=a. Здесь она обязательна, поскольку за *if* должна следовать инструкция, которая всегда заканчивается точкой с запятой.

В операторе *if* слово *else* может отсутствовать. В этом случае, если условие в скобках принимает истинное значение, выполняется Инструкция1, а если ложное, то инструкция1 пропускается и управление передается следующему оператору по тексту программы.

Например:

if (num>10) num=2\*num;

printf(“%d\n”,num);

Оператор вывода будет выполняться всегда, а оператор присваивания только в том случае, если условие будет истинным.

Операции отношения, используемые для сравнения, приведены в таблице 1.

 Таблица 1

|  |  |
| --- | --- |
| Операция | Смысл |
| < | меньше |
| <= | меньше или равно |
| = = | равно |
| >= | больше или равно |
| > | больше |
| != | не равно |

Не следует путать операцию отношения «==» с операцией присваивания «=».

Рассмотрим простейшую программу:

**#include <stdio.h>**

**main( )**

**{**

**int a,b;**

**puts(“Введите значения a и b”);**

**scanf(“%d %d”, &a,&b);**

**if (a>b) puts(“a больше b”);**

**else puts(“”); /\*демонстрация оператора if – else\*/**

**if (a==b) puts(“”); /\*демонстрация оператора if без слова
 else\*/**

**}**

Если для выполнения программы ввести числа 5 и 3, то на экране появится строка:

**а больше b**

При введении чисел 5 и 5, на экране появится две строки:

**a меньше или равно b**

**a равно b**

Иногда в условном операторе после ключевых слов *if* или *else* следует инструкция *if*, ее называют вложенной. Слово *else* всегда относится к ближнему предшествующему ему *if*. Также после ключевых слов *if* и *else* вместо одной инструкции может быть записано несколько инструкций, которые должны заключаться в фигурные скобки.

Например, в

**if (n>0)**

 **if (a>b)**

 **z=a;**

 **else**

 **z=b;**

*else* относится к внутреннему *if*, что и показано с помощью отступов. Если требуется иная интерпретация, необходимо должным образом расставить фигурные скобки:

**if (n>0)**

**{**

**if (a>b)**

 **z=a;**

**}**

**else**

**z=b;**

В данном примере *else* относится к внешнему *if*.

В сложных конструкциях, содержащих много *if* и *else*, уместно вложенные *if* обрамлять фигурными скобками.

Вместо вложенных конструкций if–else часто целесообразно использовать сложные выражения для проверки условия. Сложные выражения получаются путем объединения нескольких простых отношений с помощью логических операций. В языке С++ имеются три основные логические операции, приведенные в таблице 2.

 Таблица 2

|  |  |
| --- | --- |
| Операция | Смысл |
| && | и |
| || | или |
| ! | не |

Операция ! имеет очень высокий приоритет, он выше, чем умножения, и только круглые скобки имеют более высокий приоритет. Приоритет операции && выше, чем операции ||, а обе они имеют более низкий приоритет, чем операции отношения. Поэтому выражение

a>b&&b>c||b>d

будет интерпретировано так:

((a>b)&&(b>c))||(b>d),

т.е. **b** находится в промежутке между **c** и **a** или **b** больше **d**.

**Пример**



**main ( )**

**{ float x, y;**

**scanf ("%f", & x);**

**if (x > 0)**

**y = x\*x + 1;**

**else**

**y = x + 4;**

**printf ("%f %f", x, y);**

**}**

1.2. Оператор безусловного перехода

Его можно представить в следующей форме:

**goto метка;**

Метка – это любой идентификатор.

Например:

goto a2;

Оператор *goto* указывает, что выполнение программы необходимо продолжить, начиная с инструкции, перед которой записана метка. В программе обязательно должна быть строка, где указана метка, поставлено двоеточие и записана инструкция, к которой должен выполняться переход.

Например:

a2: k=5;

Метки в программе описывать не нужно. Применение оператора безусловного перехода в языке С является нежелательным, так как он нарушает структурную наглядность программы.

1.3. Оператор выбора s*witch*

Оператор ***switch*** позволяет выбрать одну из нескольких альтернатив (Рис.2). Он записывается в следующем виде:

**switch (выражение)**

**{case константа1, вариант 1; break;**

 **. . .**

 **case константа n, вариант n; break;**

**…**

**default: вариант n+1; break;}**

вариант 1

break

Вход

Выражение = константа 1

case

Выражение = константа n

нет

. . .

нет

default:

да

вариант n

вариант n

да

да

break

break

выход

рис. 2

На рис.2 представлена соответствующая иллюстрация.

В операторе **switch** вычисляется целое выражение в скобках (его называют селектором), и его значение сравнивается со всеми константами. При совпадении выполняется соответствующий вариант (одна или несколько инструкций). Все константы в записи оператора должны быть различными. Вариант с ключевым словом **default** (прочие) реализуется, если ни один другой не подошел (если слово **default** отсутствует, а все результаты сравнения отрицательны, то ни один вариант не выполняется. Для прекращения последующих проверок после успешного выбора некоторого варианта используется оператор **break**, обеспечивающий немедленный выход из оператора **switch**.

Например:

**#include<stdio.h>**

**main( )**

**{**

**char y;**

**scanf(“%c”,&y);**

**switch(y)**

**{**

**case ’1’:**

**printf(“Ветвь 1\n”);**

**break;**

**case ’2’:**

**case ’3’:**

**printf(“Ветвь 2 или 3\n”);**

**break;**

**default:**

**printf(“Ветви 1,2,3 не работают\n”);**

**}**

**}**

Оператор **scanf** вводит переменную **y**. Ее значение в операторе **switch** сравнивается со всеми константами операторов **case**. Если ввести символ ‘1’, то на экране появится строка:

Ветвь 1

по оператору **break** произойдет выход из переключателя **switch**, и программа завершит свою работу. Если ввести символы ‘2’ или ‘3’, то на экран будет выведена строка:

Ветвь 2 или 3

При вводе любого другого символа управление перейдет к ключевому слову **default** и на экране появится строка:

Ветви 1,2,3 не работают.

**2. Задания**

2.1. Из таблицы 3 взять задание по варианту и написать программу, используя оператор условного перехода.

2.2. Из таблицы 4 взять задание по варианту и написать программу, используя оператор выбора. Для выбора четвертой ветви использовать вариант с ключевым словом default.

##### Таблица 3

| **№ варианта** | Содержание | **Исходныеданные** |
| --- | --- | --- |
|  |   | a=2b=1,5c=1d=3e=0,5 |
|  |   | c=-2 a=1,5d=2x=3 |
|  |   | a=5b=1,2y= 0,6 |
|  |   | d=2a=3,5x=3i=2,3 |
|  |   | ω=1x=2,5a=4b=0,4 |
|  |   | a=0,5b=2h=4R=1,4 |
|  |   | p=0,6l=2h=5R=4 |
|  |   | x=0,63c=1,5d=2a=0,37 |
|  |   | y=2,6x=1,6a=0,4 |
|  |   | a=1,5b=2x=0,5 |
|  |   | x=0,5a=4,3y=2,6d=0,3 |
|  |   | a=3y=2,7i=2x=1 |
|  |   | x=0,73y=0,4t=2,6 |
|  |   | a=2x=0,54c=2,3b=1,8R=3 |
|  |   | x=0,4a=0,88c=3,6 |
|  |   | a= -1b=0,8x=1c= -0,7 |
|  |   | x=2y=3,5a=0,1b=4 |
|  |   | x=0,8y=0,2a=4b=5t=1 |
|  |   | y=2a=0,5x=1,8b=0,6 |
|  |   | x=1,8y=2t=4a=1,5R=4,6b=0,3 |

Таблица 4

| **№ варианта** | **Содержание** | **Исходные данные** |
| --- | --- | --- |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, -560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo Cu ++. Основы объектно- ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьтер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №3
“Операторы цикла в языке программирования С++”

**Цель работы:** ознакомиться с циклическими алгоритмами и операторами, реализующими эти алгоритмы. Освоить особенности применения каждого оператора. Составить программы с использованием всех операторов цикла.

1. Теоретические сведения

1.1. Оператор цикла while

Описание:

while (выражение) оператор;

Действие:

Выполняется оператор до тех пор, пока значение выражения в скобках истинно. Проверка значения выражения происходит перед каждым выполнением оператора. Когда значение выражения ложно, цикл while заканчивается. Если выражение ложно с самого начала, оператор не выполняется ни разу.

Комментарий:

Следует заметить, что после ключевого слова while и выражения, заключенного в круглые скобки, точка с запятой не ставится.

Оператор иногда называется телом цикла. В теле цикла должны выполняться действия, в результате которых меняется значение управляющего выражения. В противном случае можем получить бесконечный цикл.

Пример:

**/\*Демонстрация цикла while\*/**

**#include <stdio.h>**

**main( )**

**{**

**int i=1;**

**while (getchar()!=’R’) i++;**

**/\*оператор getchar() вводит любой символ с клавиатуры\*/**

**printf(“Символ R %d-й”,i);**

**}**

Приведенная программа позволяет определить порядковый номер первой введенной буквы R в последовательности символов. Она показывает использование цикла while, в теле которого всего одна инструкция (i++ - увеличение значения целого числа i на единицу). Если запустить эту программу на выполнение и ввести последовательность символов, например:

abFk!Rgm ,

то на экране появится строка:

Символ R 6-й.

1.2. Оператор цикла do-while

Описание:

do оператор while (выражение);

Действие:

В операторе ***do - while*** тело цикла выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

**Комментарий:** после слова ***while*** и выражения, заключенного в скобки, ставится точка с запятой. Если в теле цикла содержится более одной инструкции, то операторы цикла заключаются в фигурные скобки.

Пример:

**/\*Демонстрация цикла do-while \*/**

**#include <stdio.h>**

**main()**

**{**

**int i=0; /\*i=0, а не единице\*/**

**do i++;**

**while (getchar()!=’R’);**

**printf(“Символ R %d-й”,i);**

**}**

Программа, представленная выше, теперь написана с циклом do-while. Результат программы будет таким же.

1.3. Оператор цикла for

Описание:

for (выражение 1; выражение 2; выражение 3) оператор;

Действие:

В круглых скобках содержится три выражения. Первое из них служит для инициализации счетчика. Она осуществляется только один раз – когда цикл ***for*** начинает выполняться. Второе выражение необходимо для проверки условия, которая осуществляется перед каждым возможным выполнением тела цикла. Когда выражение становится ложным, цикл завершается. Третье выражение вычисляется в конце каждого выполнения тела цикла, происходит приращение числа на шаг.

Комментарий: в операторе цикла ***for*** точка с запятой после закрывающейся круглой скобки не ставится. Любое из трех или все три выражения в операторе могут отсутствовать, однако разделяющие их точки с запятыми опускать нельзя. Если отсутствует выражение 2, имеем бесконечный цикл. Например:

for (scanf(“%d”,&p);;p++) оператор;

В языке С++ предусмотрены две нетрадиционные операции: (++) – для увеличения на единицу и (--) – для уменьшения на единицу значения операнда. Операции ++ и -- можно записывать как перед операндом, так и после него. В первом случае (++n или --n) значение операнда (n) изменяется перед его использованием в соответствующем выражении, а во втором (n++ или n--) – после его использования.

Если отсутствуют выражения 1 и 3, цикл становится эквивалентным ***while***. Например:

for (;a<20;) оператор;

Каждое из выражений может состоять из нескольких выражений, объединенных операцией «запятая». Например:

for(i=0, j=1; i<100; i++, j++) a[i]=b[j];

Тело цикла заключается в фигурные скобки, если в нем более одного оператора.

Пример:

**/\*демонстрация цикла for\*/**

**#include <stdio.h>**

**main()**

**{int i,j=1,k;**

**for (i=1;i<=3;i++)**

 **printf(“Минск\t”);**

**/\*В цикле for три раза выполняется функция вывода\*/**

**/\*Здесь i-управляющая переменная цикла\*/**

**printf(“\nУкажите число повторений цикла\n”);**

**scanf(“%d”,&k);**

**for (i=1;i<=k;i++)**

**{j\*=i;**

 **printf(“%d”,j);}**

**/\*Здесь две инструкции (более одной), поэтому они заключаются
 в фигурные скобки\*/**

**j=i;**

**printf(“\n”);**

**/\*Переменной j присваивается значение 1 и осуществляется
 перевод курсора\*/**

**/\*В следующем цикле for выполняются те же действия, и в
предыдущем\*/**

**for (i=1;i<=k;i++) printf(“%d “, j\*=i);**

**}**

Результаты выполнения программы следующие:

Минск Минск Минск

Укажите число повторений цикла

5

1 2 6 24 120

1 2 6 24 120

1.4. Оператор *break*

Описание:

***break*** используется для прекращения выполнения цикла из-за обнаружения ошибки, для организации дополнения к условию в заголовке цикла, для прекращения бесконечного цикла.

Пример:

**while (st>0 && st<25)**

**{**

**if st==4||st==8||st==12)**

**break;**

**}**

Работа цикла полностью прекращается, как только условие в операторе ***if*** становится истинным.

1.5.Оператор *continue*

Описание:

continue;

Действие:

Этот оператор может использоваться во всех трех типах циклов. Как и в случае оператора **break**, он приводит к изменению характера выполнения программы. Однако вместо завершения работы цикла наличие оператора **continue** вызывает пропуск “оставшейся” части итерации и переход к началу следующей.

Пример. Заменим в предыдущей программе оператор **break** на **continue**.

**while (st>0 && st<25)**

**{**

**if (st==4||st==8||st==12)**

**continue;**

**}**

При истинном условии в операторе ***if*** оператор ***continue*** вызывает пропуск идущих за ним операторов тела цикла и осуществляется переход к началу следующей итерации.

2. Задания

1. Задание взять из таблицы 1 и таблицы 2 согласно варианту.
2. Разработать блок-схемы алгоритма.
3. Написать и отладить программы.

Таблица 1

| **№ варианта** | Задание |
| --- | --- |
| 1. |  |
| 2. |  |
| 3. |  |
| 4. |  |
| 5. |  |
| 6. |  |
| 7. |  |
| 8. |  |
| 9. |  |
| 10. |  |
| 11. |  |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |
| 20. |  |
| 21. |  |
| 22. |  |
| 23 |  |
| 24. |  |
| 25. |  |

### Таблица 2

| **№ варианта** | Задание |
| --- | --- |
| 1. |  |
| 2. |  |
| 3. |  |
| 4. |  |
| 5. |  |
| 6. |  |
| 7. |  |
| 8. |  |
| 9. |  |
| 10. |  |
| 11. |  |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |
| 20. |  |
| 21. |  |
| 22. |  |
| 23 |  |
| 24. |  |
| 25. |  |

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьютер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к C. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №4
“Массивы и указатели в языке программирования С++”

1. Теоретические сведения

**Цель работы:** ознакомиться с основными принципами работы с одномерными и двумерными массивами. Освоить работу с указателями и операциями над указателями.

* 1. Массивы

Массив состоит из нескольких элементов одного и того же типа. Можно выбирать любой элемент массива. Для этого необходимо задать индекс, который указывает относительную позицию элемента. число элементов массива задается при его объявлении и в дальнейшем не меняется. Если массив объявлен, к любому его элементу можно обратиться следующим образом: указать имя массива и индекс его элемента в квадратных скобках.

Массивы объявляются так же, как и переменные. Например:

int a[100];

float c[10][20];

В первой строке объявляем массив а из 100 элементов целого типа: а[0],a[1], … ,a[99] (индексация всегда начинается с нуля). Во второй строке объявлен двумерный массив вещественного типа. Двумерный массив представляется как одномерный, элементы которого являются тоже массивами. В первых квадратных скобках указывается количество строк в массиве, во вторых – количество столбцов.

**Пример 1.** Задан одномерный массив S, состоящий из десяти элементов вещественного типа. Вывести на экран дисплея значения элементов этого массива в обратном порядке.

**#include <stdio.h>**

**main()**

**{float s[10];**

**int i;**

**for (i=0;i<10;i++)**

 **scanf(“%f”,&s[i]); /\*ввод элементов массива\*/**

**for (i=9;i>=0;i--)**

 **printf(“%f”,s[i]); /\* вывод элементов в обратном порядке\*/**

**}**

**Пример 2.** Задана двумерная матрица а, имеющая пять строк и пять столбцов. Определить номер строки с наибольшим числом единиц в этой строке.

**#include <stdio.h>**

**main()**

**{int i,j,p,q=0,f=0,a[5][5];**

**for (i=0;i<5;i++)**

**for (j=0;j<5;j++)**

**scanf(“%d”, &a[i][j]); /\*ввод матрицы\*/**

**/\*поиск в матрице а строки с наибольшим числом единиц\*/**

**for (i=0;i<5;i++)**

**{p=0;**

**for (j=0;j<5;j++)**

**if (f[i][j]==1) p++;**

**if (q<p) {q=p; f=i;}**

**}**

**/\*f – номер строки с наибольшим числом единиц, q – число единиц в f-й строке\*/**

**printf(“%d %d”,f,q);**

**}**

**Пример 3.** Переставить местами элементы главной и побочной диагоналей массива D(6,6). Полученную матрицу вывести на экран дисплея.

**#include <stdio.h>**

**main()**

**{int i,j,a,d[6][6];**

**for (i=0;i<6;i++)**

**for (j=0;j<6;j++)**

**scanf(“%d”, &d[i][j]); /\*ввод матрицы\*/**

**/\*перестановка местами элементов главной и побочной
 диагоналей\*/**

**for (i=0; i<5; i++)**

**{a=d[i][i];**

**d[i][i]=d[i][6-i];**

**d[i][6-i]=a;**

**}**

**for (i=0; i<6; i++)**

**for (j=0; j<6; j++)**

**printf(“%d%c”, d[i][j], (j==5)?”\n”:” ”);**

**/\*вывод по строкам элементов матрицы\*/**

**}**

При выводе элементов матрицы по строкам применена тернарная операция ?:. Смысл этой операции сводится к следующему: если j=5 (закончен вывод элементов по строке), то курсор переводится в начало следующей строки (работает символьная константа “\n”), в противном случае выводится один пробел. Так как в операторе вывода используется символьная константа, то применена спецификация %c.

В заключении этого раздела отметим, что массив можно инициализировать, т.е. присвоить его элементам начальные значения. Это делается при объявлении типа массива, например:

int a[5]= { 0, 0, 0, 0, 0};

Это значит, что все элементы массива получают нулевое значение.

Двумерный массив можно инициализировать следующим образом:

int a[3][3] = {{10,20,30},

{40,50,60},

{70,80,90}};

При инициализации число элементов можно не указывать, т.к. в этом случае оно может быть вычислено по количеству присваиваемых значений (в фигурных скобках), например:

int a[] = {10,20,30,40,50};

* 1. Указатели
		1. **Указатели и одномерные массивы**

Указатель – это переменная, которая содержит адрес переменной. Так как указатель – это адрес некоторого объекта, то через него можно обращаться к данному объекту.

В С++ существует тесная связь между указателями и массивами. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен при помощи указателя.

Декларация

int a[10];

определяет массив а размера 10, т.е. блок из десяти последовательных объектов, представленных на рисунке, с именами a[0], a[1], … ,a[9].

Запись a[i] отсылает нас к *i*-му элементу массива. Если *ра* есть указатель, т.е. определен как

Ра

Pa+1

Pa+2

a:

a[0]

a[1]

a[9]

int \*pa; , то в результате присваивания

pa = &a[0];

*pa* будет указывать на нулевой элемент массива а; иначе говоря, *ра* будет содержать адрес элемента a[0] (см. рис.). Теперь присваивание

x=\*pa;

будет копировать содержимое а[0] в х.

Если *ра* указывает на некоторый элемент массива, то *ра+1* по определению указывает на следующий элемент (см. рис.).Таким образом, если *ра* указывает на a[0], то \*(ра+1) есть содержимое a[1], ра+1 – адрес a[1], \*(ра+i) – содержимое a[i].

Поскольку имя массива есть не что иное как адрес его начального элемента, присваивание

pa=&a[0];

можно также записать в следующем виде:

pa = a;

Так как *ра* – указатель, то в выражениях его можно использовать с индексом, то есть запись pa[i] эквивалентна записи \*(pa+i). Элемент массива одинаково разрешается изображать и в виде указателя со смещением, и в виде име-

ни массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель – это переменная, поэтому можно написать *pa=a* или *pa++*. Но имя массива не является переменной, и запись типа *a=pa* не допускается.

Следует также различать выражения \*(a+2) и \*a+2:

\*(а+2) – значение третьего элемента массива а;

\*а+2 – добавление числа 2 к значению первого элемента массива.

**Пример 4**. Вывести значения одномерного массива обычным способом и с использованием указателей.

**#include <stdio.h>**

**int a[6]={10,20,30,40,50,60}; /\*объявление и инициализация массива\*/**

**main ()**

**{int i, \*p;**

**for (i=0; i<6; i++)**

**printf(“%d”,a[i]); /\*вывод массива обычным способом\*/**

**for (p=&a[0];p<=&a[5];p++)**

**printf(“%d”,\*p); /\*вывод массива с использованием указателя\*/**

**for (p=&a[0],i=0; i<6; i++)**

**printf(“%d”,p[i]); /\*еще один вариант с использованием указателя\*/**

**}**

Дадим еще некоторые пояснения. Операция р++ увеличивает значение указателя на единицу. Если p=&a[i], то после операции р++ в р содержится адрес элемента a[i+1].

**Пример 5.** Найти среднее арифметическое массива, состоящего из шести элементов, с использованием указателя.

**#include <stdio.h>**

**int a[]={10,20,30,40,50,60}**

**main()**

**{int i,\*p;**

**float s;**

**p=a; /\*указатель получает значение адреса нулевого элемента массива\*/**

**for (s=0,i=0; i<6; i++)**

**s+=\*(p+i); /\*получение суммы элементов массива\*/**

**s=s/6; /\*среднее арифметическое массива\*/**

**printf(“%f”,s);**

**}**

**Пример 6.** Решить задачу, приведенную в примере 1, с использованием указателя.

**#include <stdio.h>**

**main()**

**{float s[10];**

**int \*p,i;**

**for (i=0;i<10;i++)**

**scanf(“%f”,s[i]);**

**p=&s[9]; /\*указатель получает значение адреса последнего элемента массива\*/**

**for (i=0; i<10; i++)**

**printf(“%f”,\*(p-i)); /\*вывод элементов в обратном порядке\*/**

**for (p=&a[9]; p>=&a[0]; p--) /\*еще один способ вывода элементов в обратном порядке\*/**

**printf(“/n%d”,\*p);**

**}**

1.2.2. Указатели и двумерные массивы

Предположим, что у нас есть описания:

int z[4][2]; /\*\*/

int pz; /\*\*/

Тогда pz=z указывает на нулевой столбец нулевой строки, т.е.

pz=&z[0][0];

**Пример 7.** Вывести на экран значения нулевого, второго и четвертого элементов массива.

**#include <stdio.h>**

**int a[]=(10,20,30,40,50,60); /\*объявление и инициализация массива а\*/**

**main()**

**{**

**for (p=a,i=0; p+i<=a+4; p++, i++)**

**printf(“%d”; (p+i)); /\*вывод на экран значений нулевого, второго и четвертого элементов\*/**

**}**

В цикле происходит одновременное увеличение указателя р и индекса i на единицу, вследствие чего на экран после нулевого элемента выводится второй, и затем четвертый. Поскольку элементы двумерного массива хранятся в памяти ЭВМ по строкам, то

pz+1 = &z[0][1], pz+2 = &z[1][0], pz+3 = &z[1][1]

и т.д.

Двумерный массив описан как массив массивов. Если z является именем массива, то каковы имена четырех строк, каждая из которых является массивом из двух элементов? Имя первой строки z[0], второй - z[1], и т.д. Однако имя массива является также указателем на этот массив в том смысле, что оно ссылается на первый его элемент. Значит:

z[0]=&z[0][0], z[1]=&z[1][0], z[2]=&z[2][0], z[3]=&z[3][0].

**Пример 8.** Задана матрица ***а***. Вывести на экран элементы главной диагонали, первой строки и значений первых элементов каждой строки матрицы, применив для этого указатели.

**#include <stdio.h>**

**int a[3][3]={{10,20,30},**

**{40,50,60},**

**{70,80,90}};**

**/\*объявление и инициализация двумерного массива\*/**

**int \*pa[3]={a[0],a[1],a[2]};**

**/\*объявление и инициализация указателя ра на строки массива а и присвоение начальных значений : pa[0]=a[0]; pa[1]=a[1]; pa[2]=a[2]\*/**

**int p=a[0]; /\*объявление указателя на нулевой элемент нулевой строки массива а\*/**

**main ()**

**{int i;**

**for (i=0;i<9;i+=4)**

**printf(“%d”,\*(p+i)); /\*вывод на экран элементов главной диагонали\*/**

**for (i=0; i<3; i++)**

**printf(“%d”,\*p[i]); /\*вывод на экран элементов первой строки\*/**

**for (i=0; i<3; i++)**

**printf(“%d”,pa[i]); /\*вывод на экран первых элементов каждой строки матрицы\*/**

**}**

Сделаем некоторые пояснения для первого оператора цикла. Представим матрицу в виде одномерного массива, записанного по строкам:

a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2], a[2][0], a[2][1], a[2][2]

Тогда элементы, стоящие на главной диагонали, занимают нулевое, четвертое и восьмое места, т.е. интервал между интересующими нас элементами равен четырем, поэтому переменная i изменяется с шагом 4. Соответственно с таким же шагом меняются адреса ячеек, содержимое которых выводится на экран.

**2. Задания**

Задание взять из таблицы согласно заданному варианту. Написать два варианта программы: без применения указателей и с указателями.

Таблица

|  |  |
| --- | --- |
| **№ варианта** | **Задание** |
|  | Определить, является ли заданная квадратная матрица А(5,5) симметричной относительно главной диагонали. |
|  | Задана матрица В(4,4). Определить, отсортированы ли все элементы первого столбца в возрастающем порядке. |
|  | Задана матрица С(5,5). Поменять местами максимальный элемент каждой строки с первым элементом соответствующей строки. |
|  | Переписать первые элементы каждой строки матрицы D(3,3), которые больше 10, в массив В. |
|  | Задана матрица Q(5,5). Заменить последний нуль в каждой строке на 5. |
|  | Задана матрица D(4,4). Определить максимальный среди положительных, минимальный среди отрицательных и поменять их местами. |
|  | Задана матрица А(4,4). Заменить первый нуль в каждом столбце на количество нулей в этом столбце. |
|  | Задана матрица F(9,3). определить, равны ли все элементы первого столбца соответствующим элементам главной диагонали. Если нет, то поменять их местами. |
|  | Задана матрица C(5,5). Получить вектор В, каждый элемент которого равен количеству нулей, стоящих в столбце матрицы. |
|  | Задана матрица В(4,4). Если в строке есть хотя бы одна единица, то заменить эту строку нулями. |
|  | Задана матрица Q(3,3). Если на главной диагонали стоит нуль, то соответствующую строку заменить единицами. |
|  | Задана матрица D(4,4). Если максимальный элемент матрицы стоит на главной диагонали, то все элементы главной диагонали сделать равными максимальному. |
|  | Задана матрица С(5,5). Если минимальный элемент стоит в первой строке, то все элементы, стоящие в строке за ним, заменить нулями. |
|  | Задана матрица А(4,4). Если максимальный элемент матрицы равен сумме элементов первой строки, то поменять местами первую строку с той строкой, где находится максимальный элемент. |
|  | Задана матрица А(4,4). Если максимальный элемент матрицы равен сумме элементов первой строки, то поменять местами первую строку с той строкой, где находится максимальный элемент. |

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo Cu ++. Основы объектно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьютер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №5
"Работа с функциями в языке программирования С++"

**Цель работы:** ознакомиться с особенностями применения функций в языке С++, с понятием прототипа и областью его применения, с понятием автоматических внешних, статических и регистровых переменных и их применением при составлении программ с использованием функций.

1. Теоретические сведения

1.1. Функции

Программы на языке С++ обычно состоят из большого числа отдельных функций (подпрограмм). Как правило, они имеют небольшие размеры и могут находиться как в одном, так и в нескольких файлах.

Связь между функциями осуществляется через аргументы, возвращаемые значения и внешние переменные.

Вызов функции осуществляется следующим образом:

<тип функции >(параметр 1, параметр 2 , …);

Если функция имеет переменное число параметров, то вместо последнего из них указывается многоточие.

Передача одного значения из вызванной функции в вызвавшую происходит с помощью оператора возврата, который записывается в следующем виде:

return (выражение);

В этом случае значение выражения (в частном случае может быть просто переменная) передается в основную программу и подставляется вместо обращения к функции.

Пусть вызывающая программа обращается к функции следующим образом:

a=fun(b,c);

Здесь b и c – аргументы, значения которых передаются в вызываемую подпрограмму.

Если описание функции начинается так:

fun(i,j)

то переменные i и j получат значения a и b соответственно.

**Пример 1.** Оформить получение абсолютной величины числа в виде функции. Сама функция может быть оформлена в виде отдельного файла. В этом случае выполняется его включение процедурой *#include.*

Программа имеет следующий вид:

**#include <stdio.h>**

**main()**

**{int a=10,b=0,c=-20;**

**int d, e, f;**

**d = abs(a); /\*обращение к функции abs\*/**

**b = abs(b);**

**f = abs(c);**

**printf(“%d %d %d”, d, b, f);**

**}**

**#include ”abc.c” /\*включение файла abc.c с функцией abs\*/**

**abs(x); /\*Функция, вычисляющая абсолютную величину числа \*/**

**int x; /\*Описание переменных, работающих в функции \*/**

**{int y;**

**y=(x<0)?–x:x; /\*Определение абсолютной величины числа\*/**

**return (y); /\*Возвращает значение у вызывающей программе\*/**

**}**

В приведенной программе описание типа функции было опущено. Это возможно только в том случае, если возвращенное значение имеет целый тип. Во всех остальных случаях описание типа функции обязательно. приведем пример, когда результатом работы функции будет число двойной точности.

**Пример 2.** Оформить в виде функции вычисление ***f= x + y/z.***

В первом примере функция хранилась в виде отдельного файла и включалась процедурой **#include**. Функция может быть включена в один файл с вызывающей программой. В этом случае процедура **#include** не требуется, а сама функция должна быть объявлена в основной программе, если она имеет не целый тип. Приведем программу для примера 2, оформленную таким способом.

Программа имеет вид:

**#include <stdio.h>**

**main()**

**{ double f,x=5.5,y=10.1,z=20.5, vv() /\*объявлены переменные и функция vv\*/**

**f=vv(x,y,z); /\*обращение к функции vv\*/**

**printf(“lf”,f); /\*вывод результата \*/**

**}**

**/\*функция \*/**

**double vv(x,y,z)**

**double x,y,z; /\*объявление переменных функции \*/**

**{double f;**

**f=sqrt(x)+y/z; /\*вычисление значения функции \*/**

**return(f); /\*возврат вычисленного значения функции \*/**

**}**

В языке С++ аргументы функции передаются по значению, т.е. вызванная функция получает временную копию каждого аргумента, а не его адрес. Это означает, что функция не может изменять оригинальный аргумент в вызвавшей ее программе. Однако это легко сделать, если передавать в функцию не переменные, а их адреса.

**Пример 3.** В приведенной ниже программе вводятся некоторые значения переменных а и b, потом в функции izm они меняются местами.

**#include <stdio.h>**

**main()**

**{int a,b;**

**scanf (“%d %d”, &a, &b);**

**izm (&a, &b); /\*обращение к функции izm; аргументами являются адреса переменных a и b\*/**

**printf(“%d, %d”,a, b);/\*вывод на экран измененных значений \*/**

**}**

**#include “izm.c” /\*включение файла izm.c с функцией izm \*/**

**/\*функция\*/**

**izm(a, d); /\*аргументы a и b являются указателями \*/**

**int \*a, \*b; /\* \*a и \*b – значения, на которые указывают
 указатели \*/**

**{int c;**

**c=\*a;**

**\*a = \*b;**

**\*b=c; /\*обмен местами \*/**

**}**

Функция ***izm()*** получает копию адресов переменных a и b, меняет местами значения, записанные по этим адресам, и передает управление в основную программу. Адреса &a и &b в основной программе не изменялись, а вот значения, на которые они указывают, поменялись местами.

Если в качестве аргумента функции используется имя массива, то ей передается адрес начала массива, а сами элементы не копируются. Функция может изменять элементы массива, сдвигаясь (индексированием) от его начала.

**Пример 4.** В массиве S поменять местами элементы: первый со вторым, третий с четвертым и т.д. Оформить этот алгоритм в виде функции reverse.

**#include <stdio.h>**

**main()**

**{int i,j,s[6]; /\* описание переменных i,j и массива s целого типа \*/**

**for (i=0; i<6; i++)**

**scanf(“%d”,&s[i]); /\*ввод элементов массива s\*/**

**reverse(s); /\*обращение к функции reverse\*/**

**for (i=0; i<6; i++)**

**printf(“%d”,s[i]); /\*вывод полученного массива \*/**

**}**

**include “reverse.c” /\*включение файла reverse.c
 с функцией reverse \*/**

**/\*функция\*/**

**reverse(s)**

**int s[]; /\*описание работающего в подпрограмме массива \*/**

**{**

**int a,i;**

**for (i=1; i<5; i+=2)**

**{a=s[i]; s[i]=s[i+1]; s[i+1]=a;} /\*обмен элементов местами\*/**

**}**

Рассмотрим особенности работы функции с двумерным массивом. В предыдущем примере в функции массив был описан как int s[]; для двумерного массива а нельзя записать a[][]. В описании двумерного массива во второй квадратной скобке должно быть указано количество столбцов, например: a[][3].

**Пример 5.** Увеличить все элементы массива а(5,5) в два раза. Оформить этот алгоритм в виде подпрограммы.

**#include <stdio.h>**

**main()**

**{int a[5][5]; /\*описание массива a\*/**

**int i,j; /\*объявление переменных i,j\*/**

**for (i=0;i<5;i++)**

**for (j=0; j<5; j++)**

**scanf(“%d”,a[i][j]); /\*ввод массива\*/**

**mas(a); /\*обращение к функции mas\*/**

**for (i=0; i<5; i++)**

**for (j=0; j<5; j++)**

**printf(“%d”, a[i][j]); /\*вывод полученного результата\*/**

**}**

**/\*функция\*/**

**mas(a)**

**int a[][5]; /\*описание массива а\*/**

**{int i,j; /\*описание переменных i,j\*/**

**for (i=0; i<5; i++)**

**for (j=0; j<5; j++)**

**a[i][j] = 2\*a[i][j]; /\*увеличение элементов массива в 2 раза\*/**

**}**

* 1. Классы памяти

В языке С++ различают четыре основных памяти: внешнюю (глобальную), автоматическую (локальную), статическую и регистровую.

Внешние переменные определены вне любой из функций, следовательно, доступны для многих из них. Область внешней переменной простирается от точки во входном файле, где она объявлена, и до конца файла. Если внешняя переменная определена в другом файле, то вступает в силу описание extern (внешний). На рис.1 показано, где объявляются и на что распространяется область действия внешних переменных, если программа main и вызываемая функция находятся в данном файле. На рис. 2 демонстрируются отличия, имеющие место, когда main и вызываемая функция находятся в разных файлах. В файле с вызываемой функцией внешние переменные будут доступны после их описания с помощью ключевого слова ***extern***.

Один файл

#include <stdio.h>

Объявление внешних переменных

main( )

{

тело программы

}

#include “имя файла”

функция

{ extern

Описание внешних переменных

Тело функции

}

Область действия внешних переменных

Область действия внешних переменных с учетом описания extern

рис. 2

#include <stdio.h>

Объявление внешних переменных

main ( )

{

тело программы

функция

тело функции

}

Область действия внешних переменных

Рис. 1

**Пример 5.** Оформить в виде функции вычисление выражения:

f=a\*x2+b\*x+c;

В приведенной ниже программе заданные переменные объявлены как внешние, причем основная программа и функция находятся в одном файле.

**#include <stdio.h>**

**int a=5, b=7, c=10,x; /\* Объявление внешних
 переменных a,b,c,x целого типа\*/**

**main ()**

**{ int f;**

**scanf (“%d”, &x); /\*Ввод значения переменной x\*/**

**f=kv(); /\*обращение к функции\*/**

**printf (“%d”,f); /\*вывод на экран значения переменной f\*/**

**}**

**/\*функция\*/**

**kv()**

**{int f;**

**f=a\*x\*x+b\*x+c; /\*вычисление значения f\*/**

**return (f); /\*возвращает значение f вызывающей программе\*/**

**}**

Если сравнить эту программу с программой, приведенной в примере 2, то можно обнаружить два различия:

1. после имени функции в скобках отсутствуют аргументы;
2. в функции не объявлены переменные, с которыми работает функция.

Это стало возможным потому, что переменные объявлены внешними, а значит они известны всему файлу, в том числе и функции.

Внешние переменные должны быть описаны до функции main(). Только в этом случае они становятся внешними (см. рис. 1).

Приведем программу для этого же примера, рассмотрев случай, когда основная программа и функция расположены в разных файлах.

**#include <stdio.h>**

**int a=5, b=7, c=10,x,f; /\* Объявление внешних переменных a,b,c,x,f целого типа\*/**

**main ()**

**{**

**scanf (“%d”, &x); /\*Ввод значения переменной x\*/**

**f=kv(); /\*обращение к функции\*/**

**printf (“%d”,f); /\*вывод на экран значения переменной f\*/**

**}**

**#include “kv.c” /\*включение файла kv.c функцией kv\*/**

**/\*функция\*/**

**kv()**

**{extern int a,b,c,x,f;**

**f=a\*x\*x+b\*x+c; /\*вычисление значения f\*/**

**return (f); /\*возвращает значение f вызывающей программе\*/**

**}**

Как было сказано выше (см. рис. 2), если основная программа и функция расположены в разных файлах, то переменные в функции должны быть вписаны при помощи ключевого слова ***extern***.

Рассмотрим теперь статические переменные. Статические переменные имеют такую же область действия, как автоматические, но они не исчезают, когда содержащая их функция закончит свою работу. Компилятор хранит их значения от одного вызова функции до другого. Статические переменные объявляются с помощью ключевого слова static. Можно статические переменные описать вне любой функции. Это создает внешнюю статическую переменную. Разница между внешней переменной и внешней статической переменной заключается в области их действия. Обычная внешняя переменная может использоваться функциями в любом файле (с помощью ключевого слова extern), в то время как внешняя статическая переменная может использоваться только функциями того же самого файла.

Регистровые переменные относятся к последнему классу. Ключевое слово register указывает, что переменная, о которой идет речь, будет интенсивно использоваться. Если это возможно, значения таких переменных помещаются во внутренние регистры процессора, благодаря чему программа будет более быстрой.

2. Задания

Взять задачу из лабораторной работы №4 и оформить ее решение в виде функции следующими способами:

1. функция расположена после ее вызова;
2. функция расположена после до ее вызова;
3. функция расположена после в другом файле;

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьютер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №6
“Работа со структурами в языке программирования С++”

**Цель работы:** познакомиться с понятием структуры и структурной переменной. Научиться создавать массивы структур и работать со вложенными структурами.

1. Теоретические сведения

**Структура** – это объединение одного либо более объектов(переменных, массивов, указателей, других структур). Как и массив, она представляет собой совокупность данных, но отличается от него тем, что к ее элементам необходимо обращаться по имени, и ее различные элементы не обязательно должны принадлежать одному типу.

Структуры удобно использовать там, где разнообразные данные, относящиеся к одному и тому же объекту, необходимо объединять. Например, ученика средней школы характеризуют следующие данные: фамилия, имя, дата рождения, класс, возраст.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым следует ее тип, список элементов, заключенных в фигурные скобки. Ее можно представить в следующем общем виде:

struct тип {тип элемента 1 имя элемента 1;

тип элемента n имя элемента n; };

Именем элемента может быть любой идентификатор. В одной строке можно записывать через запятую несколько идентификаторов одного типа.

Например:

***struct date { int day;***

***int month;***

***int year;***

 ***};***

Русские буквы использовать в идентификаторе в языке С++ нельзя.

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например:

struct date {…} a, b, c;

При этом выделяется соответствующая память.

Выведенное имя типа можно использовать для объявления записи, например: struct date day;. Теперь переменная day имеет тип date.

Разрешается вкладывать структуры одна на другую. Для лучшего восприятия структуры используем русские буквы в идентификаторах, в языке С++ этого делать нельзя.

Например:

struct УЧЕНИК { char Фамилия [15];

имя [15];

struct DATA ДАТА РОЖДЕНИЯ;

int класс, возраст;};

определенный выше тип DATA включает три элемента: День, Месяц, Год, содержащие целые значения (int). Запись УЧЕНИК включает элементы: ФАМИЛИЯ [15]; ИМЯ[15]; ДАТА РОЖДЕНИЯ, КЛАСС, ВОЗРАСТ. ФАМИЛИЯ [15] и ИМЯ [15] – это символьные массивы из 15 компонент каждый. Переменная ДАТА РОЖДЕНИЯ представлена составным элементом ( вложенной структурой) ДАТА. Каждой дате рождения соответствуют день месяца, месяц и год. Элементы КЛАСС и ВОЗРАСТ содержат значения целого типа (int). После введения типов ДАТА и УЧЕНИК можно объявить переменные, значения которых принадлежат этим типам.

Например:

struct УЧЕНИК УЧЕНИКИ [50];

массив УЧЕНИКИ состоит из 50 элементов типа УЧЕНИК.

В языке С++ разрешено использовать массивы структуры; записи могут состоять из массивов и других записей.

Чтобы обратиться к отдельному компоненту структуры, необходимо указать ее имя, поставить точку и сразу за ней написать имя нужного элемента.

Например:

Ученики [1]. КЛАСС = 3;

Ученики [1]. ДАТА РОЖДЕНИЯ. ДЕНЬ=5;

Ученики [1]. ДАТА РОЖДЕНИЯ. МЕСЯЦ=4;

Ученики [1]. ДАТА РОЖДЕНИЯ. ГОД=1979;

Первая строка указывает, что 1-й ученик учится в третьем классе, а последующие строки – его дату рождения: 5.04.79.

Каждый тип элемента структуры определяется соответствующей строкой объявления в фигурных скобках. Например, массив УЧЕНИКИ имеет тип УЧЕНИК, год является целым числом. Так как каждый элемент записи относится к определенному типу, его составное имя может появляться везде, где разрешено использовать значение этого типа. Рассмотрим пример программы:

**/\* Демонстрация записи \*/**

**#include <stdio.h>**

**struct computer {**

 **int mem;**

 **int sp;**

 **char model [20]; };**

**/\* Объявление записи типа computer, состоящей из трех элементов: mem, sp, model \*/**

**struct computer pibm =**

**{512, 1, ”ПЭВМ ЕС 1840.05”}**

**/\* Объявление и инициализация переменной pibm типа computer \*/**

**main ( )**

**{ printf (” персональная ЭВМ %s\n\n ”, pibm.model);**

**printf (”объем оперативной памяти - %d Кбайт \n”, pibm.mem);**

**printf (”производительность - %d млн. операций в секунду \n”, pibm.sp);**

**/\* вывод на экран значений элементов структуры \*/**

**}**

В данной программе объявляется запись computer, которая состоит из трех элементов: mem ( память ЭВМ ), sp (быстродействие), model [20] (модель ПЭВМ). Переменная pibm имеет тип computer и является глобальной. Строки pibm.model, pibm.mem, pibm. sp в операторе printf вызывают обращение к соответствующим элементам записи pibm типа computer , которым ранее были присвоены определенные значения.

Результат работы программы имеет вид:

**персональная ЭВМ ПЭВМ ЕС 1840.05**

**объем оперативной памяти – 512 К байт**

**производительность – 1 млн. операций в секунду**

Рассмотрим использование в программе вложенных структур:

**/\* Демонстрация вложенных структур\*/**

**# include <stdio.h>**

**struct date { int day;**

**int month;**

**int year;};**

**/\* Объявление записи типа date\*/**

**struct person { char fam [20];**

**char im [20];**

**char ot [20];**

**struct date f1;};**

**/\* Объявление структуры типа person;одним из элементов записи person является запись f1 типа date \*/**

**main ( )**

**{ struct person ind1;**

**/\* обьявление переменной ind1 типа person \*/**

**printf (”Укажите фамилию, имя, отчество, день, \n месяц”**

**” и год рождения гражданина ind1\n”);**

**scanf (” %S %S %S %d %d”, &ind1.fam, &ind1.im, &ind1.ot,**

**& ind1.f1.day, &ind1.f1.month, &ind1.f1.year );**

**/\* Ввод сведений о гражданине ind1 \*/**

**printf (” Фамилия, имя, отчество: % S % S % S \n”, ind1.fam, ind1.im, ind1.ot);**

**printf (” Год рождения - % d \n”, ind1.f1.year);**

**printf (” Месяц рождения - % d -й \n”, ind1.f1.month);**

**printf (” День рождения - % d -й \n”, ind1.f1.day);**

**/\* Вывод сведений о гражданине ind1 \*/**

**}**

Структура типа date ( дата) содержит три элемента: day (день), month ( месяц ), year (год). Структура типа person (человек) содержит четыре элемента: fam[20] (фамилия), im[20] (имя) , ot[20] (отчество), f1 (дата рождения). Последний из них (f1) – это вложенная запись типа date.

Результаты работы программы:

**Укажите фамилию, имя, отчество, день, месяц и год рождения гражданина ind1**

**Алексеев**

**Сергей**

**Петрович**

**3**

**5**

**1978**

Подчеркнутая информация вводится пользователем.

**Сведения о гражданине ind1**

**Фамилия, имя, отчество: Алексеев Сергей Петрович**

## Год рождения – 1978

**Месяц рождения – 5-й**

**День рождения – 3-й**

В следующей программе рассмотрим использование структуры в виде элементов массива pibm. Каждый элемент состоит из следующих компонентов: mem (память), sp (объем винчестера), model [20] ( модель ПЭВМ):

**/\* Массивы записей \*/**

**#include <stdio.h>**

**struct computer {**

**int mem, sp;**

**char model [20];**

**pibm [10];};**

**/\* объявление записи типа computer;**

**объявление массива pibm типа computer \*/**

**main ( )**

**{ int i, j, k, priz;**

**for ( i=0; i<10; i++)**

**{ printf (”Введите сведения о ПЭВМ %d и признак (0-конец;**

**\n другая цифра- продолжение)\n”, i);**

**printf (” модель ПЭВМ - ”);**

**scanf (”%S”, &pibm [i].model );**

**printf (”объем оперативной памяти -”);**

**scanf (”%d”, &pibm[i].mem);**

**printf (” объем винчестера - ”);**

**scanf (”%d , &pibm[i].sp ”);**

**printf (“признак - ”);**

**scanf (” %d ”, &priz );**

**k=i;**

**if (!priz) break; }**

**/\* Здесь !priz – операция отрицания priz; break – выход из цикла for, если priz=0 \*/**

**for (i=0; i<10, i++);**

**{**

**printf (”\n О какой ПЭВМ Вы хотите получить сведения?\n (Введите номер от 0 до 9)\n” );**

**scanf (”%d ”,&j );**

 **if (j>k)**

**{ printf (”Нет сведений об этой ПЭВМ \n”);**

**continue; }**

**printf (” персональная ЭВМ %s\n ”, pibm[j].model);**

**printf (”объем оперативной памяти - %d Мб\n ”, pibm[j].mem);**

**printf (”объем винчестера - %d Мб \n ”, pibm[j].sp);**

**printf (” признак – ” );**

**scanf (” %d ”, &priz);**

**if (!priz) break; }**

**/\* Ввод сведений о ПЭВМ и занесение в массив pibm записей типа computer (первый цикл for); вывод на экран сведений о ПЭВМ (второй цикл for) \*/**

**}**

Результаты работы программы:

**Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение)**

**модель ПЭВМ – АТ 486 DX**

**объем оперативной памяти – 32**

**объем винчестера – 4 Гбайта**

**признак – 1**

**Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение)**

**модель ПЭВМ – АТ 386 DX**

**объем оперативной памяти – 64**

**объем винчестера – 14 Гбайт**

**признак – 0**

**О какой ПЭВМ Вы хотите получить сведения? (Введитн номер от 0 до 9)**

**1**

**модель ПЭВМ – АТ 386 DX**

**объем оперативной памяти – 16 Мб**

**объем винчестера – 2,5 Гбайт**

**признак – 0**

**2. Задания**

Из таблицы взять задание по варианту и написать программу.

3. Содержание отчета

Отчет должен содержать:

1. задание к работе;
2. программу;
3. результаты расчетов.

|  |  |
| --- | --- |
| **№ вар.** | **Задание** |
| 1 | Опишите запись СТУДЕНТ и поместите в нее следующую информацию: Ф.И.О., оценки (математика, физика, черчение, химия, сопромат). Определите, сколько студентов имеют неудовлетворительную оценку по математике. |
| 2 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, сколько студентов имеют неудовлетворительную оценку хотя бы по одному предмету. |
| 3 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, сколько студентов сдали все экзамены на 5. |
| 4 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите средний балл группы по физике. |
| 5 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите количество отличных оценок, полученных группой по всем предметам. |
| 6 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, сколько студентов имеют средний балл от 4 до 5. |
| 7 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, какое количество неудовлетворительных оценок получено по всем предметам. |
| 8 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, какой из предметов был сдан группой лучше всего. |
| 9 | Воспользовавшись записью СТУДЕНТ из варианта №1, определите, сколько студентов не имеют задолженностей. |
| 10 | Опишите запись АНКЕТА и поместите в нее следующую информацию: Ф.И.О.(фамилия, имя, отчество), адрес (улица, номер дома, номер квартиры), пол, возраст. Определите, сколько лиц женского и сколько мужского пола проживают в одном доме. |
| 11 | Воспользовавшись записью АНКЕТА из варианта №10, определите, сколько лиц мужского пола в возрасте старше 18 лет и младше 60 проживают на одной улице. |
| 12 | Воспользовавшись записью АНКЕТА из варианта №10, определите, сколько лиц женского пола в возрасте старше 30 лет проживают в одном доме. |
| 13 | Воспользовавшись записью АНКЕТА из варианта №10, определите, сколько детей до 7 лет проживают на одной улице. |
| 14 | Воспользовавшись записью АНКЕТА из варианта №10, определите, сколько лиц мужского пола и женского в возрасте до 50 лет проживают на одной улице. |
| 15 | Воспользовавшись записью АНКЕТА из варианта №10, определите, сколько детей от 1 года до 5 проживают в одном доме. |
| 16 | Опишите запись ТРАНСПОРТ и поместите в нее следующую информацию: Ф.И.О. (фамилия, имя, отчество пассажира), багаж (количество вещей, вес в кг). Определить число пассажиров, вес багажа которых превышает 30 кг. |
| 17 | Воспользовавшись записью ТРАНСПОРТ из варианта №16, определите, имеется ли пассажир, багаж которого состоит из одной вещи весом в 20 кг. |
| 18 | Воспользовавшись записью ТРАНСПОРТ из варианта №16, определите средний вес багажа. |
| 19 | Воспользовавшись записью ТРАНСПОРТ из варианта №16, определите количество пассажиров, вес багажа которых превосходит средний. |
| 20 | Воспользовавшись записью ТРАНСПОРТ из варианта №16, определите количество пассажиров, имеющих более трех вещей. |

**4. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьютер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №7
Препроцессорные средства в языке программирования С++

**Цель работы:** ознакомиться с назначением и особенностями использования директив препроцессора. Научиться применять препроцессор для замены в тексте, для работы с макроподстановками, а также для условной компиляции программ.

1. Теоретические сведения

* 1. Препроцессор языка С++

**1.1.1. Директива # include, # define, # undef**

Транслятор С++ имеет встроенное средство, расширяющее возможности языка и называемое препроцессором. Он рассматривает программу до компиляции (отсюда и термин препроцессор ) и заменяет символические аббревиатуры в программе на соответствующие директивы, подключает указанные файлы. Для препроцессора строки программы, начинающиеся с символа #.

Его команда:

#include ”имя файла”

уже использовалась в работе, связанной с процедурами, для подключения файлов.

Рассмотрим другую директиву препроцессора

**# define <идентификатор> <подстановка>**

**# define FOOR TWO\*TWO**

**# define PX printf(“x равен %d\n”,x)**

**#define FNT “x равен % d\n”**

Эта директива вызывает замену в последующем тексте названного идентификатора на текст подстановка.

Если директива имеет вид:

# define идентификатор (идентификатор ,…идентификатор)

то это определение макроподстановки с аргументами. Её использование рассматривается ниже на примерах.

**Пример 1.**

**# include<stdio.h>**

**# define TWO 2**

**main( )**

**{ int x = TWO;**

**PX;**

**printf(FNT, X);**

**}**

После выполнения программы на экран будет выведена следующая информация:

**Х равен 2**

**Х равен 4**

Рассмотрим подробно, что произошло при выполнении программы.

Оператор

int x=TWO; превращается в int x=2;

Т.е. слово TWO заменилось цифрой 2

Оператор

PX; превращается в printf(“x равно %d\n”,x);

Оператор

x=FOOR; превращается в x=TWO\*TWO.

а далее в х=2\*2;

Оператор

printf(FNT,X); превращается в printf(“x равен %d\n”,x);

Рассмотрим пример директивы # define с аргументами. Работать с такими директивами нужно внимательно, в противном случае можно получить неожиданные результаты.

Покажем это на примере:

**Пример 2.**

**#include<stdio.h>**

**#define SQUARE(X) X\*X**

**#define PR(X) printf(“X равен %d\n”,X)**

**main( )**

**{ int x=4;**

**int z;**

**z = SQUARE(x);**

**PR(z);**

**PR(SQUARE(z));**

**PR(SQUARE(x+2));**

**pr(100/SQUARE(2));**

**}**

Всюду, где в программе SQUARE(x) оно заменяется на ***x\*x***.

Результат работы программы:

z равно 16

Z равно 4

SQUARE (x+2) равно 14

100/SQUARE (2) равно 100

Первые две строки предсказуемы. Заметим, что даже внутри кавычек в определении PR переменная замещается соответствующим аргументом.

Строка

PR(SQUARE (x));

замещается на следующую

printf(“SQUARE(x) равно %d\n”,SQUARE (x));

после первого этапа макрорасширения. Второе SQUARE (х) превращается в Х\*Х, а первое остаётся без изменении, потому что теперь находится внутри кавычек в операторе программы и защищено от дальнейшего расширения. Окончательно строка программы содержит printf(“SQUARE (x) равно %d\n”,X\*X); и выводит на экран.

SQUARE (х) равно 16

Теперь мы добрались до несколько специфических результатов.

Вспомним, что Х имеет значение 4. Это позволяет предположить, что SQUARE (Х+2) будет равно 6\*6 или 36. Но напечатанный результат говорит, что получается 14. Причина этого несоответствия состоит в том, что препроцессор не делает вычислении, он только замещает сторону, таким образом, везде вместо Х будет подставлено Х+2. Значит Х\*Х ставится Х+2\*Х+2. Если Х равен 4, то 4+2\*4+2=14

Если всё-таки требуется, чтобы SQUARE (х+2) было равно 36, то необходимо записать так:

# define SQUARE(X) (X)\*(X)

Тогда SQUARE (Х+2) становится (Х+2)\*(Х+2), и мы получим желаемый результат.

Рассмотрим следующий результат. Выражение 100/SQUARE (Х) превращается в 100/2\*2, т.е. равно 100.

Эту путаницу можно исправить, определив SQUARE (Х) следующим образом

# define SQUARE (X) (X)\*(X)

это даёт 100/(2\*2)

Чтобы выполнить два последующих примера, нам необходимо определение

# define SQUARE (Х) (Х)\*(Х)

**Пример 3.** В приведённой программе открывающая и закрывающая фигурные скобки заменены словами **begin** и **end.** Определены также другие названия для функции ввода-вывода. В результате получилось что-то среднее между программой на Паскале и СИ. Таким образом, пользователь, работающий с другим языком может строить привычные для себя конструкции.

**#include <stdio.h>**

**#define write(valw) printf(“%d”,valw) /\*выполняется замена строки write(valw) на строку printf(“%d”,valw)\*/**

**#define read(valr) scanf (“%d”,$valr) /\* выполняется замена строки read(valr) на строку scanf(“%d”,&valr)\*/**

**#define begin { /\* выполняется замена слова begin на скобку {\*/**

**#define end } /\* выполняется замена слова end на скобку }\*/**

**main( )**

**{begin**

 **int i;**

 **read (i); /\*вводится с клавиатуры значение переменной i \*/**

 **write (i); /\*выводится на экран значение i \*/**

**end**

**}**

Директива #undef отменяет самое последнее определение поименованного макроопределения. Она имеет следующий вид :

#undef идентификатор

где идентификатор – идентификатор , раннее определённый в директиве #define. Например:

#undef ESCAPE

после выполнения этой процедуры идентификатор ESCAPE становится неопределённым.

**1.1.2. Директивы #if, #ifdef, #ifndef, #else, #endif.**

Директивы. рассматриваемые в этом разделе позволяют выполнить условную компиляцию. Условная компиляция- это выборная компиляция только тех частей программы, которые удовлетворяют некоторым условиям. Одна из целей условной компиляции - сделать программу более мобильной. Изменяя несколько ключевых определений в начале файла, вы можете устанавливать различные значения и включать различные файлы для разных систем; таким образом, ненужный код не хранится в памяти во время выполнения. Решение о включении той или иной части программы применяется на этапе компиляции, а не во время выполнения.

Директива #if похожа на обычный оператор if:

#if константное выражение.

Она проверяет, будет ли отличаться от нуля выражение, составленное и константное. Выражение истинно, если оно не равно нулю. Например:

#if sys==”IBM”

#include “ibm.h”

#endif

Как видно из примера, оно заканчивается директивой #endif.

Директива

#ifdef идентификатор

Устанавливает, определён ли в данный момент идентификатор, т.е. входил ли он в команду вида #define.

Директива

#ifndef идентификатор

проверяет, не определен ли в данный момент указанный идентификатор.

Например:

#ifndef SIZE

#define SIZE 128

#endif

В результате переменная SIZE получает значение 128, если ранее она не была определена программистом. За любой из перечисленных выше директив может следовать произвольное количество строк.

Указанные директивы могут быть применены вместе с инструкцией #else. Назначение этой инструкции такое же, как и в условном операторе. Рассмотрим пример.

**#ifdef MAVIS**

**#include “horse.h”**

**#define STABLE 5**

**#else**

**#include “cow.h”**

**#define STABLE 15**

**#endif**

Директива **#ifdef** сообщает, что если идентификатор MAVIS определен препроцессором, то выполняются все последующие директивы вплоть до #else. В противном случае будут выполнены директивы, стоящие после #else до директивы #endif.

**2. Задание**

Взять задание из лабораторной работы №4. Написать программу, используя операторы Паскаля. Применив директивы препроцессора языка С++, обеспечит замену операторов Паскаля на операторы языка С++; выполнить программу.

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно- ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьютер, 1993. - 160 с.
5. Уинер Р. Язык турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №8
“Работа с файлами в языке программирования С++”

**Цель работы:** научиться работать с файлами прямого и последовательного доступов. Ознакомиться с функциями, позволяющими создать, дополнить и прочитать файл, а также с функциями, обеспечивающими запись информации в файл и считывающими информацию из файла.

1. Теоретические сведения

1.1. Файлы последовательного доступа

При последовательном доступе обмен информации производится через специальный буфер, резервируемый системой ввода-вывода. Компиляция языка С++ рассматривает ввод-вывод как поток файлов, которые поступают последовательно, байт за байтом. Каждый поток связывается с файлом на магнитном диске или с файлом, который поставлен в соответствии физическому устройству, например клавиатуре. Связь потока с файлом устанавливается при его открытии. Открытие файла осуществляется функцией ***fopen***. Данная функция возвращает указатель на файл. Указатель на файл необходимо объявлять, например:

FILE \* lst;

Здесь FILE-имя типа, описанное в стандартном определении ***stdio.h***;

lst ‑ указатель на файл (логическое имя).

Обращение к функции fopen в программе производится так:

lst = fopen (физическое имя файла, вид использования файла);

 Физическое имя файла может, например, быть “prn”‑ для устройства печати;

“b:zni.f”-для файла zni.f на дискете b:

Вид исполнения.

Если файл открывается для записи или дополнения, но ещё не существует, то он создаётся. Открытие существующего файла для записи приводит к уничтожению его старого содержимого. Попытка прочитать несуществующий файл-это ошибка (***fopen*** выдаёт указатель со значением NULL).

Для работы с файлами используются библиотечные функции fprintf, fscanf, fgets, fputs. Их применение рассмотрим ниже на примерах. После окончания работы с файлом он должен быть закрыт. Это делается с помощью библиотечной функции fclose, например:

fclose (lst)

lst ‑ указатель на файл;

Рассмотрим организацию ввода информации на печатающее устройство:

**/\*Работа с файлами (вывод на печатающее устройство )\*/**

**#include<stdio.h>**

**main ( )**

**{**

**int i = 150;**

**FILE \*lst;**

**/\*lst-указатель на файл ( объект типа FILE)\*/**

**lst= fopen (“prn”,”w”);**

***lst*** получает адрес открытого файла с именем ***prn***, предназначенного для записи в него информации (символ ***w***);

***prn*** ‑ стандартное имя устройства печати, информация из файла с адресом ***lst*** будет выводится на печатающее устройство.

***fprintf(lst,”\n число i = %d\n”,i) ;***

первый параметр функции ***fprintf*** ‑ это указатель на соответствующий файл; на печать будет выведена строка: число i = 150.

***fclose (lst);***

функция fclose закрывает файл с указателем lst теперь ссылку lst можно использовать для другого файла.

Функция ***fprintf*** подобна функции ***printf*** и отличается от неё тем, что в качестве первого параметра использует указатель на соответствующий файл.

Следующая программа показывает организацию вывода информации на дискету:

/\* Работа с файлами (запись информации в файл на дискету)\*/

**#include < stdio.h >**

**main ( )**

**{**

**int i = 150**

**FILE \*rsd;**

**rsd = fopen(“B:ZNI.F”,”r ”);**

**/\*второй параметр функции fopen-теперь “r ”,говорящий о чтении информации\*/**

**fscanf(rsd,”%d”,&i);**

**/\*из файла ZNI.F(дискета В:) будет прочитано значение i\*/**

**printf(“число i=%d(начальное значение i)\n”,i);**

**while (fscanf(rsd,”%d”,&i)!=EOF)**

**printf(“число i = %d\n,i”);**

**/\*последовательный вывод целых чисел из файла; вывод прекращается, когда будет достигнут конец файла(ЕОF)\*/**

**fclose(rsd);**

**}**

Первая функция ***fscanf*** обеспечивает чтение из файла B:ZNI.F значение целого числа ***i***. Затем полученное значение выводится функцией ***printf*** на экран дисплея. Последующий фрагмент программы (начиная с оператора ***while***) позволяет вывести другие целые числа из этого файла, если их там не более одного. После того как чисел в файле не остаётся, функция ***fscanf*** выдаёт особое значение EOF-признак конца файла. Его можно использовать для прекращения чтения информации.

Добавим в файл ZNI.F новые данные:

**/\*Работа с файлами (дополнение файла на дискете)\*/**

**#include <stdio.h>**

**main ( )**

**{**

**int c;**

**FILE \* lds;**

**lds = fopen(“B:ZNI.F”,”a”);**

**/\*Второй параметр функции fopen – теперь “а” , говорящий о возможности дополнения файла\*/**

**puts (“введите целое число”);**

**scanf (“%d”,&c);**

**fprintf (lds,”%d\n”,c);**

**/\*файл ZNI.F будет дополнен первым числом\*/**

**fclose (lds);**

**}**

В ответ на сообщение “введите целое число” необходимо ввести его, и файл добавится одним числом. Если необходимо добавить в файл несколько данных, можно организовать цикл.

С началом работы любой программы автоматически открываются три файла, и для них определяются соответствующие указатели. Первый из них - это файл для стандартного ввода информации с клавиатуры с указателем *stdin*, второй – для стандартного вывода информации на экран дисплея с указателем *stdout*, третий – для стандартного вывода ошибок на экран дисплея с указателем *stderr*. Объекты *stdin, stdout, stderr* – константы, а не переменные, и им нельзя что-либо присваивать. Файл с указателем stderr обычно используется для хранения различных диагностических сообщений.

Рассмотрим ввод и запись строк в файл. Продемонстрируем вывод информации в стандартный файл для ошибок:

/\*Работа с файлами (ввод и запись строк). Вывод информации в стандартный файл для ошибок\*/ .

**#include <stdio.h>**

**main ( )**

**{**

**char str[50];**

**FILE \*FL**

**FL=fopen ( “FAL.F”, “w”) ;**

**puts ( “Введите строку ( до 49 символов)”);**

**scanf ( “%s”, str);**

**fputs ( str, FL);**

**/\* Функция fputs записывает в файл строку str\*/**

**fprintf ( stderr, “Демонстрация вывода в стандартный файл для ошибок ”);**

**/\* текст в функции fprintf будет выведен на экран\*/**

**fclose ( FL );**

**FL=fopen ( “FAL.F”, “r”);**

**/\* Файл открывается для чтения \*/**

**fgets ( str, 20, FL);**

**/\* функция fgets считывает из файла FAL.F строку str, максимальная длина которой 20-1=19 ( 20- второй параметр fgets); получившаяся строка оканчивается символом ‘\0’\*/**

**printf ( “str = %s”, str);**

**fclose ( FL);**

**}**

После запуска программы на выполнение необходимо ввести строку до 49 символов. На экране появится:

**Введите строку ( до 49 символов )**

Введем строку:

**Технический университет**

**Демонстрация вывода в стандартный файл для ошибок**

Подчеркнутая строка будет записана в файл FAL.F с помощью функции fputs, первый параметр которой ( str ) – идентификатор записываемой строки, а второй (FL)– указатель на соответствующий файл.

Чтение строки обеспечивает функция fgets, первый параметр которой ( str ) – идентификатор читаемой строки, второй – его максимальная длина и третий ( FL) – указатель на соответствующий файл.

В результате на экране появляется строка:

**str = “ Технический университет ”**

В строке 19 символов, поскольку 20-й символ – это заключительный нуль (‘\0’).

Рассмотрим пример создания файла и чтения из него информации по определенному условию. Информация, входящая в файл: Ф.И.О. студента, группа, размер стипендии. Требуется получить информацию о студенте, фамилия которого будет введена.

Программа приведена на рис.1. В начало программы необходимо поместить процедуру #include < string.h >, так как в программе будут применены функции, работающие со строковыми переменными. Для удобства работы в программе сразу за описанием типа переменных применен оператор очистки экрана – clrscr (). Создание файла происходит в цикле while. При создании файла student.dat используется признак окончания ввода информации, обозначенный переменной Р. До тех пор, пока переменная Р не получит нулевое значение, будет выполняться цикл while и, следовательно, ввод информации.

После того, как файл создан и закрыт, в переменную fio1 вводится фамилия студента. Далее идет цикл while, в котором читается файл и осуществляется поиск введенной фамилии. Рассмотрим подробно операторы, входящие в этот цикл.

Работа функции fscanf описана выше. В следующем операторе if применена функция feof, аргументом которой является указатель на файл. Эта функция определяет, достигнут ли конец файла. Если конец файла достигнут, то оператор break прервет выполнение цикла. Работа двух последующих операторов fscanf не требует объяснений. Далее в программе применен оператор if. Длина определяется функцией strlen. Если сравниваемые строки оказались одинаковой длины, то выполняется функция strncmp, сравнивающая строки. Первыми двумя аргументами этой функции являются сами строки, а третьим аргументом – длина строки. Если строки совпали, то выводится информация о студенте, фамилия которого была введена.

**#include <stdio.h>**

**#include <string.h>**

**main ( )**

**{**

**char fio [15], gr [6], fio1 [15];**

**float st;**

**int p=1;**

**FILE \*lf;**

**clrscr ( );**

**lf=fopen ( “student.dat”, “w”);**

**while (p)**

**{ printf (“”);**

**printf (“\n”);**

**scanf (“%s “,&fio);**

**fprintf ( lf, “%s”, fio);**

**printf (“\n”);**

**scanf (“%s”, &gr);**

**fprintf ( lf, “%s”,gr);**

**printf (“\n”);**

**scanf ( “%f”,&st);**

**fprintf ( lf, “%f\n”,st);**

**printf (“\n”);**

**scanf ( “%d”,&p);**

**}**

**fclose (lf);**

**printf ( );**

**scanf ( “%s”,&fio1);**

**lf=fopen (“student.dat”,”r”);**

**while (1);**

**{**

**fscanf (lf,”%s”,&fio);**

**if (feof (lf)) break;**

**fscanf (lf,”%s”,&gr);**

**fscanf (lf,”%f”,&st);**

**if ( strlen (fio)=strlen (fio1))**

**if (strncmp(fio,fio1,strlen(fio))=0)**

**printf (“%s %s %f\n”, fio, gr, st);**

**}**

**fclose (lf);**

**}**

Рис.1.

**1.2 Файлы произвольного доступа**

Для выполнения произвольного доступа применяется функция fseek, позволяющая обрабатывать файл подобно массиву и непосредственно достигать любого определенного байта. Эта функция имеет вид:

fseek ( stream, offset, origin );

stream – указатель на файл;

оrigin – указатель направления отсчета.

Функция перемещает ( внутренний ) указатель файла, связанного с потоком stream на новое место в файле, которое вычисляется по смещению offset и указанию направления отсчета origin.

Следующая операция с указанным потоком stream будет выполнена, начиная с той позиции, на которое произошло перемещение. Аргумент origin должен быть одной из следующих констант:

0 – начало файла;

1 – текущая позиция указателя файла;

2 – конец файла.

Для того, чтоб перемещать указатель, необходимо знать количество символов в записи. Для дальнейших рассуждений обозначим номер записи через n, а количество символов в записи – через m.

Рассмотрим ситуацию, когда origin = 0. Тогда для того, чтобы установить указатель на запись с номером n offset должен быть вычислен по следующей формуле:

оffset = ( n-1)\*m

Рассмотрим вторую ситуацию, когда origin = 1. Установим указатель на начало файла (offset=0). Для того, чтобы перейти на следующую ( вторую ) запись offset должен быть равен нулю. Для перехода с первой на третью запись offset = m и т.д.

Можно двигаться не только от начала к концу файла, но и в обратном направлении. Для этого offset должен быть отрицательным. Например, если указатель стоял на пятой записи, то для того, чтоб перейти на четвертую offset = -2\*m, а для перехода с пятой на третью offset = -3\*m и т.д .

Рассмотрим третью ситуацию, когда origin = 2. В этом случае указатель перемещается от конца файла. Параметр offset может быть как с плюсом, так и с минусом. Например, для того, чтобы переместить указатель на предпоследнюю запись, offset должен быть вычислен по формуле: offset = 2\*m

При работе с файлами прямого доступа можно выйти за начало или конец файла, поэтому при работе необходимо контролировать эти ситуации, следовательно, необходимо знать размер файла. Приведенные ниже три оператора позволяют определить размер файла.

handle =open ( “имя файла”, O\_CREATE);

l = file length (handle);

close (handle);

Переменная ***handle*** имеет тип int, а переменная 1 так же, как переменная ***offset*** – тип long. После выполнения приведённых выше операторов размер файла содержится в переменной 1. Для работы операторов, определяющих размер файла, перед программой необходимо добавить процедуры:

#include <fcnt.h>

#include <io.h>

На рис.2 представлена программа, демонстрирующая прямой доступ. Первая часть программы, обеспечивающая создание файла практически такая же, как и предыдущая. Исключение состоит в том, что для каждой переменной в операторе ***fprintf*** определён размер.

Длина записи определяется, как сумма размеров определённых переменных плюс единица, т.е. для нашего примера n = 28. Далее идёт группа операторов, определяющих длину файла.

Следом идёт оператор, инициализирующий переменную n. Переменная n несёт двойную нагрузку. С одной стороны она является признаком окончания цикла (входит в оператор while), с другой определяет номер записи. Затем вычисляется значение переменной offset по формуле, смысл которой описан выше. Эта формула применена в данной программе, т.к. используемый в дальнейшем оператор fseek работает в параметром origin равным нулю. Оператор if контролирует выход указателя за начало или конец файла. Если выведено значение n равное нулю, то цикл прекращается. Если произошёл выход за конец файла (offset >= 1), то игнорируются все последующие операторы цикла, но работу можно будет продолжать, т.е. ввести следующий номер записи. Если выход за пределы файла не произошёл, то выполняются операторы цикла, смысл которых не требует дополнительных пояснений.

**#include <stdio.h>**

**#include <fcntl.h>**

**#include <io.h>**

**main ( )**

**{**

**char fio[10], gr [5];**

**float st;**

**long offset,l;**

**int p=1, handle, n;**

**FILE \*lf;**

**clrscr();**

**lf=fopen(“student.dat”,”w”);**

**while(p)**

**{**

**printf(“Введите сведения о студенте”);**

**printf(“\n Введите Ф.И.О.”);**

**scanf(“%s”,&fio);**

**printf(“\n Введите номер группы”);**

**scanf(“%s”,&gr);**

**printf(“\n Введите размер стипендии”);**

**scanf(“%s”,&st);**

**fprintf(lf,“%15s%6s%6.2f ”,fio,gr,st);**

**printf(“\n Признак окончания ввода”);**

**scanf(“%s”,&p);**

**}**

**fclose (lf);**

**handle = open(“student.dat”,O\_CREAT);**

**l=filelength(handle);**

**close(handle);**

**n=1;**

**lf=fopen(“student.dat”,”r ”);**

**while(n)**

**{**

**printf (“Введите номер о записи”);**

**scanf (“%d”,&n);**

**offset=(n-1)\*28;**

**if (offset>=1 | | n==0) continue;**

**fseek (lf,offset,0);**

**fscanf (lf,”%s %s %f ”,&fio,&gr,&st);**

**printf (“%15s%6s%6.2f\n”,fio,gr,st);**

**}**

**fclose(lf);**

**}**

Рис.2.

**2. Задания**

2.1. Создать файл последовательного доступа. Добавить к файлу несколько записей. Прочитать файл, выполнив указанное в задании действие.

2.2. Создать файл прямого доступа. Прочитать одну из записей созданного файла. В вариантах, имеющих нечетный номер, параметр offset взять равным единице, в четных вариантах offset имеет значение 2. При чтении файла необходимо обеспечить проверку, чтобы не допустить выход указателя за предел файла.

|  |  |
| --- | --- |
| **№** | Задание |
|  | Информация, входящая в файл: Ф.И.О., номер телефона, адрес, стоимость переговоров. При чтении файла вывести сведения о клиенте с указанным номером телефона. |
|  | Воспользовавшись информацией из варианта №1, найти номер телефона клиента с указанной фамилией.  |
|  | Воспользовавшись информацией из варианта №1, вывести сведения о клиенте с указанной фамилией. |
|  | Информация, входящая в файл: Ф.И.О. автора, название книги, издательство, стоимость. При чтении файла вывести информацию о книгах указанного автора. |
|  | Воспользовавшись информацией из варианта №4, получить общую стоимость всех книг. |
|  | Воспользовавшись информацией из варианта №4, вывести сведения о всех книгах указанного издательства. |
|  | Информация, входящая в файл: Ф.И.О., адрес, место работы,зарплата. При чтении файла вывести информацию о всех сотрудниках ПГУ. |
|  | Воспользовавшись информацией из варианта №7, вывести сведения о сотрудниках ПГУ, зарплата которых не превышает 200 тенге.  |
|  | Воспользовавшись информацией из варианта №7, вывести сведения о сотруднике с указанной фамилией. |
|  | Информация, входящая в файл: размер костюма, расход ткани, стоимость, форма. При чтении файла выдать сведения о конкретном размере костюма. |
|  | Воспользовавшись информацией из варианта №10, выдать сведения о стоимости всех костюмов. |
|  | Воспользовавшись информацией из варианта №10, выдать сведения о костюмах стоимостью 300 тенге |
|  | Воспользовавшись информацией из варианта №10, выдать сведения о костюмах, произведенных указанной фирмой. |
|  | Информация, входящая в файл: номер рейса, вес багажа, количество вещей. При чтении файла определить количество пассажиров, вес багажа которых превышает 30 кг . |
|  | Воспользовавшись информацией из варианта №14, определить средний вес багажа. |
|  | Воспользовавшись информацией из варианта №14, определить количество пассажиров, вес багажа которых превосходит средний. |
|  | Воспользовавшись информацией из варианта №14, определить общий вес багажа. |
|  | Воспользовавшись информацией из варианта №14, определить общее количество вещей. |
|  | Информация, входящая в файл: Ф.ИО. студента, группа, оценка по ВТ. При чтении файла подсчитывается средний балл по ВТ. |
|  | Воспользовавшись информацией из варианта №19, определить, какое количество студентов имеет неудовлетворительные оценки.  |
|  | Воспользовавшись информацией из варианта №19, вывести Ф.И.О. студентов, имеющих отличные оценки. |
|  | Воспользовавшись информацией из варианта №19, вывести Ф.И.О. студентов, имеющих оценки выше среднего балла. |
|  | Информация, входящая в файл: номер детали, наименование, количество, стоимость. При чтении файла определить общую стоимость детали. |
|  | Воспользовавшись информацией из варианта №23, определить количество деталей определенного наименования. |
|  | Воспользовавшись информацией из варианта №23, определить количество деталей, стоимость которых не превышает 25 тенге. |

**3. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно- ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьтер, 1993. - 160 с.
5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №9.
“Работа с символьными строками в языке С++»

**Цель работы:** Получение практических навыков в работе с символьными строками языка C++, обеспечение функциональной модульности программы.

**1. Темы для предварительного изучения**

* Указатели в языке C и С++.
* Представление строк.
* Функции и передача параметров.

**2. Постановка задачи**

По индивидуальному заданию создать функцию для обработки символьных строк. За образец брать библиотечные функции обработки строк языка C, но не применять их в своей функции. Предусмотреть обработку ошибок в задании параметров и особые случаи. Разработать два варианта заданной функции – используя традиционную обработку массивов и используя адресную арифметику.

Индивидуальные задания

1. Функция Copies(s,s1,n)

Назначение: копирование строки s в строку s1 n раз

2. Функция Words(s)

Назначение: подсчет слов в строке s

3. Функция Concat(s1,s2)

Назначение: конкатенация строк s1 и s2 (аналогичная библиотечная функция C – strcat)

4. Функция Parse(s,t)

Назначение: разделение строки s на две части: до первого вхождения символа t и после него

5. Функция Center(s1,s2,l)

Назначение: центрирование – размещение строки s1 в середине строки s2 длиной l

6. Функция Delete(s,n,l)

Назначение: удаление из строки s подстроки, начиная с позиции n, длиной l (аналогичная библиотечная Функция есть в Pascal).

7. Функция Left(s,l)

Назначение: выравнивание строки s по левому краю до длины l.

8. Функция Right(s,l)

Назначение: выравнивание строки s по правому краю до длины l.

9. Функция Insert(s,s1,n)

Назначение: вставка в строку s подстроки s1, начиная с позиции n (аналогичная библиотечная функция есть в Pascal).

10. Функция Reverse(s)

Назначение: изменение порядка символов в строке s на противоположный.

11. Функция Pos(s,s1)

Назначение: поиск первого вхождения подстроки s1 в строку s (аналогичная функция C – strstr).

12. Функция LastPos(s,s1)

Назначение: поиск последнего вхождения подстроки s1 в строку s.

13. Функция WordIndex(s,n)

Назначение: определение позиции начала в строке s слова с номером n.

14. Функция WordLength(s,n)

Назначение: определение длины слова с номером n в строке s.

15. Функция SubWord(s,n,l)

Назначение: выделение из строки s l слов, начиная со слова с номером n.

16. Функция WordCmp(s1,s2)

Назначение: сравнение строк (с игнорированием множественных пробелов).

17. Функция StrSpn(s,s1)

Назначение: определение длины той части строки s, которая содержит только символы из строки s1.

18. Функция StrCSpn(s,s1)

Назначение: определение длины той части строки s, которая не содержит символы из строки s1.

19. Функция Overlay(s,s1,n)

Назначение: перекрытие части строки s, начиная с позиции n, строкой s1.

20. Функция Replace(s,s1,s2)

Назначение: замена в строке s комбинации символов s1 на s2.

21. Функция Compress(s,t)

Назначение: замена в строке s множественных вхождений символа t на одно.

22. Функция Trim(s)

Назначение: удаление начальных и конечных пробелов в строке s.

23. Функция StrSet(s,n,l,t)

Назначение: установка l символов строки s, начиная с позиции n, в значение t.

23. Функция Space(s,l)

Назначение: доведение строки s до длины l путем вставки пробелов между словами.

24. Функция Findwords(s,s1)

Назначение: поиск вхождения в строку s заданной фразы (последовательности слов) s1.

25. Функция StrType(s)

Назначение: определение типа строки s (возможные типы – строка букв, десятичное число, 16-ричное число, двоичное число и т.д.).

26. Функция Compul(s1,s2)

Назначение: сравнение строк s1 и та s2 с игнорированием различий в регистрах.

27.Функция Translate(s,s1,s2)

Назначение: перевод в строке s символов, которые входят в алфавит s1, в символы, которые входят в алфавит s2.

28. Функция Word(s)

Назначение: выделение первого слова из строки s.

***Примечание:*** под «словом» везде понимается последовательность символов, которая не содержит пробелов.

**3. Пример решения задачи**

***Индивидуальное задание***

**Функция *substr(s, n, l)***

Назначение: выделение из строки s подстроки, начиная с позиции n, длиной l.

**Описание метода решения**

Символьная строка в языке C представляется в памяти как массив символов, в конце которого находится байт с кодом 0 – признак конца строки. Строку, как и любой другой массив можно обрабатывать либо традиционным методом – как массив, с использованием операции индексации, либо через указатели, с использованием операций адресной арифметики. При работе со строкой как с массивом нужно иметь в виду, что длина строки заранее неизвестна, так что циклы должны быть организованы не со счетчиком, а до появления признака конца строки.

Функция должна реализовывать поставленную задачу – и ничего более. Это означает, что функцию можно будет, например, перенести без изменений в любую другую программу, если спецификации функции удовлетворяют условиям задачи. Это также означает, что при ошибочном задании параметров или при каких-то особых случаях в их значениях функция не должна аварийно завершать программу или выводить какие-то сообщения на экран, но должна возвращать какое-то прогнозируемое значение, по которому та функция, которая вызвала нашу, может сделать вывод об ошибке или об особом случае.

Определим состав параметров функции:

int substr (src, dest, num, len);

где

* **src –** строка, с которой выбираются символы;
* **dest –** строка, в которую записываются символы;
* **num –** номер первого символа в строке src, с которого начинается подстрока (нумерация символов ведется с 0);
* **len –** длина выходной строки.

Возможные возвращаемые значения функции установим: 1 (задание параметров правильное) и 0 (задание не правильное). Эти значения при обращениях к функции можно будет интерпретировать как «истина» или «ложь».

Обозначим через **Lsrc** длину строки **src**. Тогда возможны такие варианты при задании параметров:

* **num+len <= Lsrc –** полностью правильное задание;
* **num+len > Lsrc; num < Lsrc –** правильное задание, но длина выходной строки будет меньше, чем **len**;
* **num >= Lsrc –** неправильное задание, выходная строка будет пустой;
* **num < 0** или **len <= 0 –** неправильное задание, выходная строка будет пустой.

Заметим, что интерпретация конфигурации параметров как пра­вильная/неправильная и выбор реакции на неправильное задание – де­ло исполнителя. Но исполнитель должен строго выполнять принятые правила. Возможен также случай, когда выходная строка выйдет боль­шей длины, чем для нее отведено места в памяти. Однако, поскольку на­шей функции неизвестен размер памяти, отведенный для строки, функ­ция не может распознать и обработать этот случай – так же ведут себя и библиотечные функции языка C.

**4. Описание логической структуры**

Программа состоит из одного программного модуля – файл **LAB1.C**. В состав модуля входят три функции – **main**, **substr\_mas** и **subs\_ptr**. Общих переменных в программе нет. Макроконстантой **N** определена максимальная длина строки – 80.

Функция **main** является главной функцией программы, она предназначена для ввода исходных данных, вызова других функций и вывода результатов. В функции определены переменные:

* **ss** и **dd –** входная и выходная строки соответственно;
* **n –** номер символа, с которого должна начинаться выходная строка;
* l – длина выходной строки.

Функция запрашивает и вводит значение входной строки, номера символа и длины. Далее функция вызывает функцию **substr\_mas**, передавая ей как параметры введенные значения. Если функция **substr\_mas** возвращает 1, выводится на экран входная и выходная строки, если 0 – выводится сообщение об ошибке и входная строка. Потом входная строка делается пустой и то же самое выполняется для функции **substr\_ptr**.

Функция **substr\_mas** выполняет поставленное задание методом массивов. Ее параметры: – **src** и **dest –** входная и выходная строки соответственно, представленные в виде массивов неопределенного размера; **num** и **len**. Внутренние переменные **i** и **j** используются как индексы в массивах.

Функция проверяет значения параметров в соответствии со случаем 4, если условия этого случая обнаружены, в первый элемент массива **dest** записывается признак конца строки и функция возвращает 0.

Если случай 4 не выявлен, функция просматривает num первых символов входной строки. Если при этом будет найден признак конца строки, это – случай 3, при этом в первый элемент массива dest записывается признак конца строки и функция возвращает 0.

Если признак конца в первых num символах не найден, выполня­ется цикл, в котором индекс входного массива начинает меняться от 1, а индекс выходного – от 0. В каждой итерации этого цикла один элемент входного массива пересылается в выходной. Если пересланный элемент является признаком конца строки (случай 2), то функция немедленно за­канчивается, возвращая 1. Если в цикле не встретится конец строки, цикл завершится после len итераций. В этом случае в конец выходной строки записывается признак конца и Функция возвращает 1.

Функция **substr\_ptr** выполняет поставленное задание методом ука­зателей. Ее параметры: – **src** и **dest –** входная и выходная строки соот­ветственно, представленные в виде указателей на начала строк; **num** и **len**.

Функция проверяет значения параметров в соответствии со случа­ем 4, если условия этого случая выявлены, по адресу, который задает **dest**, записывается признак конца строки и функция возвращает 0, эти дейст­вия выполняются одним оператором.

Если случай 4 не обнаружен, функция пропускает num первых символов входной строки. Это сделано циклом while, условием выхода из которого является уменьшение счетчика num до 0 или появление при­знака конца входной строки. Важно четко представлять порядок опера­ций, которые выполняются в этом цикле:

* выбирается счетчик **num**;
* счетчик **num** уменьшается на 1;
* если выбранное значение счетчика было 0 – цикл завершается;
* если выбранное значение было не 0 – выбирается символ, на который указывает указатель src;
* указатель src увеличивается на 1;
* если выбранное значение символа было 0, то есть, признак конца строки, цикл завершается, иначе – повторяется.

После выхода из цикла проверяется значение счетчика **num**: если оно не 0, это означает, что выход из цикла произошел по признаку конца строки (случай 3), по адресу, который задает **dest**, записывается признак конца строки и функция возвращает 0.

Если признак конца не найден, выполняется цикл, подобный первому циклу **while**, но по счетчику **len**. В каждой итерации этого цикла символ, на который показывает **src** переписывается по адресу, задаваемому **dest**, после чего оба указателя увеличиваются на 1. Цикл закончится, когда будет переписано **len** символов или встретится признак конца строки. В любом варианте завершения цикла по текущему адресу, который содержится в указателе **dest**, записывается признак конца строки и функция завершается, возвращая 1.

**5. Задание**

По индивидуальному заданию, указанному в постановке задачи, создать функцию для обработки символьных строк. За образец брать библиотечные функции обработки строк языка C++, но не применять их в своей функции. Осуществить тестирование программы на языке С++.

**Данные для тестирования**

Тестирование должно обеспечить проверку работоспособности функций для всех вариантов входных данных. Входные данные, на которых должно проводиться тестирование, сведены в таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **src** | **num** | **len** | **dest** |
| 1 | 012345 | 2 | 2 | 23 |
|  | 012345 | 0 | 1 | 0 |
|  | 012345 | 0 | 6 | 012345 |
| 2 | 012345 | 5 | 3 | 5 |
|  | 012345 | 2 | 6 | 2345 |
|  | 012345 | 0 | 7 | 012345 |
| 3 | 012345 | 8 | 2 | пусто |
| 4 | 012345 | -1 | 2 | пусто |
|  | 012345 | 5 | 0 | пусто |
|  | 012345 | 5 | 1 | пусто |

**Текст программы**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <stdio.h>**

**#define N 80**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Функция выделения подстроки (массивы) \*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**int substr\_mas(char src[N],char dest[N],int num,int len)**

**{ int i, j;**

**/\* проверка случая 4 \*/**

**if ( (num<0)||(len<=0) ) {**

**dest[0]=0; return 0; }**

**/\* выход на num-ый символ \*/**

**for (i=0; i<=num; i++)**

**/\* проверка случая 3 \*/**

**if ( src[i]=='\0') {**

**dest[0]=0; return 0; }**

**/\* перезапись символов \*/**

**for (i--, j=0; j<len; j++, i++)**

**{ dest[j]=src[i];**

**/\* проверка случая 2 \*/**

**if ( dest[j]=='\0') return 1; }**

**/\* запись признака конца в выходную строку \*/**

**dest[j]='\0';**

**return 1;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Функция выделение подстроки \*/**

**/\* (адресная арифметика) \*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**int substr\_ptr(char \*src, char \*dest, int num, int len)**

**{**

**/\* проверка случая 4 \*/**

**if ( (num<0)||(len<=0) )**

**return dest[0]=0;**

**/\* выход на num-ый символ или на конец строки \*/**

**while ( num-- && \*src++ );**

**/\* проверка случая 3 \*/**

**if ( !num ) return dest[0]=0;**

**/\* перезапись символов \*/**

**while ( len-- && \*src )**

**\*dest++=\*src++;**

**/\* запись признака конца в выходную строку \*/**

**\*dest=0; return 1;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**main()**

**{**

**char ss[N], dd[N];**

**int n, l;**

**clrscr();**

**printf("Вводите строку:\n");**

**gets(ss);**

**printf("начало=");**

**scanf("%d",&n);**

**printf("длина=");**

**scanf("%d",&l);**

**printf("Массивы:\n");**

**if (substr\_mas(ss,dd,n,l))**

**printf(">>%s<<\n>>%s<<\n",ss,dd);**

**else**

**printf("Ошибка! >>%s<<\n",dd);**

**dd[0]='\0';**

**printf("Адресная арифметика:\n");**

**if (substr\_ptr(ss,dd,n,l))**

**printf(">>%s<<\n>>%s<<\n",ss,dd);**

**else**

**printf("Ошибка! >>%s<<\n",dd);**

**getch(); }**

**6. Литература**

1. Подбельский В.В. Язык C++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.

2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. - 352 стр.

3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно- ориентированного программирования. - М.: Свет, 1993. - 236 с.

4. Романов В.Ю. Программирование на языке C++. Практический подход. - М.: Компьтер, 1993. - 160 с.

5. Уинер Р. Язык Турбо Cu . - М.: Мир, 1991. - 384 с.

6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.

7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.

Лабораторная работа №10.
Представление в памяти массивов и матриц

**Цель работы:** Получение практических навыков в использовании указателей и динамических объектов в языке C++, создание модульных программ и обеспечение инкапсуляции.

1. **Постановка задачи**

Для разряженной матрицы целых чисел в соответствии с индивидуальным заданием создать модуль доступа к ней, в котором обеспечить экономию памяти при размещении данных.

**2. Индивидуальные задания**

1. Все нулевые элементы размещены в левой части матрицы
2. Все нулевые элементы размещены в правой части матрицы
3. Все нулевые элементы размещены выше главной диагонали
4. Все нулевые элементы размещены в верхней части матрицы
5. Все нулевые элементы размещены в нижней части матрицы
6. Все элементы нечетных строк – нулевые
7. Все элементы четных строк – нулевые
8. Все элементы нечетных столбцов – нулевые
9. Все элементы четных столбцов – нулевые
10. Все нулевые элементы размещены в шахматном порядке, начиная с 1-го элемента 1-й строки
11. Все нулевые элементы размещены в шахматном порядке, начиная со 2-го элемента 1-й строки
12. Все нулевые элементы размещены на местах с четными индексами строк и столбцов
13. Все нулевые элементы размещены на местах с нечетными индексами строк и столбцов
14. Все нулевые элементы размещены выше главной диагонали на нечетных строках и ниже главной диагонали – на четных
15. Все нулевые элементы размещены ниже главной диагонали на нечетных строках и выше главной диагонали – на четных
16. Все нулевые элементы размещены на главной диагонали, в первых 3 строках выше диагонали и в последних 3 строках ниже диагонали
17. Все нулевые элементы размещены на главной диагонали и в верхней половине участка выше диагонали
18. Все нулевые элементы размещены на главной диагонали и в нижней половине участка ниже диагонали
19. Все нулевые элементы размещены в верхней и нижней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
20. Все нулевые элементы размещены в левой и правой четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
21. Все нулевые элементы размещены в левой и верхней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
22. Все нулевые элементы размещены на строках, индексы которых кратны 3
23. Все нулевые элементы размещены на столбцах, индексы которых кратны 3
24. Все нулевые элементы размещены на строках, индексы которых кратны 4
25. Все нулевые элементы размещены на столбцах, индексы которых кратны 4
26. Все нулевые элементы размещены попарно в шахматном порядке (сначала 2 нулевых)
27. Матрица поделена диагоналями на 4 треугольники, элементы верхнего и нижнего треугольников нулевые
28. Матрица поделена диагоналями на 4 треугольники, элементы левого и правого треугольников нулевые
29. Матрица поделена диагоналями на 4 треугольника, элементы правого и нижнего треугольников нулевые
30. Все нулевые элементы размещены квадратами 2х2 в шахматном порядке

Исполнителю самому надлежит выбрать, будут ли начинаться индексы в матрице с 0 или с 1.

**Пример решения задачи**

Индивидуальное задание:

* матрица содержит нули ниже главной диагонали;
* индексация начинается с 0.

**3. Описание методов решения**

**Представление в памяти**

Экономное использование памяти предусматривает, что для тех элементов матрицы, в которых наверняка содержатся нули, память выделяться не будет. Поскольку при этом нарушается двумерная структура матрицы, она может быть представлена в памяти как одномерный массив, но при обращении к элементам матрицы пользователь имеет возможность обращаться к элементу по двум индексам.

**Модульная структура программы**

Программное изделие должно быть отдельным модулем, файл LAB2.C, в котором должны размещаться как данные (матрица и вспомогательная информация), так и функции, которые обеспечивают доступ. Внешний доступ к программам и данным модуля возможен только через вызов функций чтения и записи элементов матрицы. Доступные извне элементы программного модуля должны быть описаны в отдельном файле LAB2.H, который может включаться в программу пользователя оператором препроцессора:

#include "lab2.h"

Пользователю должен поставляться результат компиляции – файл LAB2.OBJ и файл LAB2.H.

Преобразование 2-компонентного адреса элемента матрицы, которую задает пользователь, в 1-компонентную должно выполняться отдельной функцией (так называемой, функцией линеаризации), вызов которой возможен только из функций модуля. Возможны три метода преобразования адреса:

* при создании матрицы для нее создается также и дескриптор D[N] – отдельный массив, каждый элемент которого соответствует одной строке матрицы; дескриптор заполняется значениями, подобранными так, чтобы: **n = D[x] + y**, где **x**, **y –** координаты пользователя (строка, столбец), **n –** линейная координата;
* линейная координата подсчитывается методом итерации как сумма полезных длин всех строк, предшествующих строке **x**, и к ней прибавляется смещение **y**-го полезного элемента относительно начала строки;
* для преобразования подбирается единое арифметическое выражение, которой реализует функцию: **n = f(x,y)**.

Первый вариант обеспечивает быстрейший доступ к элементу матрицы, ибо требует наименьших расчетов при каждом доступе, но плата за это – дополнительные затраты памяти на дескриптор. Второй вариант – наихудший по всем показателям, ибо каждый доступ требует выполнения оператора цикла, а это и медленно, и занимает память. Третий вариант может быть компромиссом, он не требует дополнительной памяти и работает быстрее, чем второй. Но выражение для линеаризации тут будет сложнее, чем первом варианте, следовательно, и вычисляться будет медленнее.

В программном примере, который мы приводим ниже, полностью реализован именно третий вариант, но далее мы показываем и сущест­венные фрагменты программного кода для реализации и двух других.

**4. Описание логической структуры**

Общие переменные

В файле LAB10.C описаны такие статические переменные:

* **int NN –** размерность матрицы;
* **int SIZE –** количество ненулевых элементов в матрице;
* **int \*m\_addr –** адрес сжатой матрицы в памяти, начальное значение этой переменной – **NULL –** признак того, что память не выделена;
* **int L2\_RESULT –** общий флаг ошибки, если после выполнения любой функции он равен -1, то произошла ошибка.

Переменные **SIZE** и **m\_addr** описаны вне функций с квалификатором **static**, это означает, что вони доступны для всех функций в этом модуле, но недоступны для внешних модулей. Переменная **L2\_RESULT** также описана вне всех функций, не без явного квалификатора. Эта переменная доступна не только для этого модуля, но и для всех внешних модулей, если она в них буде описана с квалификатором extern. Такое описание имеется в файле **LAB10.H**.

Функция ***creat\_matr***

Функция **creat\_matr** предназначена для выделения в динамической памяти места для размещения сжатой матрицы. Прототип функции:

int creat\_matr ( int N );

где **N –** размерность матрицы.

Функция сохраняет значение параметра в собственной статической переменной и подсчитывает необходимый размер памяти для размещения ненулевых элементов матрицы. Для выделения памяти используется библиотечная функция C **malloc**. Функция возвращает -1, если при выделении произошла ошибка, или 0, если выделение прошло нормально. При этом переменной **L2\_RESULT** также присваивается значение 0 или -1.

Функция ***close\_matr***

Функция **close\_matr** предназначена для освобождения памяти при завершении работы с матрицей,

Прототип функции:

int close\_matr ( void );

Функция возвращает 0 при успешном освобождении, -1 – при попытке освободить невыделенную память.

Если адрес матрицы в памяти имеет значения **NULL**, это признак того, что память не выделялась, тогда функция возвращает -1, иначе – освобождает память при помощи библиотечной функции free и записывает адрес матрицы – **NULL**. Соответственно функция также устанавливает глобальный признак ошибки – **L2\_RESULT**.

Функция ***read\_matr***

Функция **read\_matr** предназначена для чтения элемента матрицы.

Прототип функции:

int read\_matr(int x, int y);

где **x** и **y –** координаты (строка и столбец). Функция возвращает значение соответствующего элемента матрицы. Если после выполнения функции значение переменной **L2\_RESULT** -1, то это указывает на ошибку при обращении.

Проверка корректности задания координат выполняется обращением к функции **ch\_coord**, если эта последняя возвращает ненулевое значение, выполнение **read\_matr** на этом и заканчивается. Если же координаты заданы верно, то проверяется попадание заданного элемента в нулевой или ненулевой участок. Элемент находится в нулевом участке, если для него номер строки больше, чем номер столбца. Если элемент в нулевом участке, функция просто возвращает 0, иначе – вызывает функцию линеаризации lin и использует значение, которое возвращает **lin**, как индекс в массиве **m\_addr**, по которому и выбирает то значения, которое возвращается.

Функция ***write\_matr***

Функция **write\_matr** предназначена для записи элемента в матрицу.

Прототип функции:

int write\_matr(int x, int y, int value);

где **x** и **y –** координаты (строка и столбец), **value –** то значение, которое нужно записать. Функция возвращает значение параметра **value**, или 0 – если была попытка записи в нулевой участок. Если после выполнения функции значение переменной **L2\_RESULT** -1, то это указывает на ошибку при обращении.

Выполнение функции подобно функции **read\_matr** с тем отличием, что, если координаты указывают на ненулевой участок, то функция записывает **value** в массив **m\_addr**.

**Функция *ch\_coord***

Функция **ch\_coord** предназначена для проверки корректности за­дания координат. Эта функция описана как static и поэтому может вызы­ваться только из этого же модуля. Прототип функции:

static char ch\_coord(int x, int y);

где **x** и **y –** координаты (строка и столбец). Функция возвращает 0, если координаты верные, -1 – если неверные. Соответственно, функция также устанавливает значение глобальной переменной **L2\_RESULT**.

Выполнение функции собственно состоит из проверки трех условий:

* адрес матрицы не должен быть **NULL**, то есть, матрица должна уже находиться в памяти;
* ни одна из координат не может быть меньше 0;
* ни одна из координат не может быть больше **NN**.

Если хотя бы одно из этих условий не выполняется, функция устанавливает признак ошибки.

**Функция *lin***

Функция **lin** предназначена для преобразования двумерных координат в индекс в одномерном массиве. Эта функция описана как **static** и поэтому может вызываться только из этого же модуля. Прототип функции:

static int lin(int x, int y);

где **x** и **y –** координаты (строка и столбец). Функция возвращает координату в массиве **m\_addr**.

Выражение, значение которого вычисляет и возвращает функция, подобрано вот из каких соображений. Пусть мы имеет такую матрицу, как показано ниже, и нам нужно найти линейную координату элемента, обозначенного буквой **A** с координатами (**x,y**):

x x x x x x

0 x x x x x

0 0 x x A x

0 0 0 x x x

0 0 0 0 x x

0 0 0 0 0 x

Координату элемента можно определить как:

n = SIZE – sizeX +offY,

где **SIZE –** общее количество элементов в матрице,

SIZE = NN \* (NN – 1) / 2 + NN;

**sizeX –** количество ненулевых элементов, которые содержатся в строке **x** и ниже,

sizeX = (NN – x) \* (NN – x – 1) / 2 + (NN – x);

**offY –** смещение нужного элемента от начала строки **x**,

offY = y – x

**5. Программа пользователя**

Для проверки функционирования нашего модуля создается программный модуль, который имитирует программу пользователя. Этот модуль обращается к функции **creat\_matr** для создания матрицы нужного размера, заполняет ненулевую ее часть последовательно увеличивающимися числами, используя для этого функцию **write\_matr**, и выводит матрицу на экран, используя для выборки ее элементов функцию **read\_matr**. Далее в диалоговом режиме программа вводит запрос на свои действия и читает/пишет элементы матрицы с заданными координатами, обращаясь к функциям **read\_matr/ write\_matr**. Если пользователь захотел закончить работу, программа вызывает функцию **close\_matr**.

Тексты программных модулей

**/\*\*\*\*\*\*\* Файл LAB10.H \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Описание функций и внешних переменных файла LAB2.C \*/**

**extern int L2\_RESULT; /\* Глобальна переменная – флаг ошибки \*/**

**/\*\*\*\*\* Выделение памяти под матрицу \*/**

**int creat\_matr ( int N );**

**/\*\*\*\*\* Чтение элемента матрицы по заданным координатам \*/**

**int read\_matr ( int x, int y );**

**/\*\*\*\*\* Запись элемент в матрицу по заданным координатам \*/**

**int write\_matr ( int x, int y, int value );**

**/\*\*\*\*\* Уничтожение матрицы \*/**

**int close\_matr ( void );**

**/\*\*\*\*\*\*\* Конец файла LAB10.H \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\*\*\*\*\*\*\*\*\*\* Файл LAB10.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* В этом файле определены функции и переменные для обработки**

**матрицы, заполненной нулями ниже главной диагонали \*/**

**#include <alloc.h>**

**static int NN; /\* Размерность матрицы \*/**

**static int SIZE; /\* Размер памяти \*/**

**static int \*m\_addr=NULL; /\* Адрес сжатой**

**матрицы \*/**

**static int lin(int, int); /\* Описание функции линеаризации \*/**

**static char ch\_coord(int, int); /\* Описание функции проверки \*/**

**int L2\_RESULT; /\* Внешняя переменная, флаг ошибки \*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Выделение памяти под сжатую матрицу \*/**

**int creat\_matr ( int N ) {**

**/\* N – размер матрицы \*/**

**NN=N;**

**SIZE=N\*(N-1)/2+N;**

**if ((m\_addr=(int \*)malloc(SIZE\*sizeof(int))) == NULL ) return L2\_RESULT=-1;**

**else return L2\_RESULT=0; /\* Возвращает 0, если выделение прошло успешно, иначе -1 \*/**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Уничтожение матрицы (освобождение памяти) \*/**

**int close\_matr(void) {**

**if ( m\_addr!=NULL ) { free(m\_addr); m\_addr=NULL; return L2\_RESULT=0; }**

**else return L2\_RESULT=-1; /\* Возвращает 0, если освобождение прошло успешно, иначе – -1 \*/ }**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Чтение элемента матрицы по заданным координатам \*/**

**int read\_matr(int x, int y) {**

**/\* x, y -координати (строка, столбец) \*/**

**if ( ch\_coord(x,y) ) return 0;**

**/\* Если координаты попадают в нулевой участок – возвращается 0, иначе – применяется функция линеаризации \*/**

**return (x > y) ? 0 : m\_addr[lin(x,y)];**

**/\* Проверка успешности чтения – по переменной L2\_RESULT: 0 – без ошибок, -1 – была ошибка \*/ }**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Запись элемента матрицы по заданным координатам \*/**

**int write\_matr(int x, int y, int value) {**

**/\* x, y -координати, value – записываемое значение \*/**

**if ( chcoord(x,y) ) return;**

**/\* Если координаты попадают в нулевой участок – записи нет, иначе – применяется функция линеаризации \*/**

**if ( x>y ) return 0;**

**else return m\_addr[lin(x,y)]=value;**

**/\* Проверка успешности записи – по L2\_RESULT \*/ }**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
/\* Преобразование 2-мерных координат в линейную \*/
/\* (вариант 3)\*/**

**static int lin(int x, int y) {**

**int n;**

**n=NN-x;**

**return SIZE-n\*(n-1)/2-n+y-x; }**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Проверка корректности обращения \*/**

**static char ch\_coord(int x, int y) { if ( ( m\_addr==NULL ) ||**

**( x>SIZE ) || ( y>SIZE ) || ( x<0 ) || ( y<0 ) )**

**/\* Если матрица не размещена в памяти, или заданные координаты выходят за пределы матрицы \*/**

**return L2\_RESULT=-1; return L2\_RESULT=0; }**

**/\*\*\*\*\*\*\*Конец файла LAB10.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\*\*\*\*\*\*\*\* Файл MAIN10.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* "Программа пользователя" \*/**

**#include "lab10.h"**

**main()**

**{**

**int R; /\* размерность \*/**

**int i, j; /\* номера строки и столбца \*/**

**int m; /\* значения элемента \*/ int op; /\* операция \*/ clrscr();**

**printf('Введите размерность матрицы >');**

**scanf("%d",R); /\* создание матрицы \*/**

**if ( creat\_matr (R) ) {**

**printf("Ошибка создания матрицы\n");**

**exit(0); } /\* заполнение матрицы \*/**

**for ( m=j=0; j<R; j++)**

**for ( i=о; i<R; i++)**

**write\_matr(i,j,++m); while(1)**

**{ /\* вывод матрицы на экран \*/**

**clrscr();**

**for (j=0; j<R; j++)**

**{ for (i=0; i<R; i++)**

**printf("%3d ",read\_matr(i,j));**

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

**printf("0 – выход\n1 – чтение\n2 – запись\n>")**

**scanf("%d",&op);**

**switch(op) {**

**case 0:**

**if (close\_matr())**

**printf("Ошибка при уничтожении\n");**

**else**

**printf("Матрица уничтожена\n");**

**exit(0);**

**case 1:**

**case 2: printf("Введите номер строки >");**

**scanf("%d",&j);**

**printf("Введите номер столбца >"); scanf("%d",&i);**

**if (op==2) {**

**printf("Введите значение элемента >"); scanf("%d",&m); write\_matr(j,i,m);**

**if (L2\_RESULT<0) pritnf("Ошибка записи\n"); }**

**else {**

**m=read\_matr(j,i);**

**if (L2\_RESULT<0) pritnf("Ошибка считывания\n");**

**else printf("Считано: %d\n",m); }**

**printf("Нажмите клавишу\n");**

**getch();**

**break;}**

**/\*\*\*\*\*Конец файла MAIN10.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**6. Варианты**

Вариант 1 требует:

* добавления к общим статическим переменным еще переменной:

static int \*D;

/\* адрес дескриптора \*/

* добавления такого блока в функцию creat\_matr:

{

int i, s;

D=(int \*)malloc(N\*sizeof(int)); for (D[0]=0,s=NN-1,i=1; i<NN; i++) D[i]=D[i-1]+s--;

}

* изменения функции lin на:

static int lin(int x, int y) { return D[x]+y; }

Вариант 2 требует:

* изменения функции lin на:

static int lin(int x, int y) {

int s;

for (s=j=0; j<x; j++)

s+=NN-j; return (s+y-x);

}

**ЛИТЕРАТУРА**

1. Подбельский В.В. Язык C++: Учебное пособие. – М.: Финансы и статистика,1995, – 560 с.

2. Страуструп Б. Язык программирования С++. - М.: Радио и связь, 1991. – 352 стр.

3. Собоцинский В.В. Практический курс Turbo C++. Основы объектно-ориентированного программирования. – М.: Свет, 1993. - 236 с.

4. Романов В.Ю. Программирование на языке C++. Практический подход. – М.:Компьютер,1993. – 160с.

5. Уинер Р. Язык Турбо Cu . – М.: Мир, 1991. – 384 с.

6. Юлин В.А., Булатова И.Р. Приглашение к Cu. – Мн.: Высш. Шк., 1990, – 224 с.

7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. – Мн.: Высш. Шк., 1991. – 156 с.

8. Березин Б. И., Березин С. Б. Начальный курс С и С++. М.: Диалог-МИФИ, 2000. 288 с.

9. Касаткин А. И., Вальвачев А. Н. Профессиональное программирование на языке Си: от Turbo C к Borland C++. М.: Высш. шк., 1992. 240 с.

10. Культин Н. Б., С/С++ в задачах и примерах. СПб.: БХВ-Петербург, 2002. 288 с.

11. Нейбауэр А. Моя первая программа на С/С++. СПб.: «Питер», 1995. 398 с.

**СОДЕРЖАНИЕ**

Введение.……………………………………………………………………………..3

1. Лабораторная работа №1 “Операторы ввода и вывода в языке

программирования Си++”.……………………………………………………….6

2. Лабораторная работа №2 “Условные и безусловные операторы в Си++” …..16

3. Лабораторная работа №3 “Операторы цикла в языке

программирования Си++”.……………………………………………………...26

4. Лабораторная работа №4 “Массивы и указатели в языке

программирования Си++”.………………………………………………...……34

5. Лабораторная работа №5 "Работа с функциями в языке

программирования Си++" ………………………………………………………42

6. Лабораторная работа №6 “Работа со структурами в языке

программирования Си++”.…………………………………………………...…49

7. Лабораторная работа №7 “Препроцессорные средства в языке

программирования Си++”.………………………………………………..…….56

8. Лабораторная работа №8 “Работа с файлами в языке

программирования Си++”………………………………………….…………...61

9. Лабораторная работа №9. “Работа с символьными строками”………………71

10. Лабораторная работа №10. “Представление в памяти массивов и матриц”.79

Литература…………………………………………………………………………..89