Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

Ульяновский государственный технический университет

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Методические указания к лабораторным работам
(первый семестр)

Составитель С.М. Наместников

Ульяновск
2008

УДК 621.394.343 (076)
ББК 32.88 я7
П33

Рецензент доцент кафедры «Системы автоматизированного проектирования» Ульяновского государственного технического университета, канд. техн. наук, доцент Сухов С. А.

Одобрено секцией методических пособий научно-методического совета университета

Программирование на языке С: методические указания к лабораторным П33 работам /сост. С. М. Наместников. – Ульяновск : УлГТУ, 2008. – 27 с.

Указания по курсу «Информатика» для студентов направления 210406 специализации «Сети связи и системы коммутации» специальности 21040665 «Сети связи и системы коммутации» разработаны в соответствии с программой курса «Информатика» и предназначен для студентов специальности «Сети связи и системы коммутации», но может использоваться и студентами других специальностей. Лабораторные работы посвящены основам программирования на языке С.

Сборник подготовлен на кафедре «Телекоммуникации».

УДК 621.394.343 (076)
ББК 32.88 я7

СОДЕРЖАНИЕ

Лабораторная работа №1

ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Цель работы: научиться создавать консольные проекты в интегрированной среде программирования Borland C++ Builder и программировать простые арифметические операции.

Создание консольных приложений в MS Visual Studio 2008

Для выполнения лабораторных работ по курсу «Информатика» рассмотрим порядок создания заготовки программы на языке Visual Studio 2008. После установки данного языка программирования на рабочем столе (или в меню пуск) появится иконка для запуска с названием «Microsoft Visual Studio 2008», выбирая которую на экране появится главное окно программы (рис. 1.1).

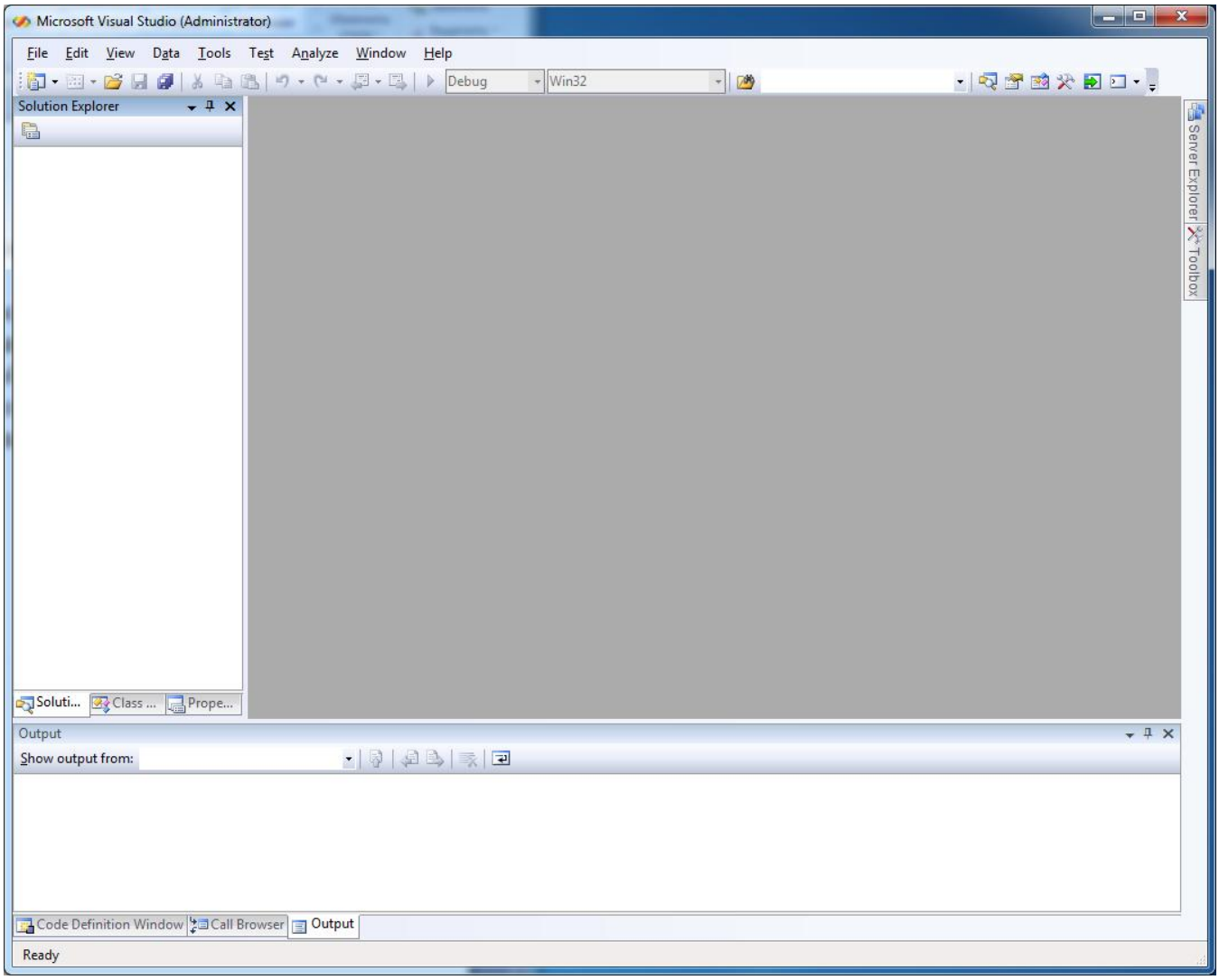

Рис. 1.1. Главное окно программы Visual Studio 2008

В самом верху окна располагается меню, с помощью которого можно осуществлять управление процессом создания, компиляции и отладки программ.

Создание программ осуществляется путем выбора в меню пункта

File->New->Project

после чего на экране появится диалоговое окно выбора типа проекта (рис. 1.2). В данном окне в типах проектов (Projects types) следует выбрать пункт Win32, а в шаблонах (Templates) Win32 Console Application. Все эти пункты показаны на рис. 1.2. После этого внизу окна в поле Name (имя) следует ввести имя проекта (английскими буквами), например, lab1, а в поле Location (расположение) указать папку, в которой будет располагаться проект, например, D:\temp. После заполнения всех указанных полей и нажатия на кнопку «OK» на экране появится окно настройки выбранного консольного проекта (рис. 1.3).

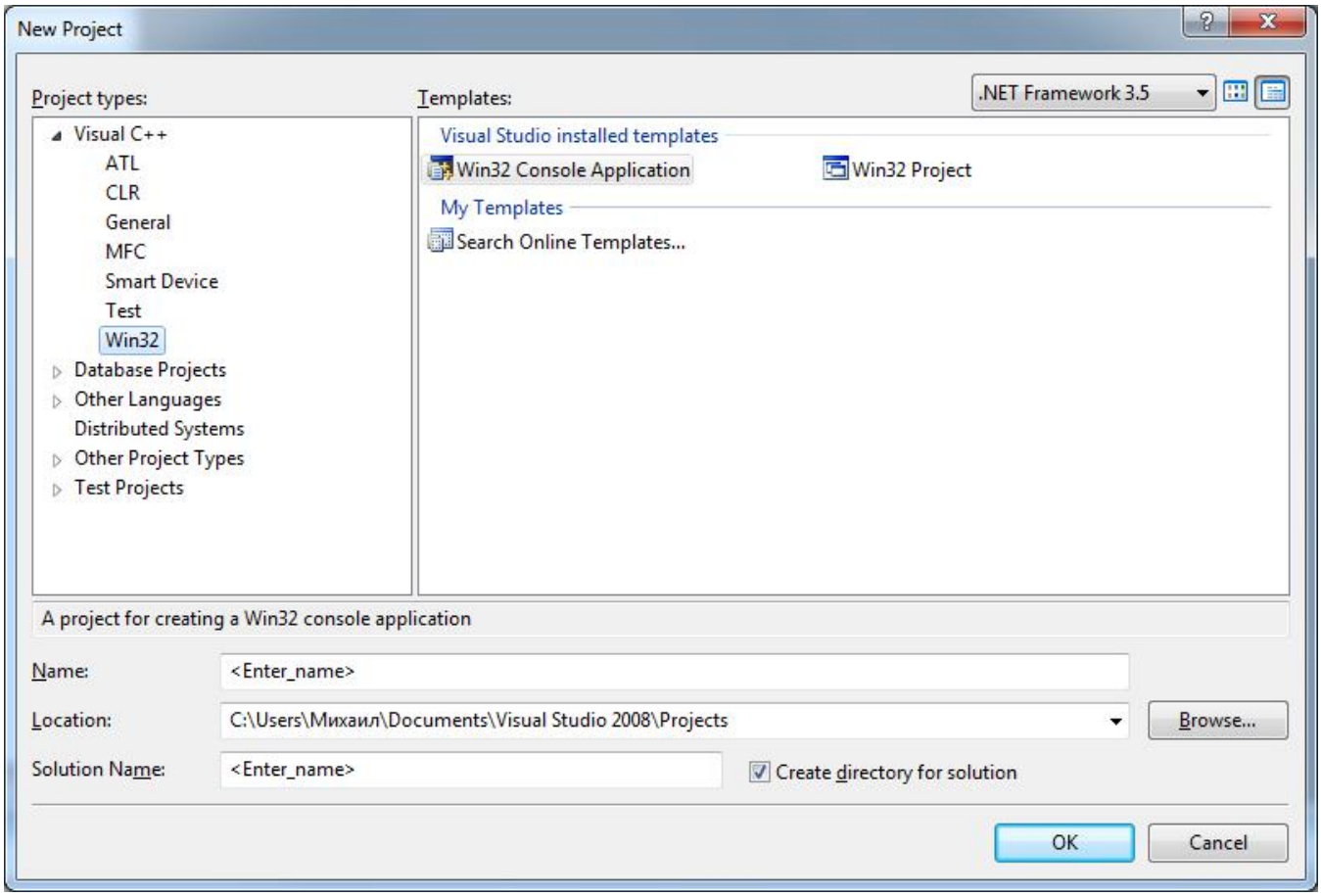

Рис. 1.2. Диалоговое окно выбора типа проекта

Здесь достаточно нажать на кнопку «Finish», после чего будет создан проект, а в главном окне программы появится список файлов консольного проекта (рис. 1.4).

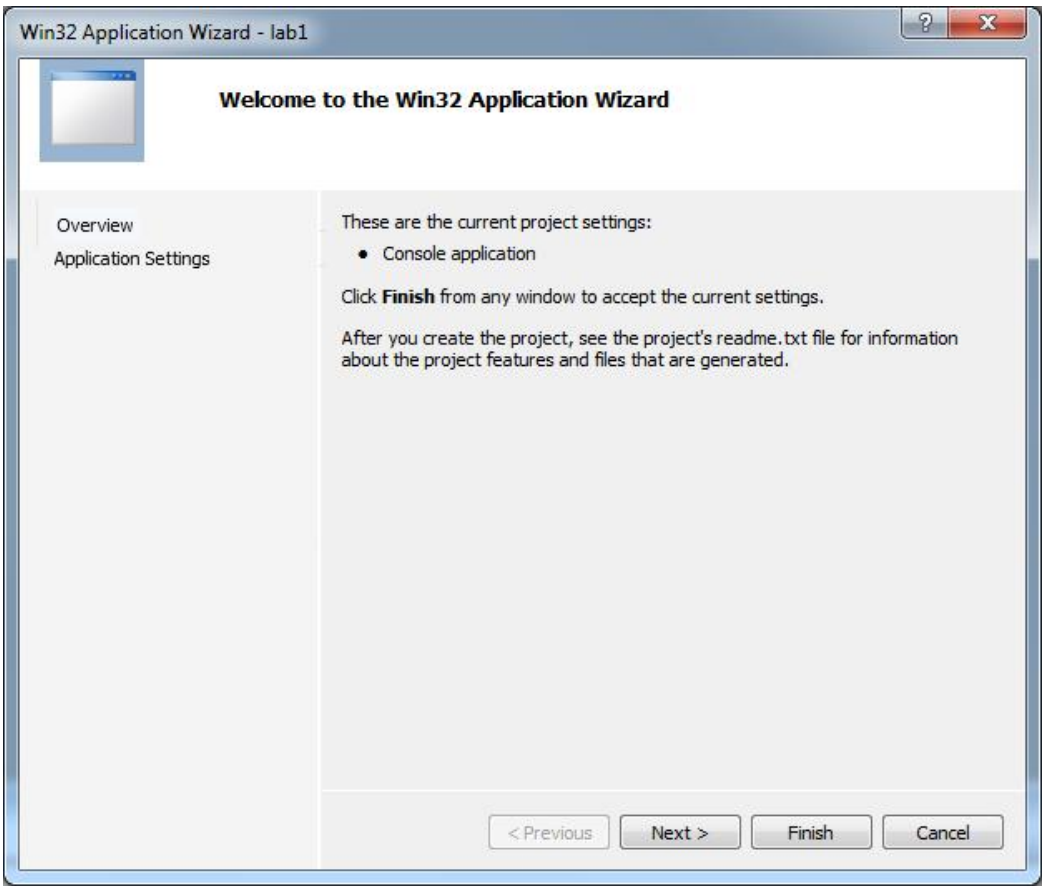

Рис. 1.3. Окно настройки консольного проекта

Слева в окне отображен список файлов проекта, из которых интерес представляет лишь файл lab1.cpp, т.к. в нем пишется непосредственно программа на языке С. Справа открыт файл lab1.cpp, в котором используется стандартный заголовочный файл проекта stdafx.h и главная функция программы _tmain() (аналог функции main()). Вся логика программы должна заключаться внутри функции _tmain() как это описано в лекциях.

Для того чтобы сохранить проект на внешнем носителе, например, Flash, необходимо открыть ранее указанную папку при создании проекта и в ней скопировать каталог с названием проекта, в данном случае – это каталог с именем lab1.

Чтобы загрузить ранее созданный проект в среду Visual Studio 2008 достаточно открыть пункт меню

File->Open->Project/Solution

и в диалоговом окне указать файл проекта, находящегося в папке проекта, в данном случае – это папка lab1.

Для запуска и компиляции программы, написанной в данной среде, используется команда меню

Debug->Start Without Debugging

или комбинация клавиш Ctrl+F5.

После запуска программы на экране появится окно, показанное на рис 1.5.

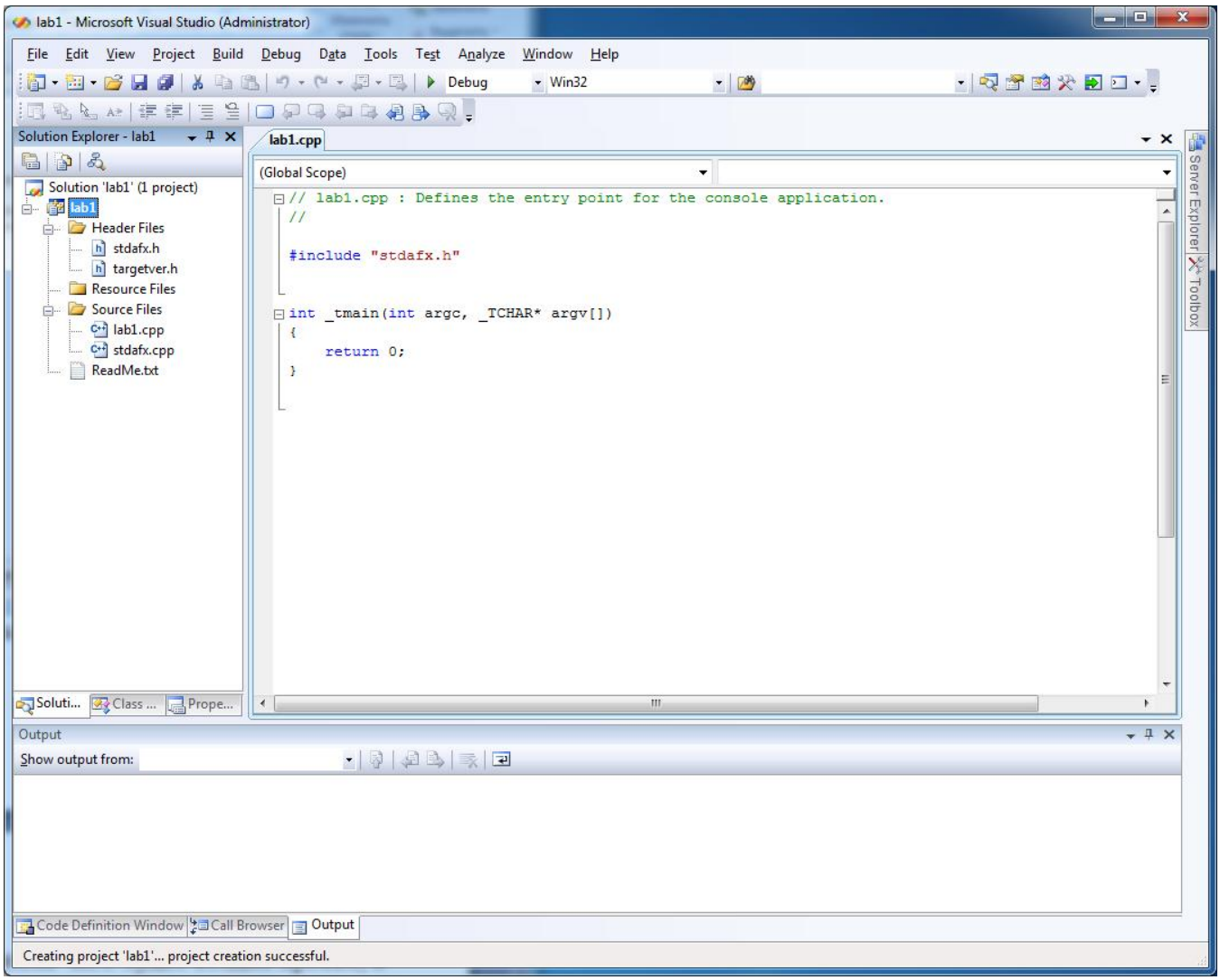

Рис. 1.4. Главное окно программы с открытым консольным проектом

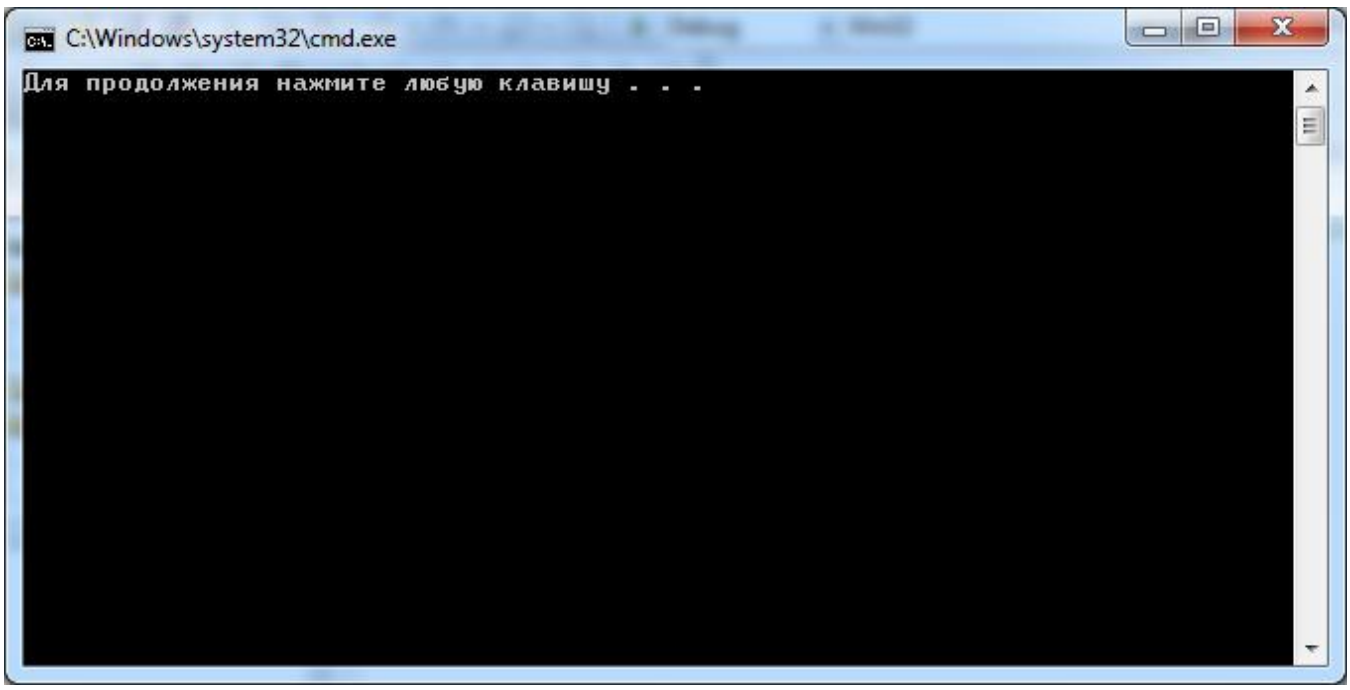

Рис. 1.5. Простое консольное приложение

Таким образом можно создавать проекты для каждой лабораторной работы, сохранять, загружать и выполнять их.

Представление данных в языке С

Для того чтобы иметь возможность работать с тем или иным типом данных необходимо задать переменную соответствующего типа. Это осуществляется с использованием следующего синтаксиса:

```
<тип переменной> <имя_переменной>;
```

например, строка

```
int arg;
```

объявлет целочисленную переменную с именем arg.

Таблица 1.1. Основные базовые типы данных

Тип	Описание
int	Целочисленный (обычно 32 бита)
short	Целочисленный (обычно 16 бит)
char	Символьный тип 8 бит
float	Вещественный тип 32 бита
double	Вещественный тип 64 бита

Отметим, что при выборе имени переменной целесообразно использовать осмысленные имена. При определении имени можно использовать как верхний, так и нижний регистры букв латинского алфавита. Причем первым символом обязательно должна быть буква или символ подчеркивания ‘_’. Вот несколько примеров:

Правильные имена	Неправильные имена
arg	&arg
cnt	$cnt
bottom_x	bottom-x
Arg	2Arg
don_t	don’t

В приведенных примерах переменные arg и Arg считаются разными, т.к. язык С при объявлении переменных различает большой и малый регистры.

В отличие от многих языков программирования высокого уровня, в языке С переменные могут объявляться в любом месте текста программы.

Арифметические операции

В языке С довольно просто реализуются элементарные математические операции: сложения, вычитания, умножения и деления. Допустим, что в программе заданы две переменные

```
int a, b;
```

с начальными значениями

```
a=4;
b=8;
```

тогда операции сложения, вычитания, умножения и деления будут выглядеть следующим образом:

```
int c;
c = a+b;        //сложение двух переменных
c = a-b;        //вычитание
c = a*b;        //умножение
c = a/b;        //деление
```

Представленные операции можно выполнять не только с переменными, но и с конкретными числами, например

```
c = 10+5;
c = 8*4;
float d;
d = 7/2;
```

Результатом первых двух арифметических операций будут числа 15 и 32 соответственно, но при выполнении операции деления в переменную d будет записано число 3, а не 3,5. Это связано с тем, что число 7 в языке С++ будет интерпретироваться как целочисленная величина, которая не может содержать дробной части. Поэтому полученная дробная часть 0,5 будет отброшена. Для реализации корректного деления одного числа на другое следует использовать такую запись:

```
d = 7.0/2;
```

или

```
d = (float )7/2;
```

В первом случае вещественное число делится на два и результат (вещественный) присваивается вещественной переменной d. Во втором варианте выполняется приведение типов: целое число 7 приводится к

вещественному типу float, а затем делится на 2. Второй вариант удобен, когда выполняется деление одной целочисленной переменной на другую:

```
int a,b;
a = 7; b = 2;
d = a/b;
```

В результате значение d будет равно 3, но если записать

```
d = (float )a/b;
```

то получим значение 3,5. Здесь следует также отметить, что если переменная d является целочисленной, то результат деления всегда будет записан с отброшенной дробной частью.

В заключение рассмотрения работы с арифметическими операциями отметим, что приоритет операций умножения и деления выше приоритета операций сложения и вычитания. Это означает, что сначала выполняются операции умножения и деления и только затем операции сложения и вычитания. Следующий пример демонстрирует приоритет арифметических операций:

```
double n=2, SCALE = 1.2;
double arg = 25.0 + 60.0*n/SCALE;
```

В приведенном примере сначала будет выполнена операция умножения, затем деления и, наконец, сложения. То есть порядок вычисления соответствует математическим правилам. Для того чтобы изменить порядок вычисления (поменять приоритеты) используются круглые скобки как показано ниже

```
double arg = (25.0 + 60.0)*n/SCALE;
```

Здесь сначала выполняется операция сложения и только затем операции умножения и деления.

Для простоты программирования в языке С реализованы компактные операторы инкремента и декремента, т.е. увеличения и уменьшения значения переменной на 1 соответственно. Данные операторы могут быть записаны в виде

```
i++;        // операция инкремента
++i;        // операция инкремента
i--;        // операция декремента
--i;        // операция декремента
```

Разницу между первой и второй формами записи данных операторов можно продемонстрировать на следующем примере:

```
int i=10,j=10;
int a = i++;          //значение a = 10; i = 11;
```

```
int b = ++j;          //значение b = 11; j = 11;
```

Из полученных результатов видно, что если оператор инкремента стоит после имени переменной, то сначала выполняется операция присваивания и только затем операция инкремента. Во втором случае наоборот, операция инкремента реализуется до присвоения результата другой переменной. Поэтому значение a = 10, а значение b = 11.

Задание на лабораторную работу

1. Создать консольный проект.
2. Написать программу вычислений в соответствии с заданным вариантом (числовые параметры задаются самостоятельно).
3. Сделать вывод о полученных результатах работы программы.

Варианты заданий

Вариант	Тип переменных	Вычислить
1	Вещественный Целочисленный	1. Периметр прямоугольника 2. Площадь круга
2	Вещественный Целочисленный	1. Площадь прямоугольника 2. Длину круга
3	Вещественный Целочисленный	1. Площадь треугольника 2. Объем параллелепипеда
4	Вещественный Целочисленный	1. Высоту параллелепипеда 2. Евклидовое расстояние между двумя точками $d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
5	Вещественный Целочисленный	1. $(a+b)^3$ 2. $(a+b)^2$
6	Вещественный Целочисленный	1. $1/a + 2/a + ... + 5/a$ 2. $f(x) = kx + b$, при $x = 1,2,...,5$
7	Вещественный Целочисленный	1. $(a-b)^2$ 2. Площадь круга
8	Вещественный Целочисленный	1. Периметр прямоугольника 2. Объем параллелограмма
9	Вещественный Целочисленный	1. 10% от числа 456 2. $a_1b_1 + a_2b_2 + ... + a_5b_5$
10	Вещественный Целочисленный	1. Длину круга 2. $f(x) = x^2 + b$, при $x = 1,2,...,5$

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Текст программы.
3. Результаты действия программы.
4. Выводы о полученных результатах работы программы.

Контрольные вопросы

1. Приведите примеры правильных имен переменных.
2. Чему будет равна переменная c в строке программы float c=7/2 ?
3. Приведите примеры неправильных имен переменных.
4. Как записывается оператор умножения в языке С?
5. Как изменится значение переменной i в строчке программы i=i+1?
6. Что такое операция декремента?

Лабораторная работа №2

ДИРЕКТИВЫ ПРЕПРОЦЕССОРА И ФУНКЦИИ PRINTF() И SCANF()

Цель работы: изучить особенности работы директив препроцессора и функций printf() и scanf().

Директивы препроцессора

Почти все программы на языке С используют специальные команды для компилятора, которые называются директивами. В общем случае директива – это указание компилятору языка С выполнить то или иное действие в момент компиляции программы. Существует строго определенный набор возможных директив, который включает в себя следующие определения:

#define, #elif, #else, #endif, #if, #ifdef, #ifndef, #include, #undef.

Директива #define используется для задания констант, ключевых слов, операторов и выражений, используемых в программе. Общий синтаксис данной директивы имеет следующий вид:

```
#define <идентификатор> <текст>
```

или

```
#define <идентификатор> (<список параметров>) <текст>
```

Следует заметить, что символ ‘;’ после директив не ставится. Приведем примеры вариантов использования директивы #define.

Листинг 1. Примеры использования директивы #define.

```
#include <stdio.h>
#define TWO 2
#define FOUR TWO*TWO
#define PX printf("X равно %d.\n", x)
#define FMT «X равно %d.\n»
#define SQUARE(X) X*X
int main()
{
 int x = TWO;
 PX;
 x = FOUR;
 printf(FMT, x);
 x = SQUARE(3);
 PX;

 return 0;
}
```

После выполнения этой программы на экране монитора появится три строки:

```
X равно 2.
X равно 4.
X равно 9.
```

Директива #undef отменяет определение, введенное ранее директивой #define. Предположим, что на каком-либо участке программы нужно отменить определение константы FOUR. Это достигается следующей командой:

```
#undef FOUR
```

Интересной особенностью данной директивы является возможность переопределения значения ранее введенной константы. Действительно, повторное использование директивы #define для ранее введенной константы FOUR невозможно, т.к. это приведет к сообщению об ошибке в момент компиляции программы. Но если отменить определение константы FOUR с помощью директивы #undef, то появляется возможность повторного использования директивы #define для константы FOUR.

Для того чтобы иметь возможность выполнять условную компиляцию, используется группа директив #if, #ifdef, #ifndef, #elif, #else и #endif. Приведенная ниже программа выполняет подключение библиотек в зависимости от установленных констант.

```
#if defined(GRAPH)
 #include <graphics.h>  //подключение графической библиотеки
#elif defined(TEXT)
 #include <conio.h>     //подключение текстовой библиотеки
#else
 #include <io.h>        //подключение библиотеки ввода-вывода
#endif
```

Данная программа работает следующим образом. Если ранее была задана константа с именем GRAPH через директиву #define, то будет подключена графическая библиотека с помощью директивы #include. Если идентификатор GRAPH не определен, но имеется определение TEXT, то будет использоваться библиотека текстового ввода/вывода. Иначе, при отсутствии каких-либо определений, подключается библиотека ввода/вывода. Вместо словосочетания #if defined часто используют сокращенные обозначения #ifdef и #ifndef и выше приведенную программу можно переписать в виде:

```
#ifdef GRAPH
 #include <graphics.h>    //подключение графической библиотеки
#ifdef TEXT
 #include <conio.h>  //подключение текстовой библиотеки
#else
 #include <io.h>     //подключение библиотеки ввода-вывода
#endif
```

Отличие директивы #if от директив #ifdef и #ifndef заключается в возможности проверки более разнообразных условий, а не только существует или нет какие-либо константы. Например, с помощью директивы #if можно проводить такую проверку:

```
#if SIZE == 1
 #include <math.h>  // подключение математической библиотеки
#elif SIZE > 1
 #include <array.h>  // подключение библиотеки обработки
                     // массивов
#endif
```

В приведенном примере подключается либо математическая библиотека, либо библиотека обработки массивов, в зависимости от значения константы SIZE.

Используемая в приведенных примерах директива #include позволяет добавлять в программу ранее написанные программы и сохраненные в виде файлов. Например, строка

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

указывает препроцессору добавить содержимое файла stdio.h вместо приведенной строки. Это дает большую гибкость, легкость программирования и наглядность создаваемого текста программы.

Функции ввода/вывода printf() и scanf()

Функция printf() позволяет выводить информацию на экран при программировании в консольном режиме. Данная функция определена в библиотеке stdio.h и имеет следующий синтаксис:

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

Здесь первый аргумент *format определяет строку, которая выводится на экран и может содержать специальные управляющие символы для вывода переменных. Затем, следует список необязательных аргументов, которые поясняются ниже. Функуция возвращает либо число отображенных символов, либо отрицательное число в случае своей некорректной работы.

В самой простой реализации функция printf() просто выводит заданную строку на экран монитора:

```
printf("Привет мир.");
```

Однако с ее помощью можно выводить переменные разного типа: начиная с числовых и заканчивая строковыми. Для выполнения этой операции используются специальные управляющие символы, которые называются спецификаторами и которые начинаются с символа %. Следующий пример демонстрирует вывод целочисленной переменной num на экран монитора с помощью функции printf():

```
int num;
num = 5;
printf("%d", num);
```

В первых двух строках данной программы задается переменная с именем num типа int. В третьей строке выполняется вывод переменной на экран. Работа функции printf() выглядит следующим образом. Сначала функция анализирует строку, которую необходимо вывести на экран. В данном случае это «%d». Если в этой строке встречается спецификатор, то на его место записывается значение переменной, которая является вторым аргументом функции printf(). В результате, вместо исходной строки «%d» на экране появится строка «5», т.е. будет выведено число 5.

Следует отметить, что спецификатор «%d» выводит только целочисленные типы переменных, например int. Для вывода других типов следует использовать другие спецификаторы. Ниже перечислены основные виды спецификаторов:

%c – одиночный символ
%d – десятичное целое число со знаком
%f – число с плавающей точкой (десятичное представление)
%s – строка символов (для строковых переменных)
%u – десятичное целое без знака
%% - печать знака процента

С помощью функции printf() можно выводить сразу несколько переменных. Для этого используется следующая конструкция:

```
int num_i;
float num_f;
num_i = 5;
num_f = 10.5;
printf("num_i = %d, num_f = %f", num_i, num_f);
```

Результат выполнения программы будет выглядеть так:

```
num_i = 5, num_f = 10.5
```

Кроме спецификаторов в функции printf() используются управляющие символы, такие как перевод строки \n, табуляции \t и др. Например, если в ранее рассмотренном примере необходимо вывести значения переменных не в строчку, а в столбик, то необходимо переписать функцию printf() следующим образом:

```
printf("num_i = %d,\n num_f = %f", num_i, num_f);
```

Аналогично используется и символ табуляции.

Для ввода информации с клавиатуры удобно использовать функцию scanf() библиотеки stdio.h, которая имеет следующий синтаксис:

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

Здесь, как и для функции printf(), переменная *format определяет форматную строку для определения типа вводимых данных и может содержать те же спецификаторы что и функция printf(). Затем, следует список необязательных аргументов. Работа функции scanf() демонстрируется на листинге 2.

Листинг 2. Пример использования функции scanf().

```
#include <stdio.h>
int main()
{
 int age;
 float weight;
 printf("Введите информацию о Вашем возрасте: ");
```

```
 scanf("%d", &age);
 printf("Введите информацию о Вашем весе: ");
 scanf("%f", &weigth);
 printf("Ваш возраст = %d, Ваш вес = %f", age, weigth);

 return 0;
}
```

Основным отличием применения функции scanf() от функции printf() является знак & перед именем переменной, в которую записываются результаты ввода.

Функция scanf() может работать сразу с несколькими переменными. Предположим, что необходимо ввести два целых числа с клавиатуры. Формально для этого можно дважды вызвать функцию scanf(), однако лучше воспользоваться такой конструкцией:

```
scanf(" %d,  %d ", &n, &m);
```

Функция scanf() интерпретирует это так, как будто ожидает, что пользователь введет число, затем – запятую, а затем – второе число.

Функция scanf() возвращает число успешно считанных элементов. Если операции считывания не происходило, что бывает в том случае, когда вместо ожидаемого цифрового значения вводится какая-либо буква, то возвращаемое значение равно 0.

Задание на лабораторную работу

1. Написать программу работы с директивами препроцессора в соответствии с номером своего варианта.
2. Написать программу с использованием функций printf() и scanf() в соответствии с номером своего варианта.
3. Сделать выводы о полученных результатах работы программ.

Варианты заданий

Вари-ант	Программирование директив препроцессора	Программирование функций printf() и scanf()
1	Программа вычисления $a+b$ с использованием директивы #define	Ввести два вещественных значения и вывести их произведение на экран монитора
2	С помощью директив #if, #else, #elif осуществить выбор строк программы для вычисления либо $2(a+b)$, либо ab	Ввести два целочисленных значения и вывести их частное на экран монитора
3	Задать константы M и N и вычислить $(aM+bN)/MN$	Ввести два вещественных значения и вывести их на экран с точностью до сотых
4	С помощью директивы #define	Ввести два целочисленных

	вычислить x^2, при $x = 1,2,...,5$	значения и вывести их разность на экран монитора
5	Задать константы $M1, M2,..., M5$ и вычислить $M1 + 2M2 + 3M3 + 4M4 + 5M5$	Ввести целочисленное и вещественное значения и вывести их сумму на экран монитора
6	С помощью директивы #define вычислить $kx + b$, при $x = 1,2,...,5$	Ввести два вещественных значения и вывести их на экран с точностью до тысячных
7	С помощью директив #if, #else, #elif осуществлять выбор строк программы для вычисления либо $(a + b)^2$, либо $(a - b)^2$	Ввести ширину и высоту прямоугольника, вычислить его периметр и вывести результат на экран монитора
8	С помощью директивы #define вычислить x^3, при $x = -2,-1,...,2$	Ввести ширину и высоту прямоугольника, вычислить его площадь и вывести результат на экран монитора
9	Задать константы M и N и вычислить $(M + N)^2 / 2$	Ввести длину основания и высоту равнобедренного прямоугольника, вычислить его площадь и вывести результат на экран монитора
10	С помощью директивы #define вычислить $(x + y)^2$, при $x, y = 1,2,...,5$	Ввести длину основания и высоту равнобедренного прямоугольника, вычислить его периметр и вывести результат на экран монитора

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. Приведите пример использования функции printf() для вывода значений двух целочисленных переменных на экран.
2. Запишите функцию scanf() для ввода символа с клавиатуры
3. Запишите директиву #define для задания константы с именем LENGTH равной 10
4. Приведите пример макроса, позволяющий возводить число в квадрат.
5. С помощью каких директив можно выполнять условную компиляцию программы?
6. Запишите функцию printf() для вывода вещественной переменной с точностью до сотых.

Лабораторная работа №3

УСЛОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА С

Цель работы: изучить особенности использования условных операторов if и switch.

Условные операторы if и switch

Для того чтобы иметь возможность реализовать логику в программе используются условные операторы. Умозрительно эти операторы можно представить в виде узловых пунктов, достигая которых программа делает выбор по какому из возможных направлений двигаться дальше. Например, требуется определить, содержит ли некоторая переменная arg положительное или отрицательное число и вывести соответствующее сообщение на экран. Для этого можно воспользоваться оператором if (если), который и выполняет подобные проверки.

В самом простом случае синтаксис данного оператора if следующий:

```
if (выражение)
 <оператор>
```

Если значение параметра «выражение» равно «истинно», выполняется оператор, иначе он пропускается программой. Следует отметить, что «выражение» является условным выражением, в котором выполняется проверка некоторого условия. В табл. 2 представлены варианты простых логических выражений оператора if.

Таблица 2. Простые логические выражения

if(a < b)	Истинно, если переменная a меньше переменной b и ложно в противном случае.
if(a > b)	Истинно, если переменная a больше переменной b и ложно в противном случае.
if(a == b)	Истинно, если переменная a равна переменной b и ложно в противном случае.
if(a <= b)	Истинно, если переменная a меньше либо равна переменной b и ложно в противном случае.
if(a >= b)	Истинно, если переменная a больше либо равна переменной b и ложно в противном случае.
if(a != b)	Истинно, если переменная a не равна

	переменной b и ложно в противном случае.
if(a)	Истинно, если переменная a не равна нулю, и ложно в противном случае.

Приведем пример использования оператора ветвления if. Следующая программа позволяет определять знак введенной переменной.

Листинг 3. Программа определения знака введенного числа.

```
#include <stdio.h>
int main()
{
 float x;
 printf("Введите число: ");
 scanf("%f",&x);
 if(x < 0)
  printf("Введенное число %f является отрицательным.\n", x);
 if(x >= 0)
  printf("Введенное число %f является неотрицательным.\n", x);

 return 0;
}
```

Анализ приведенного текста программы показывает, что два условных оператора можно заменить одним, используя конструкцию

```
if (выражение)
 <оператор1>
else
 <оператор2>
```

которая интерпретируется таким образом. Если «выражение» истинно, то выполняется «оператор1», иначе выполняется «оператор2».

В случаях, когда при выполнении какого-либо условия необходимо записать более одного оператора, необходимо использовать фигурные скобки, т.е. использовать конструкцию вида

```
if (выражение)
{
 <список операторов>
}
else
{
 <список операторов>
}
```

Следует отметить, что после ключевого слова else формально можно поставить еще один оператор условия if, в результате получим еще более гибкую конструкцию условных переходов:

```
if(выражение1) <оператор1>
else if(выражение2) <опреатор2>
else <оператор3>
```

До сих пор рассматривались простые условия типа x<0. Вместе с тем оператор if позволяет реализовывать более сложные условные переходы. В языке С имеются три логические операции:

&& - логическое И
|| - логическое ИЛИ
! – логическое НЕТ

На основе этих трех логических операций можно сформировать более сложные условия. Например, если имеются три переменные exp1, exp2 и exp3, то они могут составлять логические конструкции, представленные в табл. 3.

Таблица 3. Пример составных логических выражений

if(exp1 > exp2 && exp2 < exp3)	Истинно, если значение переменной exp1 больше значения переменной exp2 и значение переменной exp2 меньше значения переменной exp3.
if(exp1 <= exp2 || exp1 >= exp3)	Истинно, если значение переменной exp1 меньше либо равно значения переменной exp2 или значение переменной exp2 больше либо равно значения переменной exp3.
if(exp1 && exp2 && !exp3)	Истинно, если истинное значение exp1 и истинно значение exp2 и ложно значение exp3.
if(!exp1 || !exp2 && exp3)	Истинно, если ложно значение exp1 или ложно значение exp2 и истинно значение exp3.

Подобно операциям умножения и сложения в математике, логические операции И ИЛИ НЕТ, также имеют свои приоритеты. Самый высокий приоритет имеет операция НЕТ, т.е. такая операция выполняется в первую очередь. Более низкий приоритет у операции И, и наконец самый малый приоритет у операции ИЛИ.

Условная операция if облегчает написание программ, в которых необходимо производить выбор между небольшим числом возможных вариантов. Однако иногда в программе необходимо осуществить выбор одного

варианта из множества возможных. Формально для этого можно воспользоваться конструкцией if else if … else. Однако во многих случаях оказывается более удобным применять оператор switch языка С. Синтаксис данного оператора следующий:

```
switch(переменная)
{
 case константа1:
      <операторы>
 case константа2:
      <операторы>
 …
 default:
      <операторы>
}
```

Данный оператор последовательно проверяет на равенство переменной константам, стоящим после ключевого слова case. Если ни одна из констант не равна значению переменно, то выполняются операторы, находящиеся после слова default. Оператор switch имеет следующую особенность. Допустим, значение переменной равно значению константы1 и выполняются операторы, стоящие после первого ключевого слова case. После этого выполнение программы продолжится проверкой переменной на равенство константы2, что часто приводит к неоправданным затратам ресурсов ЭВМ. Во избежание такой ситуации следует использовать оператор break для перехода программы к следующему оператору после switch.

В листинге 4 представлен пример программирования условного оператора switch.

Листинг 4. Пример использования оператора switch.

```
#include <stdio.h>
int main()
{
 int x;
 printf("Введите число: ");
 scanf("%d",&x);
 switch(x)
 {
      case 1 : printf("Введено число 1\n");break;
      case 2 : printf("Введено число 2\n"); break;
      default : printf("Введено другое число\n");
 }
 char ch;
 printf("Введите символ: ");
 scanf("%c",&ch);
 switch(ch)
 {
      case 'a' : printf("Введен символ a\n"); break;
```

```
        case 'b' : printf("Введен символ b\n"); break;
        default : printf("Введен другой символ\n");
    }
    return 0;
}
```

Данный пример демонстрирует два разных варианта использования оператора switch. В первом случае выполняется анализ введенной цифры, во втором – анализ введенного символа. Следует отметить, что данный оператор может производить выбор только на основании равенства своего аргумента одному из перечисленных значений case, т.е. проверка выражений типа x<0 в данном случае невозможна.

Задание на лабораторную работу

1. Написать программу работы с условным оператором if в соответствии с номером своего варианта.
2. Написать программу с использованием оператора switch в соответствии с номером своего варианта.
3. Сделать выводы о полученных результатах работы программ.

Варианты заданий

Вариант	Оператор if	Оператор switch
1	Написать программу вычисления модуля введенного числа	Написать программу перевода введенного символа от a до f в верхний регистр
2	Написать программу проверки попадания введенного числа в диапазон от -2 до 2	Написать программу перевода введенного символа от A до F в нижний регистр
3	Написать программу проверки не вхождения введенного числа в диапазон от 0 до 5	Написать программу замены введенного символа от 0 до 9 соответствующим числом
4	Написать программу проверки на положительность введенного числа	Написать программу замены введенного числа от 0 до 9 соответствующим символом
5	Написать программу проверки на отрицательность введенного числа	Написать программу замены введенного числа от 0 до 5 соответствующим символом, а все другие значения заменять буквой z.
6	Написать программу определения знака введенного числа	Написать программу замены введенного символа от 0 до 5 соответствующим числом, а все другие символы заменять числом -1
7	Написать программу проверки попадания введенного числа в	Написать программу перевода введенного символа от a до f в

	диапазон от -6 до -2	верхний регистр, а другие символы заменять на Z
8	Написать программу проверки не вхождения введенного числа в диапазон от -5 до -1	Написать программу перевода введенного символа от A до F в нижний регистр, а все другие символы заменять на x
9	Написать программу вычисления суммы модулей двух введенных чисел	Написать программу сравнения введенного числа со значениями 0, 4,8,9 и 30
10	Написать программу вычисления $1/a$ с проверкой $a \neq 0$	Написать программу сравнения введенного символа с a, s, d, j и e

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. Запишите условный оператор if для определения знака переменной var.
2. В каких случаях следует использовать оператор switch?
3. Используя условный оператор, выполните проверку на принадлежность значения переменной диапазону [10; 20).
4. Приведите программу замены малых латинских букв большими с использованием оператора switch.
5. Как записывается логическое равенство в операторе if?
6. Приведите обозначение логического знака «не равно».
7. Какими символами обозначаются логические операции И и ИЛИ в условном операторе if?

Лабораторная работа №4

ОПЕРАТОРЫ ЦИКЛОВ ЯЗЫКА С

Цель работы: изучить особенности использования операторов цикла while, for и do while.

Теоретические сведения

Часто при создании программ на ЭВМ требуется много раз выполнить одну и ту же группу операторов. Например, для вычисления суммы ряда длиной N или перебора элементов массива с целью определения наибольшего или наименьшего значения и т.п. Во всех этих случаях необходим инструмент для реализации повторяющихся операций и таким инструментом являются операторы цикла.

Оператор цикла while

С помощью данного оператора реализуется цикл, который выполняется до тех пор, пока истинно условие цикла. Синтаксис данного оператора следующий:

```
while(<условие>)
{
 <тело цикла>
}
```

Приведем пример реализации данного цикла, в котором выполняется суммирование элементов ряда $S = \sum_{i=0}^{\infty} i$ пока $S < N$:

```
int N=20, i = 0;
long S = 0L;
while(S < N)
{
 S=S+i;
 i++;
}
```

В данном примере реализуется цикл while с условием i < N. Так как начальное значение переменной i=0, а N=20, то условие истинно и выполняется тело цикла, в котором осуществляется суммирование переменной i и увеличение ее на 1. Очевидно, что на 20 итерации значение i=20, условие станет ложным и цикл будет завершен. Продемонстрируем гибкость языка С, изменив данный пример следующим образом:

```
int N=20, i = 0;
long S = 0L;
while((S=S+i++) < N);
```

В данном случае при проверке условия сначала выполняются операторы, стоящие в скобках, где и осуществляется суммирование элементов ряда и только, затем, проверяется условие. Результат выполнения обоих вариантов программ одинаковый и S=21. Однако последняя конструкция бывает удобной при реализации опроса клавиатуры, например, с помощью функции scanf():

```
int num;
while(scanf("%d",&mun) == 1)
{
 printf("Вы ввели значение %d\n",num);
}
```

Данный цикл будет работать, пока пользователь вводит целочисленные значения и останавливается, если введена буква или вещественное число. Следует отметить, что цикл while можно принудительно завершить даже при истинном условии цикла. Это достигается путем использования оператора break. Перепишем предыдущий пример так, чтобы цикл завершался, если пользователь введет число 0.

```
int num;
while(scanf("%d",&mun) == 1)
{
   if(num == 0) break;
 printf("Вы ввели значение %d\n",num);
}
```

Цикл завершается сразу после использования оператора break, т.е. в приведенном примере, при вводе с клавиатуры нуля функция printf() выполняться не будет и программа перейдет на следующий оператор после while. Того же результата можно добиться, если использовать составное условие в цикле:

```
int num;
while(scanf("%d",&mun) == 1 && num != 0)
{
 printf("Вы ввели значение %d\n",num);
}
```

Таким образом, в качестве условия возможны такие же конструкции, что и в операторе if.

Оператор цикла for

Работа оператора цикла for подобна оператору while с той лишь разницей, что оператор for подразумевает изменение значения некоторой переменной и проверки ее на истинность. Работа данного оператора продолжается до тех пор, пока истинно условие цикла. Синтаксис оператора for следующий:

```
for(<инициализация счетчика>;<условие>;<изменение значения
счетчика>)
{
 <тело цикла>
}
```

Рассмотрим особенность реализации данного оператора на примере вывода таблицы кодов ASCII символов.

```
char ch;
for(ch = 'a'; ch <= 'z'; ch++)
      printf("Значение ASCII для %c - %d.\n",ch,ch);
```

В данном примере в качестве счетчика цикла выступает переменная ch, которая инициализируется символом ‘a’. Это означает, что в переменную ch заносится число 97 – код символа ‘a’. Именно так символы представляются в памяти компьютера. Код символа ‘z’ – 122, и все малые буквы латинского алфавита имеют коды в диапазоне [97; 122]. Поэтому, увеличивая значение ch на единицу, получаем код следующей буквы, которая выводится с помощью функции printf(). Учитывая все вышесказанное, этот же пример можно записать следующим образом:

```
for(char ch = 97; ch <= 122; ch++)
      printf("Значение ASCII для %c - %d.\n",ch,ch);
```

Здесь следует отметить, что переменная ch объявлена внутри оператора for. Это особенность языка С - возможность объявлять переменные в любом месте программы.

Существует много особенностей реализации данного оператора, отметим основные из них, которые могут заметно повысить скорость написания программ. Следующим примером продемонстрируем особенности изменения значения счетчика цикла.

```
int line_cnt = 1;
double debet;
for(debet = 100.0; debet < 150.0; debet = debet*1.1,
line_cnt++)
 printf("%d. Ваш долг теперь равен %.2f.\n",line_cnt, debet);
```

Следующий фрагмент программы демонстрирует возможность программирования сложного условия внутри цикла.

```
int exit = 1;
for(int num = 0;num < 100 && !exit; num += 1)
{
 scanf(“%d”,&mov);
 if(mov == 0) exit = 0;
 printf(“Произведение num*mov = %d.\n”,num*mov);
}
```

Оператор for с одним условием:

```
int i=0;
for(;i < 100;) i++;
```

и без условия

```
int i=0;
for(;;;) {i++; if(i > 100) break;}
```

В последнем примере оператор break служит для выхода из цикла for, т.к. он будет работать «вечно» не имея никаких условий.

Оператор цикла do while

Все представленные выше операторы циклов, так или иначе, проверяют условие перед выполнением цикла, благодаря чему существует вероятность, что операторы внутри цикла никогда не будут выполнены. Такие циклы называют циклы с предусловием. Однако бывают ситуации, когда целесообразно выполнять проверку условия после того, как будут выполнены операторы, стоящие внутри цикла. Это достигается путем использования операторов do while, которые реализуют цикл с постусловием. Следующий пример демонстрирует реализацию такого цикла.

```
const int secret_code = 13;
int code_ent;
do
{
 printf(“Введите секретный код: ”);
  scanf(“%d”,&code_ent);
}
while(code_ent != secret_code);
```

Из приведенного примера видно, что цикл с постусловием работает до тех пор, пока истинно условие, т.е. в данном случае пока значение введенного кода будет отличаться от значения секретного кода. Также следует обратить внимание на то, что после ключевого слова while должна стоять точка с запятой. При реализации данного цикла можно использовать составные условия, подобно циклу while, а также принудительно выходить из цикла с помощью оператора break.

Программирование вложенных циклов

Все рассмотренные выше операторы циклов допускают использование любых других операторов языка С внутри цикла, в том числе и операторов цикла. Это значит, что внутри одного цикла может находиться другой, что приводит к реализации вложенных циклов. Вложенные циклы необходимы для решения большого числа задач, например, вычисления двойных, тройных и т.д. сумм, просмотр элементов двумерного массива и многих других задач. В качестве примера вложенных циклов рассмотрим задачу вычисления суммы двойного ряда $S = \sum_{i=0}^{N} \sum_{j=0}^{M} i * j$:

```
long S = 0L;
int M = 10, N = 5;
for(int i = 0; i <= N;i++)
{
 for(int j = 0;j <= M;j++)
      S += i*j;
}
```

Того же результата можно добиться и с помощью оператора цикла while.

Задание на лабораторную работу

1. Написать программу работы с операторами циклов while и for в соответствии с номером своего варианта.
2. Написать программу с использованием оператора цикла do while в соответствии с номером своего варианта.
3. Сделать выводы о полученных результатах работы программ.

Варианты заданий

Вариант	Операторы циклов while и for	Оператор цикла do while
1	Вычислить $\sum_{i=1}^{50} 1/i^2$ с использованием оператора for	Написать программу ввода произвольных чисел до тех пор, пока не будет введено число 0
2	Вычислить $f(x) = kx + b$, при $x = 1,2,...,100$ с использованием оператора while	Написать программу ввода произвольных символов до тех пор, пока не будет введен символ q
3	Вычислить $\sum_{i=1}^{50} \sum_{j=1}^{30} i + j$ с помощью вложенных циклов for	Написать программу подсчета суммы 10 чисел, вводимых с клавиатуры

4	Вычислить $S=\sum_{i=1}^{\infty} i$ пока $S<50$ с помощью цикла while	Написать программу вычисления произведения 5 чисел, введенных с клавиатуры
5	Вычислить $S=\sum_{i=1}^{\infty} i^2$ пока $S<100$ с помощью цикла for	Написать программу вычисления модулей введенных чисел до тех пор, пока пользователь не введет 0
6	Вычислить $\sum_{i=1}^{50}\sum_{j=1}^{10} 1/(i+j)$ с помощью вложенных циклов while	Написать программу определения знака введенных чисел до тех пор, пока пользователь не введет 0
7	Вычислить $f(x)=x^2+b$, при $x=-10,-9,...,10$ с использованием оператора for	Написать программу определения минимального введенного числа из 10 чисел
8	Вычислить $\sum_{i=-10}^{10} 1/i^3$, $i\neq 0$ с использованием оператора for	Написать программу определения максимального введенного числа из 5 чисел
9	Вычислить $\sum_{i=-10}^{20}\sum_{j=0}^{10} 1/(i+j)^2$, $i+j\neq 0$ с помощью вложенных циклов for	Написать программу определения минимального среди положительных введенных 10 чисел
10	Вычислить $f(x)=1/x$, $x\neq 0$ при $x=-10,-9,...,10$ с использованием оператора for	Написать программу определения максимального среди отрицательных введенных 7 чисел

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. В чем отличия между операторами while и do while?
2. Дайте понятие вложенных циклов?
3. Что такое цикл с предусловием?
4. Что такое цикл с постусловием?
5. Условие остановки цикла while?
6. Для каких целей используются циклы в программировании?
7. Перечислите операторы циклов в языке С.

Лабораторная работа №5

МАССИВЫ

Цель работы: изучить базовые операции работы с одномерными и двумерными массивами.

Теоретические сведения

Представление данных в виде отдельных переменных не всегда достаточно при программировании реальных задач. Например, для представления поведения сигнала во времени или хранения информации об изображении удобно использовать специальный тип данных – массивы. Одномерные массивы можно ассоциировать с компонентами вектора, а двумерные – с матрицами. В общем случае массив – это набор элементов данных одного типа, для объявления которого используется следующий синтаксис:

```
<тип данных> <имя массива>[число элементов];
```

Например,

```
int array_int[100];  //одномерный массив 100 целочисленных
                     // элементов
double array_d[25];  //одномерный массив 25 вещественных
                     // элементов
```

Как видно из примеров, объявление массивов отличается от объявления обычных переменных наличием квадратных скобок []. Также имена массивов выбираются по тем же правилам, что и имена переменных. Обращение к отдельному элементу массива осуществляется по номеру его индекса. Следующий фрагмент программы демонстрирует запись в массив значений линейной функции $f(x) = kx + b$ и вывода значений на экран:

```
double k=0.5, b = 10.0;
double f[100];
for(int x=0;i < 100;i++)
{
 f[i] = k*x+b;
 printf("%.2f ",f[i]);
}
```

В языке С предусмотрена возможность инициализации массива в момент его объявления, например, таким образом

```
int powers[4] = {1, 2, 4, 6};
```

В этом случае элементу powers[0] присваивается значение 1, powers[1] – 2, и т.д. Особенностью инициализации массивов является то, что их размер можно задавать только константами, а не переменными. Например, следующая программа приведет к ошибке при компиляции:

```
int N=100;
float array_f[N];    //ошибка, так нельзя
```

Поэтому при объявлении массивов обычно используют такой подход:

```
#include <stdio.h>
#define N  100
int main()
{
 float array_f[N];
 return 0;
}
```

Следует отметить, что при инициализации массивов число их элементов должно совпадать с его размерностью. Рассмотрим вариант, когда число элементов при инициализации будет меньше размерности массива.

```
#define SIZE    4
int data[SIZE]={512, 1024};
for(int i = 0;i < SIZE;i++)
      printf("%d, \n",data[i]);
```

Результат работы программы будет следующим:

512, 1024, 0, 0

Из полученного результата видно, что неинициализированные элементы массива принимаются равными нулю. В случаях, когда число элементов при инициализации превышает размерность массива, то при компиляции произойдет ошибка. Поэтому, когда наперед неизвестно число элементов, целесообразно использовать такую конструкцию языка C++:

```
int data[] = {2, 16, 32, 64, 128, 256};
```

В результате инициализируется одномерный массив размерностью 6 элементов. Здесь остается последний вопрос: что будет, если значение индекса при обращении к элементу массива превысит его размерность? В этом случае ни программа, ни компилятор не выдадут значение об ошибке, но при этом в программе могут возникать непредвиденные ошибки. Поэтому программисту следует обращать особое внимание на то, чтобы индексы при обращении к элементам массива не выходили за его пределы. Также следует отметить, что первый элемент массива всегда имеет индекс 0, второй – 1, и т.д.

Для хранения некоторых видов информации, например, изображений удобно пользоваться двумерными массивами. Объявление двумерных массивов осуществляется следующим образом:

```
int array2D[100][20];     //двумерный массив 100x20 элементов
```

Нумерация элементов также начинается с нуля, т.е. array2D[0][0] соответствует первому элементу, array2D[0][1] – элементу первой строки, второго столбца и т.д. Для начальной инициализации двумерного массива может использоваться следующая конструкция:

```
long array2D[3][2] = {{1, 2}, {3, 4}, {5, 6}};
```

или

```
long array2D[][] = {{1, 2}, {3, 4}, {5, 6}};
```

В общем случае можно задать массив любой размерности и правила работы с ними аналогичны правилам работы с одномерными и двумерными массивами.

Задание на лабораторную работу

1. Написать программу работы с одномерным массивом в соответствии с номером своего варианта.
2. Написать программу с двумерным массивом в соответствии с номером своего варианта.
3. Сделать выводы о полученных результатах работы программ.

Варианты заданий

Вариант	Одномерный массив	Двумерный массив
1	Записать в массив значения функции $f(x) = kx + b$, при $x = 1,2,...,100$ и вывести его на экран	Занести в массив значения функции $f(x, y) = x + y$, $0 \le x \le 20$, $0 \le y \le 10$ и вывести его на экран
2	Записать в массив значения функции $f(x) = a\sin(x/100)$, при $x = 1,2,...,100$ и вывести его на экран	Написать программу ввода в массив 5x4 элемента чисел и поиска в нем максимального значения
3	Написать программу ввода в массив 20 чисел и поиска в нем максимального значения	Занести в массив значения функции $f(x, y) = 1/(x + y)$, $0 \le x \le 30$, $1 \le y \le 20$ и вывести его на экран
4	Записать в массив значения функции $f(x) = a\cos(x/50)$, при $x = 1,2,...,100$ и вывести его на	Написать программу ввода в массив 6x3 элемента чисел и поиска в нем минимального значения

	экран	
5	Написать программу ввода в массив 10 чисел и поиска в нем минимального значения	Занести в массив значения функции $f(x,y)=(x+y)^2$, $0 \le x \le 5$, $0 \le y \le 3$ и вывести его на экран
6	Записать в массив значения функции $f(x)=x^2+b$, при $x=1,2,...,10$ и вывести его на экран	Написать программу ввода в массив 6x5 элементов чисел и вычисления суммы элементов полученного массива
7	Написать программу ввода в массив 20 чисел и вычисления суммы элементов полученного массива	Занести в массив значения функции $f(x,y)=1/((x-y)^2+1)$, $0 \le x \le 5$, $0 \le y \le 10$ и вывести его на экран
8	Написать программу ввода в массив 5 чисел и вычисления произведения элементов полученного массива	Написать программу ввода в массив 3x3 элемента чисел и вычисления произведения элементов полученного массива
9	Записать в массив значения функции $f(x)=1/x+b$, при $x=1,2,...,50$ и вывести его на экран	Занести в массив значения функции $f(x,y)=x-y$, $0 \le x \le 20$, $0 \le y \le 10$ и вывести его на экран
10	Написать программу ввода в массив 10 чисел и поиска в нем модуля максимального значения	Написать программу ввода в массив 4x4 элементов чисел и поиска в нем модуля максимального значения

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. Каким образом задаются одномерные массивы в языке С?
2. Запишите массив целых чисел с начальными значениями 1, 2 и 3.
3. Каким образом задаются двумерные массивы в языке С?
4. В чем преимущества массивов перед переменными?
5. Как записать значение в элемент массива?
6. Как отобразить элементы массива на экране монитора?

Лабораторная работа №6

РАБОТА СО СТРОКАМИ В ЯЗЫКЕ С

Цель работы: изучить базовые операции работы со строками.

Теоретические сведения

В языке С нет специального типа данных для строковых переменных. Для этих целей используются массивы символов (тип char). Следующий пример демонстрирует использование строк в программе:

```
char str_1[100] = {'П','р','и','в','е','т','\0'};
char str_2[100] = "Привет";
char str_3[] = "Привет";
printf("%s\n%s\n%s\n",str_1,str_2,str_3);
```

В приведенном примере показаны три способа инициализации строковых переменных. Первый способ является классическим объявлением массива, второй и третий используются специально для строк. Причем в последнем случае, компилятор сам определяет нужную длину массива для записи строки. Анализируя первый и второй способы инициализации массива символов возникает вопрос: каким образом язык С++ «знает» где заканчивается строка? Действительно, массив str_2 содержит 100 элементов, а массив str_3 меньше 100, тем не менее длина строки и в первом и во втором случаях одна и та же. Такой эффект достигается за счет использования специальных управляющих кодов, которые говорят где заканчивается строка или где используется перенос внутри одной строки и т.п. В частности символ '\0' означает в языке С++ конец строки и все символы после него игнорируются как символы строки. Следующий пример показывает особенность использования данного специального символа.

```
char str1[10] = {'Л','е','к','ц','и','я','\0'};
char str2[10] = {'Л','е','к','ц', '\0','и','я' };
char str3[10] = {'Л','е', '\0','к','ц','и','я' };
printf("%s\n%s\n%s\n",str1,str2,str3);
```

Результатом работы данного кода будет вывод следующих трех строк:

Лекция
Лекц
Ле

Из этого примера видно как символ конца строки '\0' влияет на длину строк. Таким образом, чтобы подсчитать длину строки (число символов) необходимо считать символы до тех пор, пока не встретится символ '\0' или не будет достигнут конец массива. Функция вычисления размера строк уже

реализована в стандартной библиотеке языка C string.h и имеет следующий синтаксис:

```
int strlen(char* str);
```

где char* str – указатель на строку (об указателях речь пойдет ниже). Следующая программа показывает использование функции strlen().

Листинг 5. Пример использования функции strlen().

```
#include <stdio.h>
#include <string.h>
int main(void) {
 char str[] = "Привет мир!";
 int length = strlen(str);
 printf("Длина строки = %d.\n",length);
 return 0;
}
```

Результатом работы программы будет вывод на экран числа 11. Учитывая, что первый символ имеет нулевой индекс, то можно заметить, что данная функция считает и символ '\0'.

Теперь рассмотрим правила присваивания одной строковой переменной другой. Допустим, объявлены две строки

```
char str1[] = "Это первая строка";
char str2[] = "Это вторая строка";
```

и необходимо выполнить оператор присваивания

```
str1 = str2;
```

При такой записи оператора присваивания компилятор выдаст сообщение об ошибке. Для того чтобы выполнить копирование необходимо перебирать по порядку элементы одного массива и присваивать их другому массиву. Данная функция реализована в библиотеке языка C string.h и имеет следующее определение:

```
char* strcpy(char* dest, char* src);
```

Она выполняет копирование строки src в строку dest и возвращает строку dest. В листинге 6 показано использование функции strcpy().

Листинг 6. Пример использования функции strcpy().

```
#include <stdio.h>
#include <string.h>
int main(void) {
```

```
 char src[] = "Привет мир!";
 char dest[100];
 strcpy(dest,src);
 printf("%s\n",dest);
 return 0;
}
```

Кроме операций вычисления длины строки и копирования строк важной является операция сравнения двух строк между собой. В языке С две строки считаются одинаковыми, если равны их длины и элементы одной строки равны соответствующим элементам другой. Функция сравнения двух строк имеет вид:

```
int strcmp(char* str1, char* str2);
```

и реализована в библиотеке string.h. Данная функция возвращает нуль, если строки str1 и str2 равны и не нуль в противном случае. Приведем пример использования данной функции.

```
char str1[] = "Это первая строка";
char str2[] = "Это вторая строка";
if(strcmp(str1,str2) == 0) printf("Срока %s равна строке %s\n",str1,str2);
else printf("Срока %s не равна строке %s\n",str1,str2);
```

В языке С имеется несколько функций, позволяющих вводить строки с клавиатуры. Самой распространенной из них является ранее рассмотренная функция scanf(), которой в качестве параметра передается ссылка на массив символов:

```
char str[100];
scanf("%s",str);
```

В результате выполнения этого кода, переменная str будет содержать введенную пользователем последовательность символов. Кроме функции scanf() также часто используют функцию gets() библиотеки stdio.h, которая в качестве аргумента принимает ссылку на массив символов:

```
gest(str);
```

Данная функция считывает символы до тех пор, пока пользователь не нажмет клавишу Enter, т.е. введет символ перевода строки ‘\n’. Затем она записывает вместо символа ‘\n’ символ ‘\0’ и передает строку вызывающей программе.

Для вывода строк на экран помимо функции printf() можно использовать также функцию puts() библиотеки stdio.h, которая более проста в использовании. Следующий пример демонстрирует применение данной функции.

```
#define DEF “Заданная строка”
char str[] = “Это первая строка”;
puts(str);
puts(DEF);
puts(&str[4]);
```

Результат работы следующий:

Это первая строка
Заданная строка
первая строка

Еще одной удобной функцией работы со строками является функция sprintf() библиотеки stdio.h. Ее действие аналогично рассмотренной ранее функции printf() с той лишь разницей, что результат вывода заносится в строковую переменную, а не на экран:

```
int age;
char name[100], str[100];
printf(“Введите Ваше имя: ”);
scanf(“%s”,name);
printf(“Введите Ваш возраст: ”);
scanf(“%d”,&age);
sprintf(str,”Здраствуйте %s. Ваш возраст %d лет”,name,age);
puts(str);
```

В результате массив str будет содержать строку «Здраствуйте … Ваш возраст…».

Анализ последнего примера показывает, что с помощью функции sprintf() можно преобразовывать числовые переменные в строковые, объединять несколько строк в одну и т.п.

Задание на лабораторную работу

1. Написать две программы по работе со строками в соответствии с номером своего варианта.

Варианты заданий

Вариант	1-е задание	2-е задание
1	Написать программу поэлементного копирования строки «Hello World» в другой символьный массив	Написать программу объединения трех строк “The laboratory”, “work” и “№6” в четвертую строку с текстом: «The laboratory work №6» без использования функции sprintf()
2	Написать программу замены во веденной строке малых букв а на	Написать программу удаления букв н из введенной строки

	заглавные	
3	Написать программу подсчета букв е во введенной строке	Написать программу добавления слова «hello» после первого слова введенной строки *)
4	Написать программу удаления букв о из введенного слова	Написать программу сравнения двух строк с помощью функции strcmp()
5	Написать программу добавления пробела после каждой буквы а введенной строки	Написать программу замены во веденной строке заглавных букв О на малые
6	Написать программу подсчета числа слов в строке *)	Написать программу подсчета букв ‘и’ во введенной строке
7	Написать программу выделения первого слова из введенной строки *) и отображение его на экране	Написать программу удаления всех пробелов из введенной строки
8	Написать программу выделения последнего слова из введенной строки *) и отображение его на экране	Написать программу копирования первой половины введенной строки в другую строку
9	Написать программу вывода введенного слова задом на перед.	Написать программу сравнения первых половин двух введенных строк
10	Написать программу удаления последнего слова из строки *)	Написать программу замещения первой половины строки второй, а второй – первой

* считается, что слова разделяются пробелом

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. Как задаются строки в программе на языке С?
2. Для чего предназначена функция strcpy() и в какой библиотеке она определена?
3. Запишите возможные способы начальной инициализации строки.
4. Какой управляющий символ соответствует концу строки?
5. Что выполняет функция strcmp()?
6. Какую роль играют структуры в программировании?
7. Что возвращает функция strlen()?

Лабораторная работа №7

ФУНКЦИИ

Цель работы: научиться задавать свои функции и изучить правила работы с ними.

Теоретические сведения

В ранее рассмотренных примерах неоднократно использовались различные функции подключаемых библиотек. Вместе с тем существующих функций языка С недостаточно для написания собственных программ и возникает необходимость создания своих функций. В связи с этим нужно понимать, в каких случаях целесообразно создавать свои функции. Обычно это делается для избавления много раз писать один и тот же код в программе. Например, если часто выполняются действия копирования одной строки в другую, то такую операцию лучше определить в виде функции и использовать ее по мере необходимости. Для объявления функции используется следующий синтаксис:

```
<тип> <имя функции> ([список параметров]) { <тело функции> }
```

Тип определяет возвращаемый тип функции. Имя функции служит для ее вызова в программе и ее правило определения совпадает с правилом определения имен переменных. Список параметров необходим для передачи функции каких-либо данных при ее вызове. Телов функции – это набор операторов, которые выполняются при ее вызове. Следующий пример показывает правило определения пользовательских функций.

Листинг 7. Пример задания функции.

```
double square(double x)
{
 x = x*x;
 return x;
}
int main()
{
 double arg = 5;
 double sq1=square(arg);
 double sq2=square(3);
 return 0;
}
```

В данном примере задается функция с именем square, которая принимает один входной параметр типа double, возводит его в квадрат и возвращает вычисленное значение вызывающей программе с помощью оператора return. Следует отметить, что работа функции завершается при вызове оператора

return. Даже если после этого оператора будут находиться другие операторы, то они выполняться не будут. Например,

```
int square(int x)
{
 x = x*x;
 return x;
 printf("%d",x);
}
```

при вызове данной функции оператор printf() не будет выполнен никогда, т.к. оператор return завершит работу функции square. Оператор return является обязательным, если функция возвращает какие-либо значения. Если же она имеет тип void, т.е. ничего не возвращает, то оператор return может не использоваться.

Пользуясь рассмотренными правилами, можно создавать множество своих функций. При этом важно, чтобы объявление функции было раньше ее использования в программе подобно переменным. Именно поэтому во всех примерах объявление функций осуществляется до функции main(), в которой они вызываются.

Функция может принимать произвольное число аргументов, но возвращает только один или не одного (тип void). Для задания нескольких аргументов функции используется следующая конструкция:

```
void show(int x,int y,int z) {}
```

Здесь следует обратить внимание на то, что каждой переменной в списке аргументов функции предшествует ее тип. В отличие от объявления обычных переменных. Поэтому следующая программная строка приведет к сообщению об ошибке на этапе компиляции:

```
void show(int x, y, z) {}      //неверное объявление
```

Если число пользовательских функций велико (50 и выше), то возникает неудобство в их визуальном представлении в общем тексте программы. Действительно, имея список из 100 разных функций с их реализациями, в них становится сложно ориентироваться и вносить необходимые изменения. Для решения данной проблемы в языке С при создании своих функций можно пользоваться правилом: сначала задаются объявления функции, а затем их реализации.

Язык С позволяет задавать функции с одинаковыми именами, но разными типами входных аргументов. Следующий пример демонстрирует удобство использования таких функций при их вызове.

Листинг 8. Пример использования перегруженных функций.

```
#include <stdio.h>
double abs(double arg);
float abs(float arg);
int abs(int arg);
int main()
{
 double a_d = -5.6;
 float a_f = -3.2;
 int a_i;
 a_d = abs(a_d);
 a_f = abs(a_f);
 a_i = abs(-8);
 return 0;
}
double abs(double arg)
{
 if(arg < 0) arg = arg*(-1);
 return arg;
}
float abs(float arg)
{
 return (arg < 0) ? –arg : arg;
}
int abs(int arg)
{
 return (arg < 0) ? –arg : arg;
}
```

В представленной программе задаются три функции с именем abs и разными входными и выходными аргументами для вычисления модуля числа. Благодаря такому объявлению при вычислении модуля разных типов переменных в функции main() используется вызов функции с одним и тем же именем abs. При этом компилятор в зависимости от типа переменной автоматически выберет нужную функцию. Такой подход к объявлению функций называется перегрузкой.

В языке С можно задавать значения аргументов функции, которые будут использоваться по умолчанию, т.е. если программист не введет свое значение. Приведенный ниже фрагмент программы демонстрирует правило использования аргументов по умолчанию.

```
void some_func(int a = 1, int b = 2, int c = 3)
{
 printf("a = %d, b = %d, c = %d\n",a,b,c);
}
```

Благодаря начальной инициализации значений переменных, функция some_func() может быть вызвана с разным набором аргументов:

```
int main(void)
{
```

```
 show_func();
 show_func(10);
 show_func(10,20);
 show_func(10,20,30);
 return 0;
}
```

В результате, на экране появятся следующие строки:

a = 1, b = 2, c = 3
a = 10, b = 2, c = 3
a = 10, b = 20, c = 3
a = 10, b = 20, c = 30

Из полученного результата видно, что по умолчанию значения аргументов равны установленным значениям при определении функции. В случае ввода новых значений, переменные a, b и c соответственно меняют свои значения на введенные.

При использовании значений аргументов по умолчанию следует пользоваться правилом: аргументы со значениями по умолчанию должны находиться в списке аргументов функции последними. Следующий пример показывает правильные и неправильные объявления функций:

```
void my_func(int a, int b = 1, int c = 1);//правильное объявление
void my_func(int a, int b, int c = 1);     //правильное объявление
void my_func(int a=1, int b, int c = 1); //неправильное объявление
void my_func(int a, int b = 1, int c);    //неправильное объявление
```

В языке C допускается чтобы функция вызывала саму себя. Этот процесс называется рекурсией. В некоторых задачах программирования такой подход позволяет заметно упростить создаваемый программный код. Рассмотрим данный процесс на следующем примере.

Листинг 9. Пример использования рекурсивных функций.

```
#include <stdio.h>
void up_and_down(int );
int main(void)
{
 up_and_down(1);
 return 0;
}
void up_and_down(int n)
{
 printf("Уровень вниз %d\n",n);
 if(n < 4) up_and_down(n+1);
 printf("Уровень вверх %d\n",n);
}
```

Результатом работы этой программы будет вывод на экран следующих строк:

Уровень вниз 1
Уровень вниз 2
Уровень вниз 3
Уровень вниз 4
Уровень вверх 4
Уровень вверх 3
Уровень вверх 2
Уровень вверх 1

Полученный результат работы программы объясняется следующим образом. Вначале функция main() вызывает функцию up_and_down() с аргументом 1. В результате аргумент n данной функции принимает значение 1 и функция printf() печатает первую строку. Затем выполняется проверка и если n < 4, то снова вызывается функция up_and_down() с аргументом на 1 больше n+1. В результате вновь вызванная функция печатает вторую строку. Данный процесс продолжается до тех пор, пока значение аргумента не станет равным 4. В этом случае оператор if не сработает и вызовется функция printf(), которая печатает пятую строку «Уровень вверх 4». Затем функция завершает свою работу и управление передается функции, которая вызывала данную функцию. Это функция up_and_down() с аргументом n=3, которая также продолжает свою работу и переходит к оператору printf(), который печатает 6 строку «Уровень вверх 3». Этот процесс продолжается до тех пор, пока не будет достигнут исходный уровень, т.е. первый вызов функции up_and_down() и управление вновь будет передано функции main(), которая завершит работу программы.

Задание на лабораторную работу

1. Написать две программы по работе с функциями в соответствии с номером своего варианта.

Варианты заданий

Вариант	1-е задание	2-е задание
1	Написать функцию вычисления площади прямоугольника	Используя перегрузку функций, написать программу определения знака переменных разного типа
2	Написать функцию вычисления периметра прямоугольника	С помощью рекурсивной функции осуществить вывод на экран элементов одномерного массива
3	Написать функцию вычисления длины окружности	Используя перегрузку функций, написать программу вычисления суммы элементов массива разных типов

4	Написать функцию вычисления площади круга	С помощью рекурсивной функции осуществить поиск максимального элемента одномерного массива
5	Написать функцию вычисления объема параллелепипеда	Используя перегрузку функций, написать программу определения максимального значения элемента массива разного типа
6	Написать функцию вычисления евклидового расстояния между двумя точками	С помощью рекурсивной функции осуществить поиск минимального элемента одномерного массива
7	Написать функцию вычисления суммы элементов массива	Используя перегрузку функций, написать программу определения минимального значения элемента массива разного типа
8	Написать функцию нахождения максимального значения элемента массива	С помощью рекурсивной функции вычислить сумму элементов одномерного массива
9	Написать функцию нахождения минимального значения элемента массива	Используя перегрузку функций, написать программу вычисления произведения двух переменных разного типа
10	Написать функцию вычисления произведения элементов массива	С помощью рекурсивной функции вычислить среднее арифметическое элементов одномерного массива

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы о полученных результатах работы программ.

Контрольные вопросы

1. Запишите прототип функции, которая принимает два целочисленных аргумента и возвращает вещественное число.
2. Допустим, даны три функции:
   ```
   int abs(int x);
   float abs(float x);
   long abs(long x);
   ```
   Какая из этих трех функций будет вызвана в строке float a = abs(-6);?
3. Запишите функцию возведения числа в квадрат.
4. Дайте понятие рекурсии.
5. В каких задачах целесообразно использовать рекурсивные функции?
6. Приведите функцию с тремя аргументами, один из которых задан со значением по умолчанию.

Лабораторная работа №8

СТРУКТУРЫ

Цель работы: изучить синтаксис и правила работы со структурами.

Теоретические сведения

При разработке программ важным является выбор эффективного способа представления данных. Во многих случаях недостаточно объявить простую переменную или массив, а нужна более гибкая форма представления данных. Таким элементом может быть структура, которая позволяет включать в себя разные типы данных, а также другие структуры. Приведем пример, в котором использование структуры позволяет эффективно представить данные. Таким примером будет инвентарный перечень книг, в котором для каждой книги необходимо указывать ее наименование, автора и год издания. Причем количество книг может быть разным, но будем полгать, что не более 100. Для хранения информации об одной книге целесообразно использовать структуру, которая задается в языке С с помощью ключевого слова struct, за которым следует ее имя. Само определение структуры, т.е. то, что она будет содержать, записывается в фигурных скобках {}. В данном случае структура будет иметь следующий вид:

```
struct book {
 char title[100];    //наименование книги
 char author[100];   //автор
 int year;           //год издания
};
```

Такая конструкция задает своего рода шаблон представления данных, но не сам объект, которым можно было бы оперировать подобно переменной или массиву. Для того чтобы объявить переменную для структуры с именем book используется такая запись:

```
struct book lib;     //объявляется переменная типа book
```

После объявления переменной lib имеется возможность работать со структурой как с единым объектом данных, который имеет три поля: title, author и year. Обращение к тому или иному полю структуры осуществляется через точку: lib.title, lib.author и lib.year. Таким образом, для записи в структуру информации можно использовать следующий фрагмент программы:

```
printf("Введите наименование книги: ");
scanf("%s",lib.title);
printf("Введите автора книги: ");
scanf("%s",lib.author);
```

```
printf("Введите год издания книги: ");
scanf("%d",&lib.year);
```

После этого в соответствующие поля будет записана введенная с клавиатуры информация и хранится в единой переменной lib. Однако по условиям задачи необходимо осуществлять запись не по одной, а по 100 книгам. В этом случае целесообразно использовать массив структур типа book, который можно задать следующим образом:

```
struct book lib[100];
```

В этом случае программу ввода и хранения информации по книгам можно записать в виде:

Листинг 1. Инвентарный перечень книг.

```
#include <stdio.h>
struct book {
 char title[100];    //наименование книги
 char author[100];   //автор
 int year;      //год издания
};

int main()
{
 int cnt_book = 0, ch;
 struct book lib[100];
 do
 {
      printf("Введите наименование книги: ");
      scanf("%s",lib[cnt_book].title);
      printf("Введите автора книги: ");
      scanf("%s",lib[cnt_book].author);
      printf("Введите год издания книги: ");
      scanf("%d",&lib.year);
      printf("Нажмите q для завершения ввода: ");
      cnt_book++;
 }
 while(scanf("%d",ch) == 1 && cnt_book < 100);
 return 0;
}
```

Данный пример показывает удобство хранения информации по книгам. Тот же алгоритм в общем случае можно реализовать и без структуры, но тогда пришлось бы использовать два двумерных массива символов и один одномерный массив для хранения года издания. Несмотря на то, что формально такая запись была бы корректной с точки зрения языка С, но менее удобна в обращении. Графически массив структур можно представить в виде таблицы, в которой роль столбцов играют поля, а роль строк элементы массива структур.

	название	автор	год издания
lib[0]	lib[0].title	lib[0].author	lib[0].year
lib[1]	lib[1].title	lib[1].author	lib[1].year
lib[2]	lib[2].title	lib[2].author	lib[2].year
⋮			
lib[99]	lib[99].title	lib[99].author	lib[99].year

Структуры можно автоматически инициализировать при их объявлении подобно массивам, используя следующий синтаксис:

```
struct bool lib = {
 "Евгений Онегин",
 "Пушкин А.С.",
 1995
};
```

При выполнении данного фрагмента программы в переменные структуры title, author и year будет записана соответственно информация: “Евгений Онегин”, “Пушкин А.С.”, 1995. Здесь следует обратить внимание, что последовательность данных при инициализации должна соответствовать последовательности полей в структуре. Это накладывает определенные ограничения, т.к. при инициализации необходимо помнить последовательность полей в структуре. Стандарт С99 допускает более гибкий механизм инициализации полей структуры:

```
struct book lib = {.year = 1995,
                   .author = "Пушкин А.С.",
                   .title = "Евгений Онегин" };
```

или

```
struct book lib = { .year = 1995,
                   .title = "Евгений Онегин" };
```

или

```
struct book lib = {.author = "Пушкин А.С.",
                   .title = "Евгений Онегин",
                   1995 };
```

В первом и во втором примерах при инициализации указываются наименования полей через точку. При этом их порядок и число не имеет значения. В третьем примере первые два поля указаны через имена, а последнее

инициализируется по порядковому номеру – третьему, который соответствует полю year.

В некоторых случаях имеет смысл создавать структуры, которые содержат в себе другие (вложенные) структуры. Например, при создании простого банка данных о сотрудниках предприятия целесообразно ввести, по крайней мере, две структуры. Одна из них будет содержать информацию о фамилии, имени и отчестве сотрудника, а вторая будет включать в себя первую с добавлением полей о профессии и возрасте:

```
struct tag_fio {
 char last[100];
 char first[100];
 char otch[100];
};
struct tag_people {
 struct tag_fio fio; //вложенная структура
 char job[100];
 int old;
};
```

Рассмотрим способ инициализации и доступ к полям структуры people на следующем примере.

Листинг 2. Работа с вложенными структурами.

```
int main()
{
 struct people man = {
      {“Иванов”, “Иван”, “Иванович”},
      “Электрик”,
      50 };
 printf(“Ф.И.О.:%s %s %s\n”,man.fio.last,man.fio.first,
                                          man.fio.otch);
 printf(“Профессия : %s \n”,man.job);
 printf(“Возраст : %d\n”,man.old);
 return 0;
}
```

В данном примере показано, что для инициализации структуры внутри другой структуры следует использовать дополнительные фигурные скобки, в которых содержится информация для инициализации полей фамилии, имени и отчества сотрудника. Для того чтобы получить доступ к полям вложенной структуры выполняется сначала обращение к ней по имени man.fio, а затем к ее полям: man.fio.last, man.fio.first и man.fio.otch. Используя данное правило, можно создавать многоуровневые вложения для эффективного хранения и извлечения данных.

Структуры, как и обычные типы данных, можно передавать функции в качестве аргумента. Следующий пример демонстрирует работу функции отображения полей структуры на экран.

Листинг 3. Передача структур через аргументы функции.

```
#include <stdio.h>
struct tag_people {
 char name[100];
 char job[100];
 int old;
};
void show_struct(struct tag_people man);
int main()
{
 struct tag_people person = {“Иванов”,”Электрик”,30};
 show_struct(person);
 return 0;
}
void show_struct(struct tag_people man)
{
 printf(“Имя: %s\n”,man.name);
 printf(“Профессия: %s\n”,man.job);
 printf(“Возраст: %d\n”,man.old);
}
```

В приведенном примере используется функция с именем show_struct, которая имеет тип аргумента struct tag_people и переменную-структуру man. При передаче структуры функции создается ее копия, которая доступная в теле функции show_struct под именем man. Следовательно, любые изменения полей структуры с именем man никак не повлияют на содержание структуры с именем person. Вместе с тем иногда необходимо выполнять изменение полей структуры функции и возвращать измененные данные вызывающей программе. Для этого можно задать функцию, которая будет возвращать структуру, как показано в листинге 4.

Листинг 4. Функции, принимающие и возвращающие струткуру.

```
#include <stdio.h>
struct tag_people {
 char name[100];
 char job[100];
 int old;
};
void show_struct(struct tag_people man);
struct tag_people get_struct();
int main()
{
 struct tag_people person;
 person = get_struct();
 show_struct(person);
 return 0;
}
```

```
void show_struct(struct tag_people man)
{
 printf("Имя: %s\n",man.name);
 printf("Профессия: %s\n",man.job);
 printf("Возраст: %d\n",man.old);
}
struct tag_people get_struct()
{
 struct tag_people man;
 scanf("%s",man.name);
 scanf("%s",man.job);
 scanf("%d",man.old);
 return man;
}
```

В данном примере используется функция get_struct(), которая инициализирует структуру с именем man, запрашивает у пользователя ввод значений ее полей и возвращает введенную информацию главной программе. В результате выполнения оператора присваивания структуры man структуре person, происходит копирование информации соответствующих полей и автоматическое удаление структуры man.

Функциям в качестве аргумента можно также передавать массивы структур. Для этого используется следующее определение:

```
void show_struct(struct people mans[], int size);
```

Здесь size – число элементов массива, которое необходимо для корректного считывания информации массива mans. Следующий пример показывает принцип работы с массивами структур.

Листинг 5. Передача массив структур функции.

```
#include <stdio.h>
#define N 2
struct tag_people {
 char name[100];
 char job[100];
 int old;
};
void show_struct(struct people mans[], int size);
int main()
{
 struct people persons[N] = {
 { "Иванов", «Электрик», 35 },
 { "Петров", «Преподаватель», 50 },
 };
 show_struct(persons, N);
}
void show_struct(struct people mans[], int size)
{
```

```
 for(int i = 0;i < size;i++) {
      printf("Имя: %s\n",mans[i].name);
      printf("Профессия: %s\n",mans[i].job);
      printf("Возраст: %d\n",mans[i].old);
 }
}
```

При передаче аргумента persons выполняется копирование информации в массив mans и указывается дополнительный параметр size, для определения числа элементов массива mans. Затем в функции show_struct() реализуется цикл, в котором выполняется отображение информации массива структуры на экран монитора.

Задание на лабораторную работу

1. Написать программу работы со структурой в соответствии с номером своего варианта.
2. Написать программу работы с массивом структур в соответствии с номером своего варианта.
3. Сделать выводы по полученным результатам работы программы.

Варианты заданий

Вариант	Задание со структурой	Задание с массивом структур
1	Написать программу ввода домашнего адреса в структуру	Написать программу поиска книги по году издания в массиве структур
2	Написать программу ввода информации по сотруднику (Ф.И.О.,возраст,должность, кафедра)	Написать программу удаления книги из массива структур с введенным именем автора
3	Написать программу ввода информации по студенту (Ф.И.О.,группа,факультет,курс)	Написать программу поиска числа книг с заданным годом издания
4	Написать программу копирования одной структуры (с информацией о книге) в другую	Написать программу сортировки книг в массиве структур по убыванию года издания
5	Написать функцию сравнения двух структур (шаблон структуры задается произвольно)	Написать программу добавления новой книги в начало массива структур.
6	Написать функцию, принимающую значения полей структуры и возвращающую ее заполненную	Написать программу поиска сотрудников с указанным именем в массиве структур сотрудников
7	Написать программу ввода адреса учреждения (Название, город, улица, дом, подъезд)	Написать программу сортировки студентов по возрастанию номера их группы
8	Написать программу ввода информации о маршрутном такси (номер, стоимость,вид транспорта,…)	Написать программу удаления информации о сотруднике с указанным возрастом

9	Написать программу ввода информации по книге (автор, год изд., число стр., цена, название)	Написать программу сортировки книг по возрастанию их цен
10	Написать программу ввода информации о продукте (наименование, цена, магазин, вес, габариты)	Написать программу удаления учреждений с указанным названием из массива структур учреждений

Содержание отчета

5. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
6. Текст программ.
7. Результаты действия программ.
8. Выводы по полученным результатам работы программ.

Контрольные вопросы

7. Запишите структуру для хранения имени, возраста и места работы сотрудника.
8. Как задаются переменные на структуры?
9. Как задаются массивы структур?
10. Запишите инициализацию структуры хранения книг.
11. Каким образом передаются структуры функциям?
12. Можно передавать функции массив структур?

Лабораторная работа №9

ОБЪЕДИНЕНИЯ

Цель работы: изучить синтаксис и правила работы с объединениями.

Теоретические сведения

Еще одним важным типом представления данных являются объединения. Это тип данных, который позволяет хранить различные типы данных в одной и той же области памяти. Объединение задается с помощью ключевого слова union подобно структуре. Покажем особенность применения объединений на примере хранения данных, которые могут быть и вещественными и целыми или представлять собой символ. Так как наперед неизвестно какой тип данных требуется сохранять, то в объединении необходимо задать три поля:

```
union tag_value {
  int var_i;
  double var_f;
  char var_ch;
```

```
};
```

Данное объединение будет занимать в памяти 8 байт, ровно столько, сколько занимает переменная самого большого объема, в данном случае var_f типа double. Все остальные переменные var_i и var_ch будут находиться в той же области памяти, что и переменная var_f. Благодаря такому подходу происходит экономия памяти, но платой за это является возможность хранения значения только одной переменной из трех в определенный момент времени. Как видно из постановки задачи такое ограничение не будет влиять на работоспособность программы. Если бы вместо объединения использовалась структура tag_value, то можно было бы сохранять значения всех трех переменных одновременно, но при этом требовалось бы больше памяти.

Для того чтобы программа «знала» какой тип переменной содержит объединение tag_value, необходимо ввести переменную, значение которой будет указывать номер используемой переменной: 0 – var_i, 1 – var_f и 2 – var_ch. Причем эту переменную удобно представить с объединением в виде структуры следующим образом:

```
struct tag_var {
  union tag_value value;
  short type_var;
};
```

Таким образом, получаем следующий алгоритм записи разнородной информации в объединение tag_value:

```
int main()
{
  struct tag_var var[3];
  var[0].type_var = 0;
  var[0].value.var_i = 10;
  var[1].type_var = 1;
  var[1].value.var_f = 2.3;
  var[2].type_var = 2;
  var[2].value.var_ch = ‘d’;
  for(int i = 0;i < 3;i++)
  {
        switch(var[i].type_var)
        {
        case 0:printf(“var = %d\n”,var[i].value.var_i);break;
        case 1:printf(“var = %f\n”,var[i].value.var_f);break;
        case 2:printf(“var = %c\n”,var[i].value.var_ch);break;
        default: printf(“Значение переменной не определено\n”);
        }
  }
  return 0;
}
```

Как видно из примера объединение tag_value позволяет эффективно, с точки зрения объема памяти, сохранять переменные разного типа. Выигрыш в

объеме памяти для данного случая незначителен около 9-15 байт (в зависимости от стандарта языка С), но если бы таких переменных было 1000 и более, то выигрыш был бы ощутимым. В этом и заключается особенность объединений – экономия памяти при хранении данных.

Задание на лабораторную работу

1. Написать программу работы с объединением в соответствии с номером своего варианта.
2. Написать программу работы с массивом объединений в соответствии с номером своего варианта.
3. Сделать выводы по полученным результатам работы программы.

Варианты заданий

Вариант	Задание с объединением	Задание с массивом объединений
1	Написать программу хранения координаты точки в виде объединений внутри структуры для целочисленного и вещественного типов данных	Написать программу поиска заданной координаты точки в массиве структур, содержащие объединения
2	Написать программу хранения целочисленного, либо вещественного типа данных в переменной	Написать программу удаления заданного числового значения из массива объединений
3	Написать программу хранения действительной и мнимой частей комплексного числа для целочисленного и вещественного типов данных	Написать программу поиска комплексного числа в массиве структур, содержащих объединения
4	Написать программу копирования одного объединения (с информацией о координате точки) в другое	Написать программу сортировки координат точек в массиве структур, содержащих объединения по убыванию их евклидового расстояния относительно точки (0, 0)
5	Написать функцию сравнения двух структур с объединениями, содержащих комплексное число	Написать программу добавления нового комплексного числа в начало массива структур с объединениями
6	Написать функцию, принимающую структуру с объединением, хранящее число произвольного типа и возвращающую ее заполненную	Написать программу поиска заданного числа произвольного типа в массиве структур с объединением
7	Написать программу ввода информации о маршрутном такси (номер, стоимость, вид транспорта), где стоимость может быть как целочисленном значением, так и вещественным	Написать программу удаления информации о маршрутном такси с указанным номером

8	Написать программу ввода координаты точки трехмерного пространства (x, y, z) для целочисленных и вещественных типов данных	Написать программу сортировки координат точек по возрастанию координаты z.
9	Написать программу ввода информации по книге (автор, год изд., число стр., цена, название), где цена задается либо целочисленным, либо вещественным значением	Написать программу сортировки книг по возрастанию их цен
10	Написать программу ввода информации о продукте (наименование, цена, магазин, вес, габариты), где вес задается либо целочисленным, либо вещественным значением	Написать программу удаления учреждений с указанным весом из массива структур с объединением

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.
4. Выводы по полученным результатам работы программ.

Контрольные вопросы

1. Запишите объединение для хранения разнотипных данных.
2. Как задаются переменные на объединения?
3. Как задаются массивы объединений?
4. Запишите инициализацию объединения для хранения разнотипных данных.
5. Каким образом передаются объединения функциям?
6. Можно ли передавать функции массив объединений?

Лабораторная работа №10

ПЕРЕЧИСЛЕНИЯ И ТИПЫ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

Цель работы: изучить синтаксис и правила работы с перечислениями и типами, определяемые пользователем.

Теоретические сведения

В предыдущей лабораторной работе была использована переменная type_var, которая указывала номер используемой переменной, что не особенно удобно при написании сложных программ, где запомнить номер той или иной

переменной структуры довольно сложно. Проще если переменную type_var инициализировать специальными словами, например,

VT_NONE – тип переменной не определен;

VT_INT – целочисленный тип;

VT_FLOAT – вещественный тип;

VT_CHAR – символьный тип.

Этого можно достичь, если type_var определить как перечисляемый тип, который задается с помощью ключевого слова enum следующим образом:

enum tag_type {VT_NONE, VT_INT, VT_FLOAT, VT_CHAR};

Объявление перечисляемого типа выполняется подобно объявлению структуры или объединения:

enum tag_type type_var;

В результате для введенной переменной перечисляемого типа допустимо использование следующих операторов:

type_var = VT_INT; //переменная type_var принимает значение VT_INT

if(type_var == VT_NONE) // проверка

for(type_var = VT_NONE; type_var <= VT_CHAR; type_var++) //цикл

Анализ последнего оператора for показывает, что значения перечисляемого типа VT_NONE,…,VT_CHAR являются числами, которые имеют свои имена. Причем, VT_NONE = 0, VT_INT = 1, …, VT_CHAR = 3. В некоторых случаях использование имен удобнее использования цифр, т.к. их запоминать и ориентироваться в них проще, чем в числах.

Перепишем пример хранения разнородной информации с использованием перечисляемого типа, получим:

```
enum tag_type {VT_NONE, VT_INT, VT_FLOAT, VT_CHAR};
struct tag_var {
  union tag_value value;
  enum tag_type type_var;
};
int main()
{
  struct tag_var var[3];
  var[0].type_var = VT_INT;
  var[0].value.var_i = 10;
  var[1].type_var = VT_FLOAT;
  var[1].value.var_f = 2.3;
  var[2].type_var = VT_CHAR;
  var[2].value.var_ch = ‘d’;
  for(int i = 0;i < 3;i++)
  {
        switch(var[i].type_var)
        {
        case VT_INT:printf(“var = %d\n”,var[i].value.var_i);break;
        case VT_FLOAT:printf(“var = %f\n”,var[i].value.var_f);break;
        case VT_CHAR:printf(“var = %c\n”,var[i].value.var_ch);break;
```

```
            default: printf("Значение переменной не определено\n");
            }
  }
  return 0;
}
```

Из приведенного примера видно, что использование перечисляемого типа делает программу более понятной и удобной при программировании. Следует отметить, что если объявлены два перечисления

```
enum color {red, green, blue} clr;
enum color_type {clr_red, clr_green, clr_blue} type;
```

то числовые значения red, green, blue будут совпадать с соответствующими числовыми значениями vt_int, vt_float, vt_char. Это значит, что программный код if(clr == red) будет работать также как и if(clr == clr_red). Часто это не имеет принципиального значения, но, например, следующий оператор выдаст сообщение об ошибке:

```
switch(clr)
{
case red:printf("Красный цвет\n");break;
case green:printf("Зеленый цвет\n");break;
case clr_red:printf("Красный оттенок\n");break;
};
```

Ошибка возникнет из-за того, что значение red и значение clr_red равны одному числу – 0, а оператор switch не допускает такой ситуации. Для того чтобы избежать такой ситуации, при задании перечисляемого типа допускаются следующие варианты:

```
enum color_type {clr_red = 10, clr_green, clr_blue};
enum color_type {clr_red = 10, clr_green = 20, clr_blue = 30};
enum color_type {clr_red, clr_green = 20, clr_blue};
```

В первом случае значения clr_red = 10, clr_green = 11, clr_blue = 12. Во втором - clr_red = 10, clr_green = 20, clr_blue = 30. В третьем - clr_red = 0, clr_green = 20, clr_blue = 21. Как видно из полученных значений, величины могут инициализироваться при задании перечисления, а если они не объявлены, то принимают значение на единицу больше предыдущего значения.

Типы определяемые пользователем

Язык С допускает создание собственных типов данных на основе базовых, таких как int, float, struct, union, enum и др. Для этого используется ключевое слово typedef, за которым следует описание типа и его имя.

Рассмотрим действие оператора typedef на примере создания пользовательского типа с именем BYTE для объявления байтовых переменных, т.е. переменных, значения которых меняются в диапазоне от 0 до 255, и которые занимают один байт в памяти ЭВМ.

```
typedef unsigned char BYTE;
```

Здесь unsigned char – пользовательский тип, BYTE – имя введенного типа. После такого объявления слово BYTE можно использовать для определения переменных в программе:

```
BYTE var_byte;
```

Создание имени для существующего типа может показаться нецелесообразным, но иногда это имеет смысл. Так, применение оператора typedef повышает степень переносимости программного кода с одной платформы на другую. Например, тип, возвращаемый оператором sizeof, определен как size_t. Это связано с тем, что в разных реализациях языка C size_t определен или как unsigned int или как unsigned long для лучшей адаптации к той или иной операционной системы. Таким образом, составленный текст программы достаточно откомпилировать на соответствующей платформе и оператор sizeof автоматически «подстроится» под нее без переделки самой программы.

Кроме объявлений простых пользовательских типов оператор typedef можно использовать и при объявлении новых типов на основе структур. Например, удобно ввести тип COMPLEX для объявления переменных комплексных чисел. Для этого можно воспользоваться следующим кодом:

```
typedef struct complex {
  float real;
  float imag;
} COMPLEX;
```

и работать с комплексными числами

```
COMPLEX var_cmp1, var_cmp2, var_cmp3;
var_cmp1.real = 10;
var_cmp1.imag = 5.5;
var_cmp2.real = 6.3;
var_cmp2.imag = 2.5;
var_cmp3.real = var_cmp1.real + var_cmp2.real;
var_cmp3.imag= var_cmp1.imag + var_cmp2.imag;
```

Ключевое слово typedef можно использовать с любыми стандартными типами данных и типами объявленными ранее.

Задание на лабораторную работу

1. Усовершенствовать программу предыдущей лабораторной работы с использованием перечислений и типов определяемых пользователем
2. Сделать выводы по полученным результатам работы программы.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Текст программ.
3. Результаты действия программ.

4. Выводы по полученным результатам работы программ.

Контрольные вопросы

1. Запишите перечисление для разных цветов (красный, синий, зеленый).
2. Как задаются переменные на перечисления?
3. Как задаются массивы перечислений?
4. Задайте новый тип данных на структуру.
5. Для чего нужны типы определяемые пользователем?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дейтел, Харвин М. Как программировать на C++.: Пер. с англ. – 3-е изд. – М.: Бином, 2003.

2. Дэвис, Стефан Р. C++ «для чайников».: Пер. с англ. – 4-е изд.- М. [и др.]: Диалектика, 2001.

3. Культин, Никита. C/C++ в задачах и примерах.: учеб. пособие для вузов. – СПб.: BHV-Санкт-Петербург, 2001.

4. Литвиненко, Николай Аркадьевич. Технология программирования на C++. Начальный курс.: учеб. для вузов. – СПб.: БХВ-Петербург, 2005.

5. Мейн, Майкл. Структура данных и другие объекты в C++.: Пер с англ. – 2-е изд. – М.: Изд. дом «Вильямс», 2002.

Учебное издание

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Методические указания к лабораторным работам

Составитель НАМЕСТНИКОВ Сергей Михайлович

Редактор О. А. Семенова

Подписано в печать 09. 09. 2008. Формат 60×84/16.
Усл. печ. л. 1,65.
Тираж 60 экз. Заказ
Ульяновский государственный технический университет,
432027, Ульяновск, Сев. Венец, 32.
Типография УлГТУ, 432027, Ульяновск, Сев. Венец, 32.