Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«Кузбасский государственный технический университет»

Кафедра информационных и автоматизированных
производственных систем

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Методические указания к выполнению лабораторных работ
по дисциплине «Технология программирования» для студентов
специальности 230201 «Информационные системы
и технологии»

Составители В.А.Полетаев
Е. В. Башкирцева

Утверждены на заседании кафедры
Протокол № 7 от 05.04.2011

Рекомендованы к печати
учебно-методической комиссией
специальности 230201
Протокол № 287 от 07.04.2011

Электронная копия находится
в библиотеке ГУ КузГТУ

Кемерово 2011

Лабораторная работа № 1. Программирование арифметических операций

1. ЦЕЛЬ РАБОТЫ

Научиться создавать консольные проекты в интегрированной среде программирования и программировать простые арифметические операции.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Создание консольного проекта в среде

При запуске программы на экране монитора появится окно интегрированной среды. Для начала работы следует выбрать в меню «File» подменю «New» и в нем подменю «File…». В результате на экране появится диалоговое окно. В поле «*Name*» вводится название файла, в котором будет содержаться текст программы. После нажатия на кнопку «OK» появится окно для ввода текста программы.

Для компиляции созданного проекта, т.е. создания исполняемого кода exe-файла, следует выбрать меню «Compiler» и в нем подпункт «Compile». Компиляция и исполнение программы осуществляется путем нажатия комбинацией клавиш Ctrl+F5.

Представление данных в языке

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;
```

объявляет целочисленную переменную с именем arg. Основные базовые типы данных приведены в табл. 1.

Таблица 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 ***C*** при объявлении переменных различает большой и малый регистры.

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

Арифметические операции

В языке ***C*** 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 ***C***++ будет интерпретироваться как целочисленная величина, которая не может содержать дробной части. 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 ***C*** реализованы компактные операторы инкремента и декремента, т. е. увеличения и уменьшения значения переменной на 1 соответственно. Данные операторы могут быть записаны в следующем 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 $b = 11$.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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.

4. ВАРИАНТЫ ЗАДАНИЙ

№	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 параллелограмма
4.	Вещественный Целочисленный	1. $a_1b_1 + a_2b_2 + ... + a_{10}b_{10}$ 2. $f(x) = x^2 + b$, при $x = 1, 2, ..., 5$
5.	Вещественный Целочисленный	1. $(a + b)^3$ 2. $(a + b)^2$
6.	Вещественный Целочисленный	1. $(a - b)^2$ 2. Площадь круга
7.	Вещественный Целочисленный	1. Высоту параллелепипеда 2. Евклидовое расстояние между двумя точками
8.	Вещественный Целочисленный	1. Периметр прямоугольника 2. Объем параллелограмма
9.	Вещественный Целочисленный	1. 10% от числа 456 2. $a_1b_1 + a_2b_2 + ... + a_{10}b_{10}$

10.	Вещественный Целочисленный	1. Длину круга 2. $f(x) = x^2 + b$, при $x = 1, 2, ..., 5$

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.

6. 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 ***C***?
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()

1. ЦЕЛЬ РАБОТЫ

Изучить особенности работы директив препроцессора и функций printf() и scanf().

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Директивы препроцессора

Почти все программы на языке ***C*** используют специальные команды для компилятора, которые называются директивами. В общем случае директива – это указание компилятору языка ***C*** выполнить то или иное действие в момент компиляции програм-

мы. 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 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 графич. 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 графич. 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 математич. 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. Функция 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. 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. 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);
```

где 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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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.

4. ВАРИАНТЫ ЗАДАНИЙ

№	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 вычислить x^2, при $x = 1, 2, ..., 5$	Ввести два целочисленных значения и вывести их разность на экран монитора
5.	Задать константы $M_1, M_2, ..., M$ 5 и вычислить $M_1 + 2M_2 + 3M_3 + 4M_4 + 5M_5$	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

5. 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.

6. 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

1. ЦЕЛЬ РАБОТЫ

Изучить особенности использования условных операторов if и switch.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

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, с определением знака введенной переменной.

```
#include <stdio.h>
int main()
{ float x;
   printf(«Введите число:»);
   scanf(«%f»,&x);
   if(x < 0)
```

```
          printf(«Введенное число %f является отрицатель-
ным./n», x);
        if(x >= 0)
          printf(«Введенное число %f является неотрицатель-
ным./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 ***C*** имеются три логические опера-

ции: && – логическое И, || – логическое ИЛИ, ! – логическое НЕТ.

На основе этих трех логических операций можно сформировать более сложные условия. Например, если имеются три переменные expl, 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 || expl >= 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 ***C***.

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.

Синтаксис данного оператора следующий:

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

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. 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/П»);
   }
   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/П»);
   }
   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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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.

4. ВАРИАНТЫ ЗАДАНИЙ

1. Используя оператор if написать программу вычисления модуля введенного числа. Используя оператор switch написать программу перевода введенного символа от *a* до *f* в верхний регистр
2. Используя оператор if написать программу проверки попадания введенного числа в диапазон от -2 до 2. Используя оператор switch программу перевода введенного символа от *a* до *f* в нижний регистр.

3. Используя оператор if написать программу проверки не вхождения введенного числа в диапазон от 0 до 5. Используя оператор switch программу замены введенного символа от 0 до 9 соответствующим числом.

4. Используя оператор if написать программу проверки на положительность введенного числа. Используя оператор switch программу замены введенного числа от 0 до 9 соответствующим символом.

5. Используя оператор if написать программу проверки на отрицательность введенного числа. Используя оператор switch программу замены введенного числа от 0 до 5 соответствующим символом, а все другие значения заменять буквой *z*.

6. Используя оператор if написать программу определения знака введенного числа. Используя оператор switch программу замены введенного символа от 0 до 5 соответствующим числом, а все другие символы заменять числом -1.

7. Используя оператор if написать программу проверки попадания введенного числа в диапазон от -6 до -2. Используя оператор switch программу перевода введенного символа от *a* до *f* в верхний регистр, а другие символы заменять на *z*.

8. Используя оператор if написать программу проверки не вхождения введенного числа в диапазон от -5 до -1. Используя оператор switch программу перевода введенного символа от *A* до *F* в нижний регистр, а все другие символы заменять на *x*.

9. Используя оператор if написать программу вычисления суммы модулей двух введенных чисел. Используя оператор switch программу сравнения введенного числа со значениями 0, 4, 8, 9 и 30.

10. Используя оператор if написать программу вычисления $1 / a$ с проверкой $a \neq 0$. Используя оператор switch написать программу сравнения введенного символа с *a*, *s*, *d*, *j* и *e*.

5. 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.

6. 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».

Лабораторная работа № 4.
Операторы циклов языка С

1. ЦЕЛЬ РАБОТЫ

Изучить особенности использования операторов цикла while, for и do while.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

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=2 0, 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, изменив данный пример следующим образом:

```
int N=2 0, 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(«Bbi 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(«Bbi 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(«Bbi 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 значе-
ния
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 ***C*** – 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 ***C*** 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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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.

4. ВАРИАНТЫ ЗАДАНИЙ

№	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 $S = \sum_{i=-10}^{10} 1/i^3$, $i \neq 0$ с использованием оператора for	Написать программу определения максимального введенного числа из 5 чисел
9.	Вычислить $\sum_{i=-10}^{20}\sum_{j=0}^{10} 1/(i + j)^2$ 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

5. 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.

6. 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 ***C***.

Лабораторная работа № 5.
Массивы

1. ЦЕЛЬ РАБОТЫ

Изучить базовые операции работы с одномерными и двумерными массивами.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

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 од-

ного 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 ***C*** 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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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.

4. ВАРИАНТЫ ЗАДАНИЙ

№	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 $f(x, y) = x + y$, $0 < x < 20$, $0 < y < 10$ 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 5 × 4 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 $f(x, y) = 1/(x + y)$, $0 < x < 30$, $1 < y < 20$ 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 экран	Написать программу ввода в массив 6 × 3 элемента чисел и поиска в нем минимального значения
5.	Написать программу ввода в массив 10 чисел и поиска в нем минимального значения	Занести в массив значения функции $f(x, y) = (x + y)^2$, $0 < x < 5$, $0 < y < 3$ 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 6 × 5 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 $f(x, y) = 1/\left((x - y)^2 + 1\right)$, $0 < x < 5$, $0 < y < 10$ и вывести его на экран
8.	Написать программу ввода в массив 5 чисел и вычисления произведения элементов полученного массива	Написать программу ввода в массив 3 × 3 элемента чисел и вычисления произведения элементов полученного массива
9.	Записать в массив значения функции $f(x) = 1/\tilde{o} + b$, при x = 1, 2,..., 50 и вывести его на экран	Занести в массив значения функции $f(x, y) = \tilde{o} - y$, $0 < x < 20$, $0 < y < 10$ 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 4 × 4 элементов чисел и поиска в нем модуля максимального значения

5. 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.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом задаются одномерные массивы в языке ***C***?
2. Запишите массив целых чисел с начальными значениями 1, 2 и 3.
3. Каким образом задаются двумерные массивы в языке ***C***?
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 C

1. ЦЕЛЬ РАБОТЫ

Изучить базовые операции работы со строками.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

В языке ***C*** 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 ***C***++ «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 ***C***++ 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().

```
#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. Далее показано использование 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 ***C*** 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 ***C*** 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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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

4. ВАРИАНТЫ ЗАДАНИЙ

1. Написать программу:
   - поэлементного копирования строки «Hello World» в другой символьный массив;
   - 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 заглавные;
   - удаления букв «н» из введенной строки.
3. Написать программу:
   - подсчета букв «е» во введенной строке;
   - добавления слова «hello» после первого слова введенной строки.
4. Написать программу:
   - удаления букв «о» из введенного слова;
   - сравнения двух строк с помощью функции strcmp().
5. Написать программу:
   - добавления пробела после каждой буквы «а» введенной строки;
   - замены во веденной строке заглавных букв «О» на малые.
6. Написать программу:
   - подсчета числа слов в строке;
   - подсчета букв «и» во введенной строке.
7. Написать программу:

- выделения первого слова из введенной строки и отображение его на экране;
- удаления всех пробелов из введенной строки.

8. Написать программу:

- выделения последнего слова из введенной строки и отображение его на экране;
- копирования первой половины введенной строки в другую строку.

9. Написать программу:

- вывода введенного слова задом наперед;
- сравнения первых половин двух введенных строк.

10. Написать программу:

- удаления последнего слова из строки;
- замещения первой половины строки второй, а второй – первой.

5. 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.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как задаются строки в программе на языке ***C***?
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

1. ЦЕЛЬ РАБОТЫ

Научиться задавать свои функции и изучить правила работы с ними.

2. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

В ранее рассмотренных примерах неоднократно использовались различные функции подключаемых библиотек. Вместе с тем существующих функций языка ***C*** 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. Тело функции – это набор операторов, которые выполняются при ее вызове. Следующий пример показывает правило определения пользовательских функций.

```
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 ***C*** при создании своих функций можно пользоваться правилом: сначала задаются объявления функции, а затем их реализации.

Язык ***C*** позволяет задавать функции с одинаковыми именами, но разными типами входных аргументов, т. е. функция будет перегружена. Следующий пример демонстрирует удобство использования таких функций при их вызове.

```
#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 ***C*** 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.

```
#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.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

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

4. ВАРИАНТЫ ЗАДАНИЙ

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

5. 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.

6. 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.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

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

2. Дэвис, Р. C++ «для чайников».: пер. с англ. – 4-е изд. / Р. Дэвис, Р. Стефан. – М.: Диалектика, 2001.

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

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

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

Дополнительная литература

6. Блэк, Ю. Сети ЭВМ: Протоколы, стандарты, интерфейсы: пер. с англ. / Ю. Блэк. – М.: Мир, 1990. – 506 с.

7. Емельянов, Г. А. Передача дискретной информации: учеб. для вузов / Г. А. Емельянов, В. О. Шварцман. – М.: Радио и связь, 1982. – 240 с.

8. Ирвин, Дж. Передача данных в сетях: инженерный подход: пер. с англ. / Дж. Ирвин, Д. Харль. – СПб.: БХВ-Петербург, 2003. – 448 с.

СОДЕРЖАНИЕ

Составитель
Вадим Алексеевич Полетаев
Елена Владимировна Башкирцева

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Методические указания к выполнению лабораторных работ
по дисциплине «Технология программирования» для студентов
специальности 230201 «Информационные системы
и технологии»

Печатается в авторской редакции

Подписано в печать 05.05.2011. Формат 60×84/16.
Бумага офсетная. Отпечатано на ризографе. Уч.-изд. л. 3,1.
Тираж 36 экз. Заказ
ГУ КузГТУ. 650000, Кемерово, ул. Весенняя, 28.
Типография ГУ КузГТУ. 650000, Кемерово, ул. Д. Бедного, 4 А.