|  |
| --- |
| **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ** |
|  |
|  |
|  |
| **Практическое занятие №2.**  **Тема: Взаимодействие функций в программах** |
|  |
|  |
|  |

**Цель:** получение практических навыков проектирования взаимодействия функций в программах с использованием локальных и глобальных переменных.

**Основные теоретические положения**

**Общие сведения о функциях и области видимости переменных**

Функция – это логически самостоятельная именованная часть программы, в которую может передаваться любое количество значений аргументов (или не передаваться), а функция может возвращать (или не возвращать) значение.

***Объявление функции***

При объявлении функции указывается (слева направо):

* тип возвращаемого функцией значения;
* имя функции;
* в круглых скобках – типы и имена параметров (переменных).

Как и всякое объявление, конструкция заканчивается точкой с запятой. Пример.

**int buf (int a);**

Данное объявление означает, что функция с именем **buf** будет возвращать целочисленный значение. В скобках указан один целочисленный параметр **а**, значение которого следует передать в функцию при её использовании.

***Определение функции***

Определение функции – это описание операций, которые выполняются в ее рамках. Если объявление функции должно предшествовать её использованию, то определение функции может быть сделано в любом месте программы (за исключением случая «функция в функции»).

Пример функции, которая выполняет сложение двух чисел и возвращает сумму:

**int summa(int x, int y)**

**{**

**int z;**

**z=x+y;**

**return z;**

**}**

***Вызов функции***

Для вызова функции надо указать 1) ее имя и 2) в скобках – список аргументов в соответствии с прототипом. Если функция возвращает значение, и оно дальше будет использоваться в программе, то его необходимо присвоить какой-либо переменной. Например:

**w = delta (x,y);**

В данном примере вызывается функция **delta**, которой передаются два аргумента. После выполнения функция возвращает значение, которое присвоится переменной **w**.

Любая программа на Си++ начинает выполняться с главной функции **main( )**. Обычно из нее вызываются другие функции. В общем случае любая функция может быть вызвана из любой. Одна из функций становится вызывающей и временно передает управление вызываемой функции, которая, выполнив определенные операции, возвращает управление вызывающей.

Далее приведен пример программы, в которой используются две функции. По условию требуется вычислить z = x². Возведение числа в квадрат оформляется в виде отдельной функции.

#**include<iostream>**

**using namespace std;**

**int square (int a);** // объявление функции square()

**int main ( )**

**{**

**int z;**

**int x = 3;**

**z = square (x);**

**cout<< ” square =” << z << ”\n”;**

**return 0;**

**}**

// определение функции square

**int square (int a)**

**{**

**int b;**

**b = a\*a;**

**return b;** // оператор возврата управления

**}**

**Локальные и глобальные переменные**

Переменные, которые объявлены внутри функции, называются внутренними переменными или **локальными**. Эти переменные могут использоваться только в тех функциях, где они объявлены, и никакие другие функции доступа к ним не имеют.

Каждая локальная переменная функции возникает только в момент обращения к этой функции и исчезает после выхода из нее. Такие переменные называются автоматическими, т. к. они образуются и исчезают одновременно с входом в функцию и выходом из нее; они не сохраняют своих значений от вызова к вызову.

Чтобы переменная не только сохраняла свои значения в течение работы программы, но и чтобы ею могли пользоваться другие функции, она должна быть не локальной, а глобальной переменной. Глобальные (внешние) переменные доступны повсеместно. Их можно использовать в любом месте программы, в любой функции. Кроме того, поскольку внешние переменные существуют постоянно, а не возникают и не исчезают на период выполнения функции, свои значения они сохраняют и после возврата из функций, их установивших.

Внешняя переменная должна быть объявлена вне тела любой функции и только один раз; в этом случае ей будет выделена память. Обычно внешние переменные объявляются в начале программы, после директив препроцессора**.**

Поскольку внешние переменные доступны всюду, их можно использовать в качестве связующих посредников между функциями как альтернативу связей через аргументы и возвращаемые значения.

Область видимости переменной (функции) – это часть программы, в которой этой переменной можно пользоваться. Для локальных переменных – это функция, где они объявлены. Область действия внешней переменной простирается от точки программы, где она объявлена, до конца файла.

Рассмотрим пример, в котором две функции взаимодействуют через глобальные переменные: в функции **main()** задаётся значение переменной, а в функции **square()** вычисляется её квадрат. Результат выводится в главной функции.

**#include<iostream>**

**using namespace std;**

**void square();** // объявление функции

**int a, b;** // объявление внешних переменных

**int main()**

**{**

**a = 3;**

**square();**

**cout << b <<endl;**

**return 0;**

**}**

**void square()**

**{**

**b =a\*a;**

**}**

Т.к. связь между функциями осуществляется через внешние переменные, то в функцию **square( )** ничего не передаётся и она ничего не возвращает. В главной функции **main( )** присваивается значение переменной **а** и вызывается функция **square( )**, которая вычисляет **а2** и присваивает его внешней переменной **b**, значение которой выводится на экран в главной функции.

**Типы данных**

C++ допускает, чтобы основные типы данных **char,int** и **double** имели предшествующие им модификаторы. Модификатор используется для изменения значения базового типа, чтобы он более точно соответствовал потребностям различных ситуаций.

Модификаторы типа данных:

* **signed**
* **unsigned**
* **long**
* **short**

Все допустимые комбинации базовых типов и модификаторов для 32-разраядной среды приведены в таблице 1.

Таблица 1

| **Тип** | **Длина в битах** | **Диапазон** |
| --- | --- | --- |
| char | 8 | от -128 до 127 |
| unsigned char | 8 | от 0 до 255 |
| signed char | 8 | от -128 до 127 |
| int | 32 | от -2147483648 до 2147483647 |
| unsigned int | 32 | от 0 до 4294967295 |
| signed int | 32 | от -2147483648 до 2147483647 |
| short int | 16 | от -32768 до 32767 |
| unsigned short int | 16 | от 0 до 65535 |
| signed short int | 16 | от -32768 до 32767 |
| long int | 32 | от -2147483648 до 2147483647 |
| unsigned long int | 32 | от 0 до 4294967295 |
| signed long int | 32 | от -2147483648 до 2147483647 |
| float | 32 | от 3.4e-38 до 3.4е+38 |
| double | 64 | от 1.7е-308 до 1.7е+308 |
| long double | 80 | от 3.4е-4932 до 1.1e+4932 |
| bool | 8 | true / false |
|  | | |

**Оператор sizeof**

Размер памяти, который отводится под конкретный тип данных, в общем случае зависит от марки компьютера. Чтобы определить, сколько байт памяти выделено под ту или иную переменную, в С++ используется оператор **sizeof()**. В следующем примере показано, в каких вариантах можно применять данный оператор.

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int integer;**

**long longer;**

**long\* pl = &longer;**

**float drob;**

**cout << "INT: "<< sizeof(integer) <<endl;**

**cout << "LONG: "<< sizeof(longer) <<endl;**

**cout << "FLOAT: "<< sizeof(drob) <<endl;**

**cout << "POINTER: "<< sizeof(pl) <<endl;**

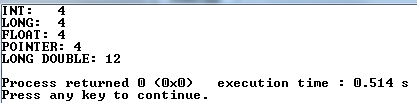
**cout <<"LONG DOUBLE:"<<sizeof(long double)<<endl;**

**return 0;**

**}**

Предпоследняя строчка программы показывает, что в операторе **sizeof()** внутри скобок можно указывать не только программные объекты, но и типы данных.

Результат работы программы представлен ниже.



**Преобразование типов данных**

Преобразование значения переменной одного типа в значение другого типа называется приведение типа и бывает явным и неявным:

* при явном приведении перед выражением следует указать в круглых скобках имя типа, к которому необходимо преобразовать исходное значение;
* при неявном приведении преобразование происходит автоматически.

Пример явного приведения типа.

**int x = 5;**

**double y = 15.3;**

**cout<<(float)(x/y);**

В данном примере частное от деления x/yпринудительно приводится к типу данных **float**.

Пример неявного приведения типа.

**int x = 5;**

**double y = 15.3;**

**y = x;** // происходит неявное приведение типа int к double

**x = y;** // происходит неявное приведение типа double к int

Если в выражении встречаются два операнда разных типов, то действуют следующие правила:

* все операнды преобразуются к типу наибольшего операнда. Процесс такого преобразования называется расширением типа;
* все типы char и short int преобразуются к типу int. Процесс такого преобразования называется целочисленным;
* если один из операндов имеет тип double, тогда любой другой операнд приводится к типу double. Даже, в случае с типом char, происходит приведение к типу double.

Ниже приведены примеры автоматического преобразования между типами char и int.

**char c;**

**int d;**

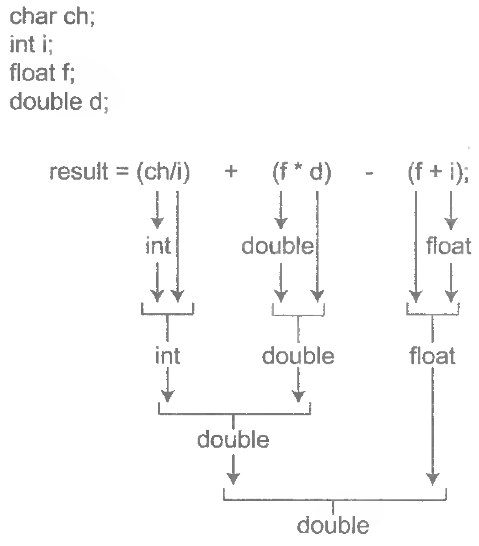
**c = 'A';**

**d = c; // d = 65**

**d = 67;**

**c = d; // c = 'C'**

Следующий пример демонстрирует, как производится преобразование типов данных в выражениях.



**Понятие указателя**

Указатель – это переменная, содержащая адрес другой переменной. При объявлении, перед такого рода переменной ставится знак «\*». Примеры:

**int\* ip;** // **ip -** указатель на переменную типа **int**

**char\* b;** // **b -** указатель на тип **char**

Для получения адреса используется унарный оператор **&**(амперсанд).

Так, если

**int a;**

**int \*ptr;**

то следующая операция правильна:

**ptr = &a;**

важно, чтобы типы данных указателя и переменной совпадали.

Для обращения к данным через указатель используется оператор косвенного доступа – «**\***» (звездочка). Он ещё называется «обращением по адресу» или «разадресацией». Если, например

**int x;**

**int \*pt;**

**. . .**

то запись

**x = \*pt;**

означает, что переменной **x** присваивается значение, которое хранится по адресу **pt**.

Следующий простой пример демонстрирует работу с указателем.

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**double var = 2.72;**

**double\* ptr = &var;** // инициализация указателя

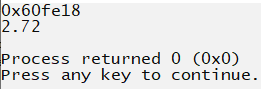
**cout << ptr << endl;** // вывод значения указателя (адрес)

**cout << \*ptr << endl;** // вывод значения переменной

**return 0;**

**}**

Вид экрана:



**Ссылки**

Ссылка является элементом, родственным указателю и также как и указатель содержит адрес памяти. Отличие ссылки от указателя состоит в том, что при использовании ссылки автоматически происходит её разадресация, тем самым обеспечивается прямой доступ к данным, на которые указывает ссылка. Рассмотрим это отличие на примере.

В первом случае в отдельную функцию передаётся адрес как указатель.

**#include <iostream>**

**using namespace std;**

**void pointin(int \*n)** //объявление параметра-указателя

**{**

**\*n=\*n\*\*n;** //используется операция «обращение по адресу»

**}**

**int main()**

**{**

**setlocale(0,"");**

**int i = 15;**

**pointin(&i);**

**cout << "Квадрат значения i: " << i << '\n';**

**return 0;**

**}**

В функции **pointin(int \*n)** переменная **n** – это указатель, его содержимым является адрес. Поэтому внутри функции, чтобы получить доступ к данным, указатель разадресовывается (выполняется операция «обращение по адресу»).

Во втором случае в отдельную функцию передаётся адрес как ссылка.

**#include <iostream>**

**using namespace std;**

**void refin(int &n)** // объявление параметра-ссылки

**{**

// в следующей команде знак \* не требуется,

**n=n\*n;** // ссылка автоматически обеспечивает

// доступ к данным

**}**

**int main()**

**{**

**setlocale(0,"");**

**int i = 15;**

**refin(i);**

**cout << "Квадрат значения i: " << i << '\n';**

**return 0;**

**}**

Здесь в функцию **refin(int &n)** как и в функцию **pointin(int \*n)** тоже передаётся адрес, но посредством ссылки. Поэтому в операциях она используется как обыкновенная переменная.

Следует обратить внимание на то, что при использовании ссылки не только обращение по адресу, но и обратная операция – взятие адреса – также происходит автоматически. Поэтому при вызове функции **refin()** в качестве аргумента у неё указывается не адрес переменной, а сама переменная: **refin(i).**

**Понятие массива**

Массив относится к составному типу данных. Он представляет собой набор простых однотипных данных. Они расположены в памяти вплотную друг за другом. При объявлении массива после его имени в скобках указывается количество элементов массива.

Объявляют массивы так:

**float mass[20];** // массив из двадцати дробных чисел

**int m[100];** // сто - элементный целочисленный массив

**char s[25];** // массив из двадцати пяти символов

При использовании массива в программе число в скобках указывает *номер элемента массива* и называется *индексом*. Индекс всегда начинается с нуля.

Ниже показан пример, в котором объявляется символьный массив, присваиваются значения элементам массива и некоторые из них выводятся на экран.

**#include<iostream>**

**using namespace std;**

**int main( )**

**{**

**char mass[6];**

**int k, l, m;**

**cout<<”Insert 3 numbers(0-5)”;**

**cin>>k>> l>>m;**

**mass[0]=’e’;**

**mass[1]=’o’;**

**mass[2]=’n’;**

**mass[3]=’p’;**

**mass[4]=’r’;**

**mass[5]=’d’;**

**cout<<mass[k]<<mass[l]<<mass[m];**

**return 0;**

**}**

В данном примере объявлен символьный массив из шести элементов **mass[6]** и три целочисленные переменные **k, l, m**. Сначала на экран выводится сообщение, которое информирует пользователя о том, что следует ввести три целых числа в диапазоне от нуля до пяти. Эти числа будут использованы в качестве номеров элементов массива. После того как числа введены происходит инициализация (присваивание начальных значений) элементов массива символьными константами. Затем на экран выводятся три из них, номера которых пользователь ввел с клавиатуры.

***Обеспечение доступа к массиву из функции***

Чтобы передать массив в функцию в качестве параметра, следует при вызове функции в скобках указать только имя массива. В следующем примере в главной функции задаётся массив и передаётся в функцию **summa()**, которая определяет сумму его элементов.

**#include <iostream>**

**using namespace std;**

**void summa(float a[10])**

**{**

**float s=0;**

**int i;**

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

**s=s+a[i];**

**cout << s;**

**}**

**int main()**

**{**

**float m[10];**

**int i;**

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

**{**

**cout << "Insert element: ";**

**cin >> m[i];**

**}**

**summa(m);**

**return 0;**

**}**

**Практические задания**

*Выберите один из вариантов задания.*

**Вариант А.**

Каждое из первых двух заданий следует оформить в виде отдельной функции. *В главной функции задавать исходные данные и выводить результаты на экран.*

Программу разработать с функциями в двух версиях. В первой версии глобальные переменные не использовать. Во второй версии взаимодействие функций организовать через глобальные переменные (общую область памяти).

**Задание 1.** Задать данные указанного типа (Приложение А), вычислить их сумму и определить размер отведённой для неё памяти. Вывести на экран значение суммы.

**Задание 2.** Заполнить пятиэлементный массив с клавиатуры и определить *Вычисляемый показатель* (Приложение А).

**Задание 3.** Объединить обе версии в одну программу. Разработать меню для управления программой, в котором по желанию пользователя обеспечить:

1. выполнение любого из заданий для любой из версий и

в любом порядке;

1. выход из программы.

**Вариант В.**

Каждое из двух заданий следует оформить в виде отдельной функции. *В главной функции задавать исходные данные и выводить результаты на экран.*

Программу разработать с функциями в двух версиях. В первой версии глобальные переменные не использовать. Во второй версии взаимодействие функций организовать через глобальные переменные (общую область памяти).

**Задание 1.** Задать данные указанного типа (Приложение А), вычислить их сумму и определить размер отведённой для неё памяти. Вывести на экран значение суммы.

**Задание 2.** Заполнить пятиэлементный массив с клавиатуры и определить *Вычисляемый показатель* (Приложение А).

**Вариант С.**

**Задание 1.** Задать данные указанного типа (Приложение А), вычислить их сумму и определить размер отведённой для неё памяти. Вывести на экран значение суммы.

**Задание 2.** Заполнить пятиэлементный массив с клавиатуры и определить *Вычисляемый показатель* (Приложение А).

Каждое задание оформить в виде отдельной функции.

**Приложение А**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№**  **варианта** | **Задание 1** | | **Задание 2** | |
| **Типы данных слагаемых** | **Тип данных суммы** | **Тип данных массива** | **Вычисляемый показатель** |
|  | int, double | long int | float | Наименьшая дробная часть элемента |
|  | unsigned char, float | int | double | Сумма положительных элементов массива |
|  | unsigned char, char | float | char | Последняя буква по алфавиту |
|  | double, long int | float | signed long int | Количество положительных чисел |
|  | unsigned int, int | char | long double | Максимальное значение |
|  | short int, float | Long int | unsigned char | Наименьшее значение |
|  | char, double | float | long int | Наибольший остаток по модулю 8 |
|  | long double, long int | int | bool | Количество значений true |
|  | signed long int, char | float | double | Наименьший элемент |
|  | int, char | char | unsigned int | Наименьший остаток по модулю 5 |
|  | unsigned char, float | signed long int | double | Наибольшая дробная часть элемента |
|  | double, float | int | char | Количество букв ‘a’ |
|  | char, unsigned short int | float | char | Первая буква по алфавиту |
|  | int, char | double | bool | Количество элементов со значением false |
|  | unsigned long int, int | double | signed long int | Наименьший элемент |
|  | char, int | float | float | Сумма элементов массива |
|  | float, char | unsigned int | double | Наибольшая целая часть элемента |
|  | int, signed char | float | unsigned long int | Количество нулевых элементов |
|  | float, char | char | int | Количество ненулевых элементов |
|  | float, short int | signed long int | bool | Количество элементов со значением false |
|  | float, unsigned char | signed int | double | Наименьшая целая часть элемента |
|  | double, float | unsigned int | char | Количество букв ‘е’ |
|  | char, unsigned short int | double | char | Последняя буква по алфавиту |
|  | int, char | float | bool | Количество элементов со значением true |
|  | long int, float | char | signed long int | Сумма элементов массива |
|  | char, int | double | float | Наименьший элемент массива |
|  | double, char | unsigned int | double | Наибольшая дробная часть элемента |
|  | int, signed char | double | unsigned int | Количество ненулевых элементов |
|  | float, double | int | float | Наибольшее число |
|  | long, float | unsigned int | char | Количество букв ‘c’ |