В этом модуле мы приступаем к подробному рассмотрению функций. Функции — это строительные блоки C++, а потому без полного их понимания невозможно стать успешным программистом на C++. Вы узнаете, как создать функцию и передать ей аргументы, а также о локальных и глобальных переменных, о прототипах функций, областях видимости функций и рекурсии.

Основы

Функция — это подпрограмма, которая содержит один или несколько операторов и выполняет определенную задачу. Каждая из приведенных ранее в книге программ содержала одну функцию `main()`. Функции считаются строительными блоками C++, поскольку программа в C++, как правило, представляет собой группу функций. Все действия программы реализованы в виде функций. Функция содержит инструкции, которые составляют исполняемую часть программы.

Очень простые программы (как представленные здесь до сих пор) содержат только одну функцию `main()`, в то время как большинство других включают несколько функций. Сложные программы насчитывают сотни функций.

ВАЖНО!

5.1 Синтаксис функций в C++

Все функции в C++ имеют следующий синтаксис:

```
тип_возвращаемого_значения имя_функции(список_параметров)
{
   // тело функции
}
```

С помощью *типа_возвращаемого_значения* указывается тип значения, возвращаемого функцией. Это может быть практически любой тип, за исключением массива. Если функция не возвращает никакого значения, необходимо указать тип `void`. Если функция действительно возвращает значение, оно должно иметь тип, совместимый с указанным в определении функции. Каждая функция имеет *имя_функции*. В качестве имени можно использовать любое допустимое значение, которое еще не было задействовано в программе. После имени функции в круглых скобках указывается *список_параметров*, который представляет собой последовательность пар (состоящих из типа данных и имени), разделенных запятыми. Параметры — это переменные, которые получают значение аргументов, передаваемых функции при вызове. Если функция не имеет параметров, то *список_параметров* отсутствует, то есть круглые скобки остаются пустыми.

В фигурные скобки заключено тело функции. Тело функции составляют инструкции C++, которые определяют действия функции. Функция завершается

(и управление передается вызывающей процедуре) после закрывающей фигурной скобки.

ВАЖНО!

5.2 Создание функций

Создать функцию очень просто. Поскольку все функции используют один и тот же синтаксис, их структура подобна структуре функции `main()`, с которой вам уже приходилось иметь дело. Начнем с простого примера, который содержит две функции: `main()` и `myfunc()`. До выполнения этой программы (или чтения последующего описания) внимательно изучите ее код и попытайтесь предугадать, что она должна отобразить на экране:

```
// Эта программа содержит две функции: main() и myfunc()

#include <iostream>
using namespace std;

void myfunc(); // Прототип функции myfunc()  <-- Прототип функции myfunc()

int main()
{

   cout << "In main()\n";

   myfunc(); // Вызываем функцию myfunc()

   cout << "Back in main()\n";

   return  0;
}

// Определение функции
void myfunc()
{
   cout << "Inside myfunc()\n";
}
```

Функция myfunc()

Программа работает следующим образом. Сначала вызывается функция `main()` и выполняется ее первая инструкция `cout`. Затем из функции `main()` вызывается функция `myfunc()`. Обратите внимание на то, как этот вызов реализуется в программе: указывается имя функции, `myfunc`, за которым следует пара круглых скобок и точка с запятой. Вызов любой функции представляет собой инструкцию C++ и поэтому должен завершаться точкой с запятой. Затем функция `myfunc()` выполняет свою единственную инструкцию `cout` и возвращает управление функции `main()`, причем той строке кода, которая расположена непосредственно за вызовом функции. Наконец, функция `main()` выполняет свою вторую

инструкцию `cout`, которая завершает всю программу. На экране мы должны увидеть такие результаты:

```
In main()
Inside myfunc()
Back in main()
```

То, как организован вызов функции `myfunc()` и ее возврат, представляет собой конкретный вариант общего процесса, который применяется ко всем функциям. Чтобы вызвать функцию, достаточно указать ее имя с парой круглых скобок. После вызова управление программой переходит к функции. Выполнение функции продолжается до обнаружения закрывающей фигурной скобки. Когда функция завершается, управление передается инициатору ее вызова, той строке кода, которая следует непосредственно за вызовом.

В этой программе обратите внимание на следующий код:

```
void myfunc(); // Прототип функции myfunc()
```

Как отмечено в комментарии, это — *прототип* функции `myfunc()`. Подробнее прототипы будут рассмотрены ниже, но без кратких пояснений здесь не обойтись. Прототип функции объявляет функцию до ее определения. Прототип позволяет компилятору узнать тип значения, возвращаемого этой функцией, а также количество и тип параметров, которые она может иметь. Компилятору нужно знать эту информацию до первого вызова функции. Поэтому прототип располагается до функции `main()`. Единственная функция, которой не требуется прототип, — это `main()`, поскольку она встроена в язык C++.

Ключевое слово `void`, которое предваряет как прототип, так и определение функции `myfunc()`, формально заявляет о том, что функция `myfunc()` не возвращает никакого значения. В C++ функции, не возвращающие значений, должны объявляться с использованием ключевого слова `void`.

ВАЖНО!

5.3 Аргументы функции

Функции можно передать одно или несколько значений. Значение, передаваемое функции, называется *аргументом*. Аргументы представляют собой средство передачи в функцию информации.

При создании функции, которая принимает один или несколько аргументов, необходимо объявить переменные, которые получат значения этих аргументов. Эти переменные называются *параметрами функции*. Рассмотрим пример определения функции с именем `box()`, которая вычисляет объем параллелепипеда и отображает полученный результат:

```
void box(int length, int width, int height)
{
   cout << "volume of box is " << length * width * height << "\n";
}
```

При каждом вызове функции `box()` будет вычислен объем параллелепипеда путем перемножения значений, переданных ее параметрам `length`, `width` и `height`. Обратите внимание, что эти параметры объявлены в виде списка, заключенного в круглые скобки, расположенные после имени функции. Объявление каждого параметра отделяется от следующего запятой. Так объявляются параметры для всех функций (если они их используют).

При вызове функции необходимо указать три аргумента. Например:

```
box(7, 20, 4);
box(50, 3, 2);
box(8, 6, 9);
```

Значения, заданные в круглых скобках, являются *аргументами*, передаваемыми функции `box()`. Каждое из этих значений копируется в соответствующий параметр. Так, при первом вызове функции `box()` число `7` копируется в параметр `length`, `20` — в параметр `width`, а `4` — в параметр `height`. При выполнении второго вызова `50` копируется в параметр `length`, `3` — в параметр `width`, а `2` — в параметр `height`. Во время третьего вызова в параметр `length` будет скопировано число `8`, в параметр `width` — число `6` и в параметр `height` — число `9`.

Ниже приведена программа, которая показывает, как работает функция `box()`:

```
// Простая программа для демонстрации функции box()

#include <iostream>
using namespace std;

void box(int length, int width, int height); // Прототип функции box()

int main()
{
   box(7, 20, 4);
   box(50, 3, 2);
   box(8, 6, 9);

   return  0;
}

// Вычисление объема параллелепипеда
void box(int length, int width, int height)
{
   cout << "volume of box is " << length * width * height << "\n";
}
```

Передаем аргументы функции box()

Параметры принимают значения аргументов, переданных функции box()

При выполнении программа выводит следующие результаты:

```
volume of box is 560
volume of box is 300
volume of box is 432
```

ПРИМЕЧАНИЕ

Помните, что термин *аргумент* относится к значению, которое используется при вызове функции. Переменная, которая принимает значение аргумента, называется *параметром*. Функции, принимающие аргументы, называются *параметризованными*.

Проверьте себя

1. Как выполняется программа при вызове функции?
2. Чем аргумент отличается от параметра?
3. Если функция использует параметр, то где он объявляется?

ВАЖНО!

5.4 Возвращение значений из функций

В предыдущих примерах возврат из функции к инициатору ее вызова происходил при обнаружении закрывающей фигурной скобки. Но это приемлемо не для всех функций. Иногда необходимо более гибкое средство управления возвратом значения — инструкция `return`.

Инструкция `return` имеет две формы применения. Первая позволяет возвращать значение, а вторая — нет. Начнем со второй версии. Если тип возвращаемого значения определяется ключевым словом `void` (то есть функция не возвращает значения вообще), то для выхода из функции достаточно использовать такую форму инструкции `return`:

```
return;
```

Ответы

1. После вызова управление программой переходит к функции. Когда выполнение функции завершается, управление передается инициатору ее вызова, той строке кода, которая расположена непосредственно за вызовом функции.
2. Аргумент — это значение, передаваемое функции. Параметр — это переменная, которая принимает это значение.
3. В круглых скобках, стоящих после имени функции.

При обнаружении return управление программой немедленно передается инициатору ее вызова. Любой код в функции, расположенный за этой инструкцией, игнорируется. Рассмотрим следующую программу:

```
// Использование инструкции return

#include <iostream>
using namespace std;
void f();

int main()
{
   cout << "Before call\n";

   f();

   cout << "After call\n";

   return  0;
}

// Определение void-функции, которая использует инструкцию return
void f()
{
   cout << "Inside f()\n";

   return ; // возвращение к инициатору вызова

   cout <<"This won't display.\n";
}
```

Немедленное возвращение к инициатору вызова без выполнения следующей инструкции cout

Ниже показано, как выглядят результаты выполнения этой программы:

```
Before call
Inside f()
After call
```

Как мы видим из результатов выполнения этой программы, функция f() возвращается в функцию main() сразу после обнаружения return. Следующая после return инструкция cout никогда не выполнится.

Рассмотрим более практичный пример использования return. Функция power(), задействованная в представленной ниже программе, отображает результат возведения целочисленного значения в положительную целую степень. Если показатель степени отрицательный, return немедленно завершает эту функцию еще до попытки вычислить результат:

```
#include <iostream>
using namespace std;

void power(int base, int exp);
```

```
int main()
{
   power(10, 2);
   power(10, -2);

   return  0;
}

// Возводим целое число в положительную степень
void power(int base, int exp)
{
   int i;

   if(exp < 0) return; // Отрицательные показатели степени не обрабатываются


   i = 1;

   for( ; exp; exp--) i = base * i;
   cout << "The answer is: " << i;
}
```

Результат выполнения этой программы таков:

```
The answer is: 100
```

Если значение параметра `exp` отрицательное, как при втором вызове функции `power()`, вся оставшаяся часть функции опускается.

Функция может содержать несколько инструкций `return`. В этом случае возврат значения будет выполнен при обнаружении одной из них. Например, следующий фрагмент кода вполне допустим:

```
void f()
{
   // ...

   switch(c) {
      case 'a': return;
      case 'b': // ...
      case 'c': return;
   }
   if(count < 100) return;
   // ...
}
```

Но следует иметь в виду, что излишне большое количество инструкций `return` может ухудшить ясность алгоритма и ввести в заблуждение тех, кто будет в нем разбираться. Несколько таких инструкций стоит использовать лишь в том случае, если они способствуют ясности функции.

Возврат значений

Функция может возвращать значение инициатору своего вызова. Таким образом, возвращаемое функцией значение — это средство получения информации из нее. Чтобы вернуть значение, используйте вторую форму инструкции `return`:

```
return значение;
```

Эту форму `return` можно применять только с теми функциями, которые возвращают значение, отличное от `void`.

Функция, возвращающая значение, должна определить его тип: он должен быть совместимым с типом данных, используемых в инструкции `return`. В противном случае неминуема ошибка во время компиляции программы. Функция может возвращать данные любого допустимого в C++ типа, за исключением массива.

Чтобы проиллюстрировать процесс возврата функцией значений, перепишем уже известную вам функцию `box()`. В этой версии `box()` возвращает объем параллелепипеда. Обратите внимание на расположение функции с правой стороны от символа операции присваивания. Это означает, что переменной, которая находится с левой стороны, присваивается значение, возвращаемое функцией:

```
// Возврат значения

#include <iostream>
using namespace std;

int box(int length, int width, int height); // функция возвращает объем

int main()
{
   int answer;

   answer = box(10, 11, 3); // возвращаемое значение присваивается переменной
   cout << "The volume is " << answer;

   return  0;
}

// Эта функция возвращает значение
int box(int length, int width, int height)
{
   return  length * width * height ;
}
```

Значение, возвращаемое функцией box(), присваивается переменной answer

Возвращается объем

Ниже показан результат выполнения этой программы:

```
The volume is 330
```

В этом примере функция box() с помощью return возвращает значение выражения length * width * height, которое затем присваивается переменной answer. Другими словами, значение, возвращаемое инструкцией return, становится значением выражения box() в вызывающем коде.

Поскольку box() теперь возвращает значение, ее прототип и определение не предваряются ключевым словом void. (Запомните: ключевое слово void используется только для функции, которая *не* возвращает значение.) На этот раз функция box() объявляется как возвращающая значение типа int.

Разумеется, int — не единственный тип данных, которые может возвращать функция. Мы уже говорили, что она способна возвращать данные любого допустимого в C++ типа, за исключением массива. Например, в показанной ниже программе мы переделаем функцию box() так, чтобы она принимала параметры типа double и возвращала значение того же типа:

```
// Возврат значения типа double

#include <iostream>
using namespace std;

// Использование параметров типа double
double box(double length, double width, double height);

int main()
{
   double answer;

   answer = box(10.1, 11.2, 3.3); // Присваиваем возвращаемое значение
   cout << "The volume is " << answer;

   return  0;
}

// Эта версия функции box() использует данные типа double
double box(double length, double width, double height)
{
   return  length * width * height ;
}
```

Ниже показан результат выполнения этой программы:

```
The volume is 373.296
```

Следует отметить, что если функция, которая возвращает значение, отличное от void, завершается из-за обнаружения закрывающей фигурной скобки, инициатору вызова функции вернется неопределенное (неизвестное) значение. Из-за особенностей формального синтаксиса C++ функция, возвращающая значение, отличное от void, на самом деле не обязана выполнять инструкцию

return. Но поскольку функция объявлена как возвращающая значение, оно должно быть возвращено «во что бы то ни стало» — пусть даже это будет «мусор». Конечно, хорошая практика программирования подразумевает, что функция, которая возвращает значение, отличное от void, должна делать это посредством явно выполняемой инструкции return.

ВАЖНО!

5.5 Использование функций в выражениях

В предыдущем примере значение, возвращаемое функцией box(), присваивалось переменной, а затем значение этой переменной отображалось с помощью инструкции cout. Эту программу можно переписать в более эффективном варианте, используя возвращаемое функцией значение непосредственно в cout. Например, функцию main() из предыдущей программы мы можем представить в таком виде:

```
int main()
{
   // Непосредственное использование значения, возвращаемого функцией box()
   cout << "The volume is " << box(10.1, 11.2, 3.3);

   return  0;
}
```

Использование значения, возвращаемого функцией box(), непосредственно инструкцией cout

При выполнении этой инструкции cout автоматически вызывается box() для получения возвращаемого ею значения, которое и выводится на экран. Получается, совершенно необязательно присваивать его сначала некоторой переменной.

В общем случае void-функцию можно использовать в выражении любого типа. При его вычислении функция, которая в нем содержится, автоматически вызывается с целью получения возвращаемого ею значения. Например, в следующей программе суммируется объем трех параллелепипедов, а затем отображается среднее значение.

```
// Использование функции box() в выражении

#include <iostream>
using namespace std;

// Используем параметры типа double
double box(double length, double width, double height);

int main()
{
   double sum;

   sum = box(10.1, 11.2, 3.3) + box(5.5, 6.6, 7.7) +
         box(4.0, 5.0, 8.0);
```

```
   cout << "The sum of the volumes is " << sum << "\n";
   cout << "The average volume is " << sum / 3.0 << "\n";

   return  0;
}

// В этой версии функции box() используются параметры типа double
double box(double length, double width, double height)
{
   return  length * width * height ;
}
```

При выполнении эта программа генерирует следующие результаты:

```
The sum of the volumes is 812.806
The average volume is 270.935
```

Проверьте себя

1. Покажите две формы оператора `return`.
2. Может ли `void`-функция возвращать значение?
3. Может ли функция быть частью выражения?

Правила действия областей видимости функций

До сих пор мы использовали переменные, не рассматривая формально, где их можно объявить, как долго они существуют и какие части программы могут получить к ним доступ. Эти атрибуты определяются правилами действия областей видимости, установленными в C++.

Правила действия областей видимости любого языка программирования — это правила, которые позволяют управлять доступом к объекту из различных частей программы. Также они определяют продолжительность «жизни» переменной. В языке C++ определено две основные области видимости: *локальная* и *глобаль-*

Ответы

1. Ниже показаны две формы оператора `return`:

   return;

   return *значение*;
2. Нет, `void`-функция не может возвращать значение.
3. Вызов функции, которая возвращает значение, отличное от `void`, можно использовать в выражении любого типа. В этом случае функция, которая в нем содержится, автоматически выполняется с целью получения возвращаемого ею значения.