

Введение в программирование на C++

Артамонов Ю.Н.

Филиал «Котельники» университета «Дубна»

12 сентября 2018 г.

Содержание

- 1 О некоторых отличиях языка C++ от языка C

Потоковый ввод/вывод в C++

Язык C++ был разработан Бьёрном Страуструпом как надмножество языка C, это значит, что в основном программисты могут использовать компилятор C++ для компиляции существующих программ на C без каких-либо изменений, а некоторые ведущие компании вообще не предлагают специальных средств для разработки программ на языке C (например, Visual Studio). Первое отличие, которое бросается в глаза - это новые возможности C++ по организации ввода - вывода данных. Вместо уже известных `printf`, `scanf` рекомендуется использовать соответственно `cout`, `cin`, а также операцию передачи данных в поток `< <` (произносится «послать в») и операцию чтения данных из потока `> >` (произносится «взять из»). В качестве примера рассмотрим две реализации одной и той же программы. Для компиляции программы на C++ в linux можно использовать команду:

```
g++ name.cpp -o nameout
```

Первая реализация программы примера

```
#include <stdio.h>
int main()
{
    int a;
    printf("Введите целое число:");
    scanf("%d",&a);
    printf("Итак, Вы ввели число %d\n", a);
    return 0;
}
```

Как видно, эта программа на C++ вообще ничем не отличается от аналогичной программы на языке C (разве что раньше мы допускали вольность не указывать тип возвращаемого значения у функции `main`).

Вторая реализация программы

```
#include <iostream>
using namespace std;
int main()
{
    int a;
    cout<<"Введите целое число:";
    cin >> a;
    cout <<"Итак, Вы ввели число " << a<<endl;
    return 0;
}
```

В данном примере отличий уже много. Рассмотрим каждое из них более детально.

Особенности второй реализации

Во-первых, как видно, вместо библиотеки стандартного ввода/вывода *stdio.h* подключается библиотека потокового ввода/вывода *iostream*. После этого становятся доступны функции `cout`, `cin`, `endl`. Правда для их удобного использования следует указать *пространство имен* (*namespace*), используемых по умолчанию. Для этой цели служит директива препроцессору `using namespace std;`. Если этого не писать, вызвать функции `cout`, `cin`, `endl` тоже можно, но уже длиннее: `std::cout`, `std::cin`, `std::endl`.

Во-вторых, для вывода на экран используется поток `cout`, в котором не нужно указывать спецификаторы `%d` и тому подобные вещи. Это уже много удобнее. Библиотека сама понимает, как выводить переменную соответствующего типа на экран (форматированный вывод с заданной точностью рассмотрим позже). Это относится и к `cin` - все, что нужно, просто перенаправить вывод из `cin` в переменную `a`. При этом следует обратить внимание, амперсанд `&` перед переменной, получающий ее физический адрес, также не нужен.

Особенности второй реализации (продолжение)

Следует также обратить внимание на использование `endl` для переноса строки (на самом деле специальная аббревиатура `endl` отвечает за закрытие буфера, что эквивалентно переносу строки). Конечно, вместо `endl` можно было использовать старый, добрый `esc`-код:

```
cout <<"Итак, Вы ввели число " << a<<' '\n';
```

Можно вводить, выводить сразу несколько переменных разных типов, не заботясь об этом, например, так:

```
std::cin >> a >> b;
```

```
std::cout <<"Итак, Вы ввели число " << a << ", " << b << std::endl;
```

В любом случае - это новая более гибкая возможность с сохранением старого функционала.

Особенности объявления переменных

В языке C все объявления должны находиться в начале блока до любых исполняемых операторов. В языке C++ объявления могут размещаться всюду, где может стоять исполняемый оператор, при условии, что они предшествуют использованию того, что объявляется. Например,

```
for (int i=0; i<=9; i++) cout << i;  
cout << endl;
```

При этом область действия локальной переменной в языке C++ начинается с ее объявления и распространяется до закрывающейся правой скобки }. Т.е. в нашем примере по выходу из цикла переменная `i` уже не видна. И код:

```
for (int i=0; i<=9; i++) cout << i;  
cout << endl;  
cout << i;
```

приведет к сообщению об ошибке.

Использование прототипов

Прототипы функций позволяют компилятору C контролировать правильность вызова функций с точки зрения соответствия типов. В ANSI C они не являются обязательными.

В языке C++ для всех функций требуется их прототип!

Хотя функция, определенная в файле до первого обращения к ней, действует как прототип. Например, так можно:

```
#include <iostream>
using namespace std;
int f(int a, int b) {return a+b;}
int main() { cout <<f(6,8); return 0; }
```

А так нельзя:

```
#include <iostream>
using namespace std;
int main() { cout <<f(6,8); return 0; }
int f(int a, int b) {return a+b;}
```

Использование прототипов (продолжение)

Здесь уже нужно использовать прототип

```
#include <iostream>
using namespace std;
int f(int, int);
int main() { cout << f(6,8); return 0; }
int f(int a, int b) {return a+b;}
```

В языке C для определения пустого списка параметров в круглые скобки помещается ключевое слово `void`. Если в круглых скобках прототипа функции в C ничего не содержится, то для этой функции полностью отключается проверка параметров и не делается никаких предположений относительно числа этих параметров и их типа. При обращениях к этой функции могут передаваться любые параметры без выдачи компилятором каких-либо ошибок.

В C++ при задании пустого списка параметров в круглых скобках также либо записывается `void`, либо вообще ничего не записывается. Но это лишь означает, что у функции по-честному нет параметров.

Использование прототипов (продолжение)

Реализация программы в виде набора функций хороша с точки зрения разработки программного обеспечения, но обращения к функциям связаны с накладными расходами времени исполнения (нужно время, чтобы переключиться на функцию, а затем вернуться обратно в место ее вызова). В C++ предусмотрены встроенные функции, позволяющие уменьшить накладные расходы при вызове функций - особенно для функций небольших размеров. Для этого можно использовать модификатор `inline`, который помещается перед типом возвращаемого функцией значения в определении функции и рекомендует компилятору генерировать в месте вызова функции ее копию кода, с тем, чтобы избежать вызова функции. Компромисс состоит в том, что вместо одной копии функции, в которую передается управление всякий раз при ее вызове, в программу вставляется несколько копий этой функции. Компилятор может игнорировать `inline` и обычно так и поступает, за исключением очень маленьких функций.

Параметры-ссылки

В языке C при всех обращениях к функциям параметры передаются по значению. Передача параметров по ссылке имитируется в C путем передачи в функцию указателя на объект и последующего доступа к этому объекту путем разыменования указателя в вызываемой функции. В языке C++ в функции можно объявлять параметры -ссылки. Чтобы указать, что параметр передается по ссылке необходимо после типа параметра в прототипе функции поместить символ амперсанда &. Например, можно написать такую программу на C++:

```
#include <iostream>
using namespace std;
void f(int &);
void f(int & a) {a++;}
int main(){
    int a;
    cin>>a;
    f(a);
    cout<<a<<endl;}
```

Параметры-ссылки

Это было бы эквивалентно следующей программе в C

```
#include <stdio.h>
void f(int *);
void f(int* x) {(*x)++;}
int main()
{
    int a;
    scanf("%d",&a);
    f(&a);
    printf("%d", a);
}
```

Ссылки также могут служить в качестве псевдонимов для других переменных внутри некоторой функции. Например, :

```
int a = 666;
int &b = a;
b+=111;
cout<<a<<endl;
```

Параметры-ссылки

В последнем примере это эквивалентно увеличению переменной `a` на 111. Переменные-ссылки должны инициализироваться при их объявлении и не могут быть переназначены в качестве псевдонимов других переменных.

Динамическое распределение памяти

Для упрощения управления динамической памятью в C++ предназначены операции `new`, `delete`.

В C для этих целей использовались `malloc`, `free`. При этом `malloc` требовало явного указания количества выделяемых байт с помощью `sizeof`. В C++ память выделить проще. Оператору `new` не нужно указывать размер памяти, он сам определит его по типу создаваемого объекта. При этом создаваемый объект можно еще инициализировать:

```
float *a = new float (3.14);
```

В этом примере выделяется память для объекта типа `float`, на который указывает указатель `a`.

Динамическое распределение памяти (продолжение)

Можно также создавать динамические массивы:

```
#include <iostream>
using namespace std;
int main()
{
    int *ar;
    ar = new int [3];
    cin >> ar[0]; cin >> ar[1]; cin >> ar[2];
    cout<<ar[0]<<ar[1]<<ar[2];
    delete [] ar;
    return 0;
}
```


Динамическое распределение памяти (продолжение)

В данном примере массив заполняется случайными числами:

```
#include <stdlib.h>
#include<iostream>
using namespace std;
int main()
{
    int n;
    cout<<"Введите размерность массива n = ";
    cin>>n;
    int *h;
    h = new int [n];
    srand(time(0));
    for (int i = 0; i<n; i++){
        h[i] = 1+rand()%100;
        cout<<h[i]<<endl;
    }
    delete [] h;
    return 0;
}
```

Задание 1.1

Реализуйте алгоритм нахождения простых чисел с помощью решета Эратосфена.

Пример работы с динамическими двумерными массивами

```
#include <stdlib.h>
#include<iostream>
using namespace std;
int main(){
    int n,m, **h; cin>>n>>m; srand(time(0));
    h = new int *[n];
    for (int i=0;i<n;i++) h[i] = new int [m];
    for (int i = 0; i<n; i++){
        for (int j=0;j<m;j++){
            h[i][j]=rand()%9;
            cout<<h[i][j];
        }
        cout<<endl;
    }
    for (int i = 0;i<n;i++)
        delete [] h[i];
    delete [] h;
    return 0;
}
```

Аналогичный пример - реализация с помощью функций

```
#include <stdlib.h>
#include<iostream>
using namespace std;
void zap(int, int, int **);
int main(){
    int n,m;
    cin>>n>>m;
    int **h;
    h = new int *[n];
    for (int i=0;i<n;i++) h[i] = new int [m];
    zap(n,m,h); srand(time(0));
    for (int i = 0; i<n; i++){
        for (int j=0;j<m;j++) cout<<h[i][j];
        cout<<endl;}
    for (int i = 0;i<n;i++)
        delete [] h[i];
    delete [] h;
    return 0;}
```

Аналогичный пример - реализация с помощью функций (продолжение)

```
void zap(int n, int m, int **ar)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++) ar[i][j] = rand() % 9;
}
```

Задание 1.2

Реализуйте алгоритм транспонирования матрицы размером $n \cdot m$

Задание 1.3

Реализуйте алгоритм умножения матриц $n \cdot m$, $m \cdot k$.

Задание 1.4

Реализуйте алгоритм циклического сдвига строк матрицы на 1 элемент вправо.

Например, если матрица была вида:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix},$$

то в итоге должна получиться матрица:

$$\begin{pmatrix} 2 & 3 & 4 & 1 \\ 6 & 7 & 8 & 5 \\ 10 & 11 & 12 & 9 \end{pmatrix}$$

Задание 1.5

Задача Иосифа. По кругу располагаются n людоедов. Ведущий считает по кругу, начиная с первого, людоеды съедают каждого m -го собрата. Круг смыкается, счет возобновляется со следующего после съеденного. Так продолжается, пока не останется только один людоед. Найти номер оставшегося в живых людоеда.