Объектно-ориентированное программирование на алгоритмическом языке C++

МИРЭА, Институт Информационных технологий,
кафедра Вычислительной техники

Автор: доцент, канд. физ.-мат. наук,
Путуридзе Зураб Шотаевич

Параметризованные конструкторы

Синтаксис вызова параметризированного конструктора при объявлении объекта:

«имя класса» «имя объекта» (список аргументов);

«имя класса» «имя объекта» = «имя класса» (список аргументов);

Если у конструктора только один параметр, то можно использовать альтернативный способ вызова конструктора:

«имя класса» «имя объекта» = аргумент;

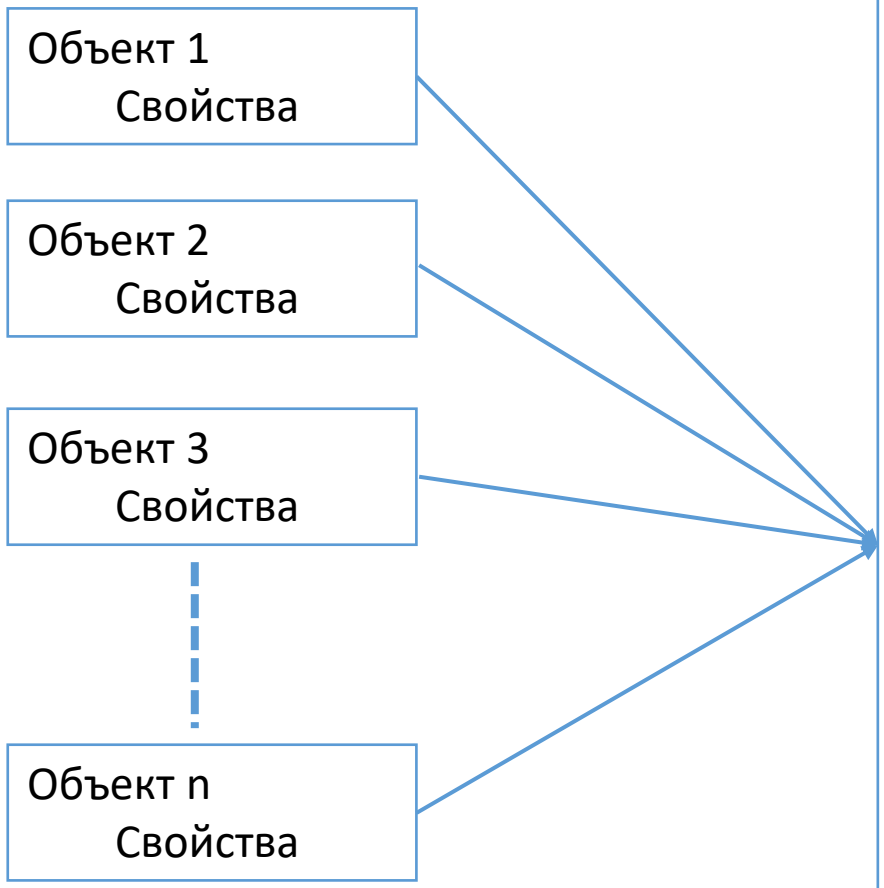
Параметризованные конструкторы

```
#include <iostream>
using namespace std;
class cl_3 {
     int i;
public:
          cl_3   ( );        // конструктор по умолчанию
          cl_3   ( int j ); // конструктор с одним параметром
    void show_i ( );
};
cl_3 :: cl_3 ( )  {
    cout << "Конструктор по умолчанию \n";
    i = 10;
}
cl_3 :: cl_3 ( int j )  {
    cout << "Конструктор с параметром \n"; ;
    i = j;
}
void  cl_3 :: show_i ( )  {
    cout << “i = ” << i << “\n”;
}
```

Параметризованные конструкторы

```
#include <iostream>
#include <locale>
using namespace std;
int main ( )
{
    setlocale ( LC_ALL, "Russian" );
    cl_3 ob;           // отработка конструктора по умолчанию
    cl_3 ob1 ( 5 );    // отработка конструктора с параметром
    cl_3 ob2 = 7;      // отработка конструктора с параметром
    ob.show_i  ( );    // вызов открытого метода.
    ob1.show_i ( );    // вызов открытого метода.
    ob2.show_i ( );    // вызов открытого метода.
    return  0;
}
Программа выведет на консоль следующее:
Конструктор по умолчанию
Конструктор с параметром
Конструктор с параметром
i = 10
i = 5
i = 7
```

Указатель this




```
Метод 1
{
    // код реализации
}
Метод 2
{
    // код реализации
}
Метод 3
{
    // код реализации
}

……….

Метод  m
{
    // код реализации
}
```

Указатель this

```
#include <iostream>
using namespace std;
class cl_1 {
     int val;
public:
    void load_val ( int val );
    int  get_val  ( );
};
void cl_1 :: load_val ( int val )    {
     this -> val = val;
}
int cl_1 :: get_val ( )      {
     return this -> val; // тоже самое, что и return val;
}
```

Указатель this

```
int main ( )
{
    cl_1   ob; // отработка конструктора по умолчанию
    ob.load_val ( 50 ); // инициализация свойства val.
                        // вывод значения val.
    cout << "val = " << ob.get_val ( );
    return  0;
}
```

Программа выведет на консоль следующее:

```
val = 50
```

Указатели и ссылки на объекты

```
#include <iostream>
using namespace std;
 // ----- Заголовочная часть.
class cl_1 {
private:
     int i;
public:
          cl_1   ( ); // конструктор
     void show_i ( );
     int  n;
};
 // ----- Часть реализации.
cl_1 :: cl_1 ()
{
    cout << "В конструкторе \n";
    i = 10;
}
void  cl_1 :: show_i ( ) {
    cout << “i = “ << i << “\n”;
}
```

Указатели и ссылки на объекты

В C++ ссылка – это простой ссылочный тип.

Объявление ссылки означает задание альтернативного идентификатора. По сути, ссылка это указатель, который привязан к определенной области памяти

```
cl_1   ob;          //объявление объекта, 0xdd000075
cl_1 & r_ob = ob; //альтернативное имя объекта, 0xdd000075
                    //объявление и инициализация
cout  << "\n" << & ob << "\n" << & r_ob;
```

Программа выведет на консоль следующее:

```
0xdd000075
0xdd000075
```

Указатели и ссылки на объекты

```
#include <iostream>
#include <locale>
using namespace std;
int main ( )
{
        setlocale ( LC_ALL, "Russian" );
        cl_1   ob;              // объявление объекта
        cl_1 * p_ob;            // объявление указателя объекта заданного класса
        p_ob = & ob;            // присвоение к указателю объекта адреса объекта
        p_ob -> show_i ( );     // вывод значения свойства объекта
        return  0;
}
```

Программа выведет на консоль следующее:

```
В конструкторе
i = 10
```

Указатели и ссылки на объекты

Оператор взятия адреса «&» используется для уже созданного объекта с целью получить его адрес, а ссылка это только задание альтернативного имени объекта.

```
cl_1   a;          // объявление объекта класса cl_1
cl_1   b;          // объявление объекта класса cl_1
cl_1 & ra =   a;   // объявление альтернативной имени объекта a и инициализация
cl_1 * p  = & ra; // объявление и инициализация указателя
a.n =   3;
b.n =   5;
cout << p -> n << "    " << ra.n << "\n";
p   = & b;
cout << p -> n << "    " << ra.n << "\n";

Программа выведет на консоль следующее:
В конструкторе
В конструкторе
3     3
5     3
```

Присвоение объектов

```
#include <iostream>
using namespace std;
class cl_2 {
    int i;
public:
    cl_2 ( );
    void set_i  ( int i );
    int * p_i;
};
void  cl_2 :: cl_2 ( )
{
    i   =   10;
    p_i = & i;
}
void  cl_2 :: set_i ( int i )
{
    this -> i = i;
}
```

Присвоение объектов

```
int main ( )
{
    cl_2  a, b;           // объявление объектов
    cout << * ( a.p_i ) << " " << * ( b.p_i ) << "\n";
    a.set_i ( 15 );       // инициализация свойства i в объекте a
    cout << * ( a.p_i ) << " " << * ( b.p_i ) << "\n";
    a = b;                // присвоение объекту a объекта b
    cout << * ( a.p_i ) << " " << * ( b.p_i ) << "\n";
    b.set_i ( 35 );       // инициализация свойства i в объекте a
    cout << * ( a.p_i ) << " " << * ( b.p_i ) << "\n";
    return  0;
}
```

Программа выведет на консоль следующее:

```
10      10
15      10
10      10
35      35
```

Присвоение объектов

a	b
a.p_i -> a.i (a.p_i == & a.i)	b.p_i -> b.i (b.p_i == & b.i)
После присвоение a = b;	
a.p_i -> b.i (a.p_i = b.p_i ; a.p_i == & b.i)	b.p_i -> b.i (b.p_i == & b.i)
После изменения значения b.i: b.set_i (35);	

```
cout << * ( a.p_i ) << " " << * ( b.p_i ) << "\n";
```

Возвращает:

35 35

т.к. a.p_i == b.p_i, содержат один и тот же адрес

Присвоение объектов

Синтаксис выражения:

«имя объекта 1» = «имя объекта 2»;

```
#include <iostream>
using namespace std;
class cl_1 {
    int i;
public:
    void set_i  ( int k );
    void show_i ( );
};
void  cl_1 :: set_i ( int k )
{
    i = k;
}
void  cl_1 :: show_i ( )
{
    cout << "i = " << i << "\n";
}
```

Присвоение объектов

```
int main ( )
{
    cl_1  ob_1, ob_2;     // объявление объектов
    ob_1.set_i ( 11 );    // инициализация свойства i в ob_1.
    ob_2.set_i ( 15 );    // инициализация свойства i в ob_2.
    ob_1.show_i ( );      // вывод значения свойства объекта ob_1.
    ob_1 = ob_2;          // присвоение объекту ob_1 объекта ob_2.
    ob_1.show_i ( );      // вывод значения свойства объекта ob_1.
    return  0;
}
```

Программа выведет на консоль следующее:

```
i = 11
i = 15
```

Передача объектов функциям

Объекты можно передавать функциям в качестве аргументов точно так же, как передаются переменные других типов. Параметр функции объявляется как имеющий тип класса. При вызове функции объект этого класса используется в качестве аргумента. Как и для переменных других типов, по умолчанию объекты передаются в функции по значению. Внутри функции создается копия аргумента и эта копия, а не сам объект, используется функцией. Поэтому изменение копии объекта внутри функции не влияет на сам объект.

Передача объектов функциям

```
#include <iostream>
using namespace std;
class samp {
    int i;
public:
          samp  ( int n );
    void set_i ( int n );
    int  get_i ( );
};

samp :: samp  ( int n )
{
       i = n;
}
void samp :: set_i ( int n )
{
       i = n;
}
int samp :: get_i ( )
{
       return i;
}
```

Передача объектов функциям

```
/* Заменяет переменную o.i ее квадратом. Однако это не влияет на объект,
используемый для вызова функции sqr_it () */
void sqr_it ( samp o )
{
    o.set_i ( o.get_i ( ) * o.get_i ( ) );
    cout << "Для копии объекта a значение i равно: " << o.get_i ( );
    cout << "\n";
}

int main ( )
{
    samp a ( 10 );
    sqr_it (  a );          // передача объекта a по значению
    cout << "Переменная a.i в функции main ( ) не изменилась: ";
    cout << a.get_i();      // выводится 10
    return 0;
}
```

```
Программа выведет на консоль следующее:
Для копии объекта a значение i равно: 100
Переменная a.i в функции main() не изменилась: 10
```

Передача объектов функциям

Когда при вызове функции создается копия объекта, конструктор для создания копии не вызывается.

При завершении работы функции копия удаляется, деструктор созданной копии вызывается.

Передача объектов функциям

```
#include <iostream>
using namespace std;
class samp {
     int i;
public:
     samp  ( int n );
    ~samp  ( );
 int get_i ( );
};

samp :: samp ( int n )
{
       i = n;
       cout << "Работа конструктора \n";
}
samp :: ~samp ( )
{
       cout << "Работа деструктора \n";
}
int samp :: get_i ( )
{
       return i;
}
```

Передача объектов функциям

```
// Возвращает квадрат переменной o.i
int sqr_it ( samp o ) {
    return o.get_i ( ) * o.get_i ( );
}

int main ( )
{
      samp a ( 10 );
      cout << sqr_it( a ) << "\n";
      return 0;
}
```

```
Программа выведет на консоль следующее:
Работа конструктора
100
Работа деструктора
Работа деструктора
```

Конструктор копии

Описание заголовка конструктора копии

«имя класса» (const «имя класса» & «имя параметра»);

```
class cl_1 {
        int i;
public:
        cl_1   ( );
        cl_1   ( const cl_1 & ob );
  int   get_i ( );
};
cl_1 :: cl_1   ( ) {
     cout << "Конструктор по умолчанию \n";
     i = 15;
}
cl_1 :: cl_1   ( const cl_1 & ob ) {
     cout << "Конструктор копии \n";
     i = ob.i + 5;
}
int cl_1 :: get_i ( ) {
     return i;
}
```

Конструктор копии

```
void func ( cl_1 ob_local ) {
     cout  << “ob_local  i = “ << ob_local.get_i ( ) << "\n";
}
int main ( ) {
       setlocale ( LC_ALL, "Russian" );
       cl_1  ob;          // Конструктор по умолчанию
       func ( ob );       // Конструктор копии
       cout  << “ob      i = “ << ob.get_i ( );
       return 0;
}
```

```
Программа выведет на консоль следующее:
Конструктор по умолчанию
Конструктор копии
ob_local  i = 20
ob        i = 15
```

Объекты в качестве возвращаемого значения метода или функции

```
class cl_1 {
        int i;
public:
        cl_1   ( );
        cl_1   ( const cl_1 & ob );
       ~cl_1   ( );
  void set_i ( int k );
  int  get_i ( );
};
cl_1 :: cl_1   ( ) {
     cout << "Конструктор по умолчанию \n";
     i = 10;
}
cl_1 :: cl_1   ( const cl_1 & ob ) {
     cout << "Конструктор копии \n";
     i = ob.i + 3;
}
cl_1 :: ~cl_1   ( ) { cout << "Деструктор \n";}
void cl_1 :: set_i ( int k ) { i = k;      }
int  cl_1 :: get_i ( )       { return i;   }
```

Объекты в качестве возвращаемого значения метода или функции

```
cl_1 func ( ) {
    cl_1 ob_local;
    ob_local.set_i ( 13 );
    cout  << “ob_local  i = “ << ob_local.get_i ( ) << "\n";
    return ob_local;
}
int main ( ) {
       setlocale ( LC_ALL, "Russian" );
       cl_1  ob;                  // Конструктор по умолчанию
       cout  << “ob     i = “ << ob.get_i ( ) << "\n";
       ob = func ( );
       cout  << “ob     i = “ << ob.get_i ( ) << "\n";
       return 0;
}
```

```
Программа выведет на консоль следующее:
Конструктор по умолчанию
ob        i = 10
Конструктор по умолчанию
ob_local  i = 13
Деструктор
ob        i = 13
Деструктор
```

Встраиваемые методы

```
// В заголовочной части
«тип метода» «имя метода» ( [«параметры»] )
{
	// код реализации
}
```

```
// В части реализации
inline «тип метода» «имя класса» :: «имя метода» ( [«параметры»] )
{
	// код реализации
}
```

Пример

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

class cl_1 {
   int i;
   int k;
public:
   int get_i ( ) { return i; }
   int get_k ( );
};

inline int cl_1 :: get_k( )
{
   return k;
}
```

Дружественная функция

Дружественная функция - не является членом класса, но получает доступ к закрытым элементам объекта класса.

```
friend «тип функции» «имя функции» ( «параметры» );
```

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

class cl {
     // ...
public:
     friend void frnd ( cl ob );
     // ...
 };
```

Предварительное объявление наименования класса

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

class myclass;

int sum ( myclass x )
{
  return x.a + x.b;
}

class myclass {
  int a, b;
public:
  myclass ( int i, int j ) { a=i; b=j; }
  friend int sum ( myclass x ); // дружественная функция
};

int main()
{
  myclass n ( 3, 4 );
  cout << sum ( n );
  return 0;
}
```

Дружественный класс

```
friend «имя дружественного класса»;
```

```
#include <iostream>
using namespace std;
class cl_f;
class cl_1 {
  int a, b;
public:
  cl_1 ( int i, int j ) { a=i; b=j; }
  friend  cl_f; // дружественный класс
};
class cl_f {
public:
     void change ( cl_1 & c ) { c.a = c.b; }
};
```