Объектно-ориентированное программирование на алгоритмическом языке C++

МИРЭА, Институт Информационных технологий, кафедра Вычислительной техники

Основные определения

- **Объект -** то, что может быть индивидуально описано и рассмотрено.

- **Система** - множество взаимосвязанных и взаимодействующих объектов для решения одной или множества задач (достижения одной или множества целей).

Объявление объекта и доступ к его элементам

Объявление

«имя класса» «имя объекта»{

описание полей класса

} «имя объекта 1», «имя объекта 2»;

Доступ

«имя объекта».«имя элемента объекта» ([список аргументов]) ;

Пример 1

```
#include <iostream>
using namespace std;
 // ----- Заголовочная часть.
class   myclass {
   int a;
public:
         myclass ( ); // конструктор
    void show    ( );
};
// ----- Часть реализации.
myclass :: myclass ( ) {
   cout << "В конструкторе \n"; ;
   a = 10;
}
void  myclass :: show ( ) {
   cout << a ;
}
```

```
// ----- Основная программа
int main ( )
{
    myclass    ob;      // объявление объекта, отработка
конструктора

    ob.show ( );      // вызов открытого метода.

    return  0;
}
```

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 ( );
};
 // ----- 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

```
// ----- Основная программа
int main ( )
{
   cl_1   ob;            // объявление объекта
   cl_1 * p_ob;          // объявление указателя объекта

   p_ob = & ob;          // инициализация указателя объекта

   p_ob -> show_i ( );   // вывод значения свойства объекта
   (*p_ob).show_i ( );
   return  0;
}
```

Программа выведет на консоль следующее:

```
В конструкторе
i = 10
```

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

```
int  a;          // переменная типа int,
                 // по адресу  0xdd000075
int & ra = a; // альтернативное имя переменной
                 // по адресу  0xdd000075
cout << &a << "\n" << &ra << "\n";
```

Программа выведет на консоль следующее:

```
0xdd000075
0xdd000075
```

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.

```
int   a  =   3;   // переменная типа int
int   b  =   5;   // переменная типа int
int & ra =   a;   // альтернативное имя переменной a
int * p  =  &a;

p = & b;
cout << *p << "\n" << ra << "\n";
```

Программа выведет на консоль следующее:

```
5
3
```

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.
    cout << “val = ” << ob.get_val ( );  // вывод
                                         // значения val.
    return  0;
}
```

Программа выведет на консоль следующее:

```
val = 50
```

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
```

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

```
int main ( )  {
    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
```

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

При передаче объекта в функцию по значению создается копия объекта, конструктор копии не вызывается.

При завершении работы функции копия удаляется, деструктор 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
Работа деструктора
100
Работа деструктора
```