
Представление объекта в виде указателя: операторы * и ->

А. Г. Фенстер, <http://info.fenster.name>

16 ноября 2009 г.

Конспект семинара №4 по объектно-ориентированному программированию. Пожалуйста, не относитесь к этому как к руководству по C++ или ООП: здесь изложен только самый минимум информации по теме; тем не менее, этого минимума должно хватить для выполнения второго задания.

О найденных ошибках и неточностях, пожалуйста, пишите мне по адресу fenster@fenster.name. Также буду рад получить любые комментарии по поводу этого текста.

В [первой](#) и [второй](#) частях было введено понятие класса и показано, как определяются конструктор, деструктор и основные операторы. В [третьей](#) части было введено понятие шаблона класса.

Темой этой части будет являться определение унарного оператора * (разыменование, не путать с бинарным оператором умножения) и унарного оператора -> (селектор члена объекта). Эти операторы позволяют работать с экземплярами произвольного (определённого пользователем) класса как с указателем, хотя на самом деле этот объект указателем не является.

Сначала вспомним, как устроена работа с указателями (всё это подробно обсуждалось в прошлом семестре на практике по языку C).

Указатели и операция разыменования

Пусть T — некоторый тип данных и a — переменная этого типа:

T a ;

Определено понятие адреса переменной `a`, то есть, условно, номера ячейки памяти, в которой расположена переменная `a`. Адрес переменной можно узнать при помощи оператора `&` и присвоить переменной типа `T *`:

```
T *p;
p = &a;
```

Не путайте оператор взятия адреса со ссылкой на некоторое значение, которая также вводится при помощи символа `&`.

Имея переменную-указатель (т. е. переменную типа `T *`), можно получить значение, на которое она указывает, при помощи операции разыменования `*`:

```
int a = 7;
int *p = &a;
int x = *p; // в x присваивается 7
```

Если же переменная является указателем на структуру, то обратиться к членам этой структуры можно двумя способами:

```
struct s
{
    int m, n;
};
s a;
s *p = &a;
(*p).m = 1; // разыменование указателя
           // и обращение к члену структуры
p->n = 2;   // -> объединяет разыменование
           // и выбор члена структуры
```

В языке `C` обращение через `->` было просто более удобным, но семантически эквивалентным способом обращения к элементу структуры через указатель (всегда можно было заменить `p->a` на `(*p).a`).

В `C++` существует возможность определить операторы `*` и `->` для своего класса, в этом случае с объектом этого класса можно будет работать как с указателем. Необходимо понимать, что в случае, когда эти операторы определены программистом, эквивалентность `p->a` и `(*p).a` вовсе не обязательно имеет место, хотя хорошим стилем является по возможности сохранять смысл операторов при их переопределении.

Оператор *

Оператор * может возвращать значение любого типа. Он нужен, если хочется представить свой объект как указатель на тип T, тогда `operator*` должен возвращать T. Пример:

```
class IntPtr
{
    int *p;
public:
    IntPtr(int *q) { p = q; }
    int operator*() { return (*p); }
};
```

Объект этого класса можно использовать как указатель на `int`:

```
int a = 7;
IntPtr p = &a; // то же, что IntPtr p(&a);
std::cout << *p << "\n"; // напечатает 7
```

Оператор ->

Аналогичным образом можно определить оператор `->`, который обязан возвращать либо указатель на структуру или класс, либо класс с определённым оператором `->`. Пример использования:

```
struct s
{
    int a, b;
};

class C
{
public:
    s *operator->() { ... }
};
```

Здесь оператор `->` возвращает указатель на структуру типа `s`, следовательно, имея объект класса `C`, можно использовать оператор `->` следующим образом:

```
C obj;  
std::cout << obj->a << ", " << obj->b << "\n";
```

Итераторы

Итератором называется объект, позволяющий перебирать содержимое некоторого контейнера (например, элементы списка, не задумываясь над внутренней структурой этого контейнера. В случае односвязного списка итератор должен предоставлять возможность обращения к текущему элементу и перехода к следующему элементу списка, а также определения того, что достигнут конец списка.

В стандартной библиотеке языка C++ определён шаблонный класс для хранения списка `std::list<T>`, имеющий итератор, определённый как подкласс (класс внутри класса) `std::list<T>::iterator` и допускающий использование по такой схеме:

```
#include <list>  
  
std::list<int> li;  
...  
for (std::list<int>::iterator i = li.begin; i != li.end(); ++i)  
{  
    // здесь *i -- значение текущего элемента списка  
}  
  
struct s  
{  
    int a, b;  
};  
std::list<s> ls;  
...  
for (std::list<s>::iterator i = ls.begin; i != ls.end(); ++i)  
{  
    // здесь допустимы обращения i->a, i->b  
}
```

Для реализации такого итератора нужно объявить класс `Iterator` внутри класса `List`:

```
template<typename T>
class List
{
    struct item
    {
        T data;
        item *next;
    };
    item *head;
public:
    class Iterator
    {
        ...
    };
    Iterator begin() const { ... }
    Iterator end() const { ... }
    ...
};
```

Итератор должен хранить указатель на текущий элемент списка. Как подклассу класса `List`, ему доступна информация о том, как организован список (он «видит» структуру `item`). В конструкторе он может принимать указатель `item *` на некоторый элемент списка, тогда метод `begin` будет просто создавать итератор, указывающий на начало списка (`head`), а метод `end` — итератор с нулевым указателем.

Естественно, для класса `Iterator` необходимо определить операторы `++` (для перехода к следующему элементу), оператор `!=` и (самое главное) операторы для получения значения текущего элемента: `operator*` и `operator->`. Первый из них будет использоваться для списков из элементов «простых» типов, а второй — для списков, элементами которых являются классы или структуры, как в примере с `std::list` выше. Соответственно, оператор `*` должен возвращать тип `T` (значение элемента списка), а оператор `->` — тип `T *` (адрес элемента списка).