Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

31 октября 2017 г.

Оглавление

Лекция 9

Объектно-ориентированное программирование

9.1 Класс

Класс — это пользовательский тип, который объединяет данные и код, их обрабатывающий.

Раньше мы "клали"данные структуры в объект **struct**, а рядом дописывали функции для нее. Теперь же мы хотим, чтобы с нашей структурой можно было работать только через функции, объявленные внутри нее. Этот принцип носит название *принципа сокрытия данных*, а такая организация кода – *инкапсуляция*.

9.2 Пример класса

Самый простой способ сделать пример класса – это использовать **struct**. Построим структуру Студент, имеющую вид:

1. **age** – возраст студента
2. **name** – имя
3. Остальные методы, а именно:
   (a) **init** – создание экземпляра
   (b) **aging** – увеличение возраста на 1
   (c) **print** – печать информации о студенте.

Код 9.1: **struct** как пример класса

```
1  #include<iostream>
2
3  struct Student
4  {
5    int age;
6    std::string name;
7
8    void init(int _age, const std::string& _name)
9    {
10       this->name = _name;
11       // Alternatively, we can write
12       name = _name;
13       this->age = _age;
14   }
15
16   void aging()
17   {
18       age++;
19       std::cout << name << ": I'm now " << age << " years old" << "\n";
20   }
21
22   void print() const    // We cannot change the object itself
23   {
24       std::cout << name << "-" << age << "\n";
25   }
26 };
27
28 int main()
29 {
30     Student a, b;
31     //a.init(17, "Vasya");    // Won't work
32     // "Vasya" is char*[], but not std::string
33     a.init(17, std::string("Vasya"));
34     b = a;  // Dangerous!
35     a.print();
36     b.print();
37 }
```

В коде выше есть некоторые особенности:

1. Строка 20 — **const** после объявления функции означает, что внутри функции мы не можем менять сам объект класса.

2. Строка 31 – не будет работать, так как "Vasya"есть **char[]***, а метод класса требует **std::string**.

3. Строка 34 – опасная операция (побитовое копирование может вести себя непредсказуемо, прим. ТФ).

4. Мы можем изменять снаружи переменные *внутри* объекта, то есть, имеет силу код **a.age += 10**. Это плохо, и дальше мы рассмотрим, как такую проблему решить.

9.3 Конструкторы и деструкторы

Конструктор

Когда мы создавали объект "студент"в коде выше, нам каждый раз надо было инициализировать его через функцию **init()**. Однако мы можем это забыть сделать.

Для защиты от этого существуют конструкторы класса: особые методы, вызываемые при создании экземпляра класса. Перепишем код:

Код 9.2: Наш класс с конструктором

```
#include<iostream>

struct Student
{
  int age;
  std::string name;

  Student(int _age, const std::string& _name)
  {
      name = _name;
      age = _age;
  }

  void aging()
  {
      age++;
      std::cout << name << ": I'm now " << age << " years old" << "\n";
  }

  void print() const    // We cannot change the object itself
  {
      std::cout << name << "-" << age << "\n";
  }
};

int main()
{
    Student c(17, "Petya");
    c.aging();
    c.print();
}
```

Конструктор ничего не возвращает. Помимо этого, объект **не** считается созданным до тех пор, пока не будет выполнен его конструктор.

Деструктор

По аналогии с конструктором существует *деструктор*.

Деструктор — особый метод, который ничего не возвращает и вызывается при удалении объекта.

Конструктор по умолчанию

Если мы не создадим конструктор, то компилятор это сделает за нас, создав *конструктор по умолчанию*. То же самое с деструктором.

Однако если мы создадим свой конструктор, то экземпляры класса уже надо будет клепать по его правилам, то есть, нам надо будет передавать объекту при создании те аргументы, которые принимает конструктор (в нашем случае **age** и **name**).

Код 9.3: Особенности работы конструктора

```
1 ps = new Student;    // Won't work; we don't have empty constuctor!
2 ps = new Student(18, "Misha");  // What is OK
3
4 m = new Student[10]; // Won't work; no default constuctor!
```

В примере выше на строке 4 требуется вызов конструктора по умолчанию (при создании массива), а его нет – поэтому она и помечена как неработающая.

9.4 Примеры работы конструктора и деструктора

Код 9.4: Пример вызова деструктора

```
1  #include<iostream>
2
3  struct Student
4  {
5    int age;
6    std::string name;
7
8    Student(int _age, const std::string& _name)
9    {
10       name = _name;
11       age = _age;
12       std::cout << "I AM CREATED: " << name << "-" << age << std::endl;
13   }
14
15   void aging()
16   {
17       age++;
18       std::cout << name << ": I'm now " << age << " years old" << "\n";
19   }
20
21   void print() const    // We cannot change the object itself
22   {
23       std::cout << name << "-" << age << "\n";
24   }
25
26   ~Student()
27   {
28       std::cout << "I AM DESTROYED. RIP: ";
29       print();
30   }
31 };
32
33 int main()
34 {
35     Student c(17, "Petya"); // Create Petya
36     c.aging();
37     {
38         Student b(18, "Vasya"); // Create Vasya
39         b.aging();
40     }   // Destroy Vasya
41 }   // Destroy Petya
```

В коде выше на строке 12 мы делаем запись о том, что объект создан. На строке 26 объявлен дестуктор – код внутри будет создан, когда объект перестанет существовать.

Код 9.4 компилируется: попробуйте его запустить на своей машине и увидите, как работает деструктор.

9.5 Пример. Структура Stack

Код 9.5: Структура Stack

```
1  struct Stack
2  {
3      double *s;
4      int allocated; // bytes allocated to stack
5      int top;
6
7      Stack(int max_size=100) // It is default and custom constructor
8      {
9          s = new double[max_size];
10         allocated = max_size;
11         top = 0;
12     }
13
14     void push(double x)
15     {
16         if(top < allocated)
17             s[top++] = x;
18     }
19
20     double pop()
21     {
22         if(top == 0) return -1;
23         return s[--top];
24     }
25
26     ~Stack()
27     {
28         delete[] s;
29     }
30 };
```

Обратите внимание на конструктор (строка 7) – он может использоваться как конструктор по-умолчанию (если не передавать ему параметры, так можно, когда есть значение параметра по умолчанию), так и как свой конструктор (если передать параметр **max_size**). В конце мы освобождаем ту память, которую занял объект **Stack**.

Код всего проекта можно смотреть здесь.