Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

24 февраля 2015

Создание объекта

При создании объекта требуется:

- выделить под него память;
- проинициализировать его переменные различными значениями.

Для автоматического осуществления перечисленных действий в классах используются конструкторы.

Конструктор

Метод, который вызывается компилятором автоматиечски при создании объекта.

- нельзя вызвать явно, как обычный метод;
- имя конструктора всегда сопадает с именем класса;
- у конструктора отсутсвует тип возвращаемого значения, даже void;
- конструктор не может быть константным, виртуальным или статическим методом.

Конструктор по умолчанию

Конструктором по умолчанию называется конструктор у которого нет параметров или все параметры имеют значения по умолчанию.
Генерируется конпилятором автоматически, если программист явно не определил свой конструктор.

Автоматический конструктор по умолчанию(default)

- вызывает конструкторы для всех встроенных объектов класса;
- вызывает конструктор базового класса (при наследовании).

Компилятор не генерирует default конструктор, если:

- в классе объявлен любой другой конструктор;
- в классе объявлены константные члены данных;
- данный класс является производным, а в базовом классе конструктор объявлен как private.

Автоматический конструктор по умолчанию(default)

```
//animal.h
class Animal
{
  unsigned int age;
};

//main.cpp
#include "animal.h"
Animal aGlobal;

int main()
{
Animal aLocal;
}
```

Пользовательский конструктор по умолчанию(default)

```
//animal.h
class Animal
{
  enum AnimalType { Predator, Herbivous, Unknown };
  unsigned int age;
  AnimalType type;

public:
   Animal()
   { age = 0; type = Unknown; }
};

//main.cpp
#include "animal.h"
int main() {
Animal aLocal; //Выделена память и вызван
    конструктор по умолчанию.
}
```

Конструктор с параметрами

```
//animal.h
enum AnimalType { Predator, Herbivous, Unknown };
class Animal
{
  unsigned int age;
  AnimalType type;

public:
   Animal(int age, AnimalType t)
   { this->age = age; type = t; }
};
//main.cpp
#include "animal.h"
int main()
{
Animal aLocal(10, Predator); //Выделена память и
    вызван конструктор по умолчанию.
Animal aLocal2; //???
}
```

Конструктор с одним параметром

```
class A
{
int m_a;
public:
A(int a) { m_a = a; }
};

int main()
{
A a = 1;
A a(1);
}
```

Если добавить к конструктору ключевое слово explicit, то это запретит приводить тип компилятору неявно.

Конструкторы базовых типов

```
int  i = 0;
int i1 (0);

class A
{
  int i;
  const int x;

public:
  A() : i(0), x(0) {}
};
```

Перегрузка конструкторов

```
class Animal
{
...
public:
  Animal() { /*...*/ }
  Animal(int age, AnimalType) { /* ... */ }
};


int main()
{
  Animal an1;
  Animal an2(10, Predator);
}
```

Конструктор с параметрами по умолчанию

```
class Animal
{
...
public:
  Animal(int age = 0, AnimalType = Unknown) { /*
      ... */ }
//  Animal() {} //FIXME?
};


int main()
{
  Animal an1;
  Animal an2(10);
  Animal an3(10, Predator);
}
```

Динамическое создание объектов

```
Animal *p1 = new Animal;
Animal *p2 = new Animal();
Animal *p3 = new Animal(10);

delete p1;
delete p2;
delete p3;
```

Деструкторы в объекте

- в большинстве случаев вызывается компилятором неявно;
- можно вызвать явно, как метод;
- имя деструктора совпадает с именем класса, но с символом ~;
- не принимает параметров и ничего не возвращает;
- не может быть константным или статическим;
- может быть виртуальным.

Автоматический деструктор

- вызывает деструкторы встроенных объектов данного класса;
- вызывает деструкторы базовых классов.

Классы с динамической памятью

```
class Animal
{
    ...
    char *name;
public:
    Animal(unsigned int age, AnimalType, const char *
        pName);
```

Плохой способ реализации

```
Animal::Animal(unsigned int age, AnimalType t, char
    *pName)
{
  this->age = age;
  type = t;
  name = pName; //<--Потенциальная ошибка в
      программе
}
int main() {
  char ar[] = "Sharik";
  Animal a1(1, Unknown, ar);
  Animal a2(1, Unknown, ar); //если мы переименуем
      животное a2, то a1, тоже изменит имя!
  char *name = new char[64];
  cin >> name;
  Animal a3(1, Unknown, name);
  delete [] name;
  //продолжаем работать с a3 => это приведет к
      краху программы }
```

Правильный вариант реализации

```
Animal::Animal(unsigned int age, AnimalType t, char
    *pName)
{
  this->age = age;
  type = t;
  name = new char[ strlen(pName) + 1 ];
  strcpy(name, pName); //НО, кто очистит
      динамическую память от поля name ?
}
```

Удаление динамической памяти в классах

```
Animal::~Animal()
{
  delete [] name;
}
```

Явную релизацию деструктора, необходимо писать каждый раз, когда идет выделение динамической памяти в классе.

Конструктор копирования

Если нет явной реализации, генерируется компилятором автоматически. Используется для:

```
\item поэлементного копирования всех полей в классе;
\item вызывает конструктор копирования базового класса;
\item вызывает конструкторы копирования для встроенных объектов.
```

Компилятор не создает автоматический конструктор копирования, если:

```
\item в классе объявлены константные члены данных;
\item в классе объявлены ссылки;
\item в базовом классе конструктор копирования находится в секции \textbf{private}.
```

Конструктор копирования

```
{
  Animal a1(10, Predator, "Bobik");
  Animal a2 = a1; //Вызов конструктора копирования
      !!!
  Animal a3(a2);
} //вызов деструкторов
```

ВОПРОС: Какая потенциальная проблема возникнет между объектами a1, a2 и a3, даже если у них корректно реализовон конструктор с параметрами и деструктор?

Реализация правильного конструктора копирования

```
Animal::Animal(const Animal &other)
{
  age = other.age;
  type = other.type;
  name = new char[ strlen(other.name) + 1 ];
  strcpy(name, other.name);
}
```

Конструктор копирования обычно в качестве параметра принимает константную ссылку, на объект такого же типа как и сам класс.

Неявный вызов конструктора копирования

```
void F(A a)
{ //вызов конструктора копирования
  //в функции происходит работа с копией
} //вызывается деструктор временного объекта a

A copy()
{
   A temp;
   return temp; //происходит вызов конструктора
       копирования
                  //в результате в вызывающую функцию
                      передается
                  //копия локального объекта
} //деструктор для объекта temp
```

move конструктор копирования

```
class Animal
{
...
public:
Animal(const Animal&); //обычный конструктор
    копирования
Animal(Animal &&); //move конструктор копирования
};

Animal f()
{
  Animal tmp;
  ...

  return tmp; //вызов move конструктора копирования
}
```

Реализация move конструктора копирования

```
Animal::Animal(Animal &&other)
{
  age = other.age;
  type = other.type;
  name = other.name; //отбираем указатель на
      временный объект
  other.name = nullptr; //присваиваем строке 0, т.к
      . деструктор всеравно удалит эту строку
}
```

Ключевое слово const и классы

const применимо к объектам и методам класса, в качестве принимаемого или возвращаемого значения.

Константные методы класса

```
class A
{
  int m_a;
public:
  int getA() const { //const означение, что все
      поля read-only
    //m_a++
    return m_a;
  }

  void setA(int a) {
    m_a = a;
  }
};
```

Ключевое слово mutable

mutable — означает, что поле может быть изменено из const-метода

```
class A
{
  mutable int counter;
  int m_a;
public:
  int getA() const { //const означение, что все
      поля read-only
    //m_a++;
    counter++; //OK
    return m_a;
  }

  void setA(int a) {
    m_a = a;
  }
};
```

Константные объекты

Для константных объектов можно вызывать, только константные методы

```
class A
{
public:
  void f1();
  void f2() const;
};

int main()
{
  A a;
  a.f1(); //OK
  a.f2(); //OK

  const A a;
  a.f1(); //???
  a.f2(); //???
}
```

Массив объектов

```
class A
{
  public:
   A(int) {}
};

int main()
{
  A arr[5]; //???
  A arr2[2] = { A(1), A(2) }; //???
  A arr3[5] = { A(1), A(2) }; //???
  A arr4[] = { 1, 2, 3 }; //???

  A *p = new A[10]; //???

  delete [] p; //???
}
```

Массив указателей на объекты

```
int main()
{
  A *arr[5]; //???
  A *arr2[2] = { new A(5), new A(4) };

  delete //???
}
```