МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение
высшего профессионального образования
"Донской государственный технический университет"
ДГТУ

Кафедра "Программное обеспечение вычислительной техники и автоматизированных систем"

Конструкторы и деструктор

Методические указания к лабораторной работе по дисциплине "Объектно-ориентированное программирование"

Ростов-на-Дону
20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.

УДК 512.3

Конструкторы и деструктор: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – с.

В методической разработке рассматриваются конструкторы и деструктор класса и особенности их реализации в C++. Даны задания по выполнению лабораторной работы. Методические указания предназначены для студентов направления 231000 "Программная инженерия".

© Издательский центр ДГТУ, 20

I. Конструкторы класса.

Для инициализации (задания начального значения) объекта класса (как при статическом, так и при или динамическом создании объекта) для него вызывается специальный метод класса, называемый конструктором. Конструктор – это метод, имя которого в точности совпадает с именем класса. Для класса можно создать несколько конструкторов, тогда они будут отличаться списками своих аргументов. Если в описании класса ни один конструктор не задан явно, компилятор создаст конструктор без аргументов. Но если задать хотя бы один конструктор для своего класса, компилятор не станет создавать конструктор по умолчанию. Например,

```
class A {
        int a;
  public:
        A(int x) { a = x; } /* конструктор с одним аргументом типа int, конструктор по умолчанию не создается */
};

int main() {
x       A a1;       /* Ошибка. Нет конструктора без аргументов. */
        A a2(10); /* Правильно. Вызывается конструктор с аргументом типа int */
        return 0;
}
```

II. Конструктор копирования.

В ситуациях, когда требуется создание копии объекта, т.е. когда новый объект создается на основе уже существующего объекта того же класса, вызывается конструктор специального вида – конструктор копирования. Это конструктор с одним аргументом – ссылкой на объект собственного класса. Например,

```
class A {
        int a;
  public:
        A() { a = 0; } /* конструктор без аргументов */
        A(A& x) {     /* конструктор копирования */
                a = x.a;
        }
};

int main() {
        A a1;           /* Вызывается конструктор без аргументов */
        A a2( a1 );   /* Вызывается конструктор копирования */
        A a3 = a1;   /* Вызывается конструктор копирования */
        return 0;
}
```

Если аргумент функции или метода – объект класса (но не ссылка на объект) то при вызове в ней (или в нем) создается новый локальный объект как копия переданного объекта, т.е. будет вызываться конструктор копирования. То же самое происходит, если функция или метод возвращают объект класса.

Если конструктор копирования для класса не задан явно, то компилятор создает его. Этот конструктор просто копирует значения полей старого объекта в соответствующие поля нового объекта. Это может приводить к проблеме повисших ссылок, если хотя бы одно поле класса указатель.

III. Конструкторы преобразования.

Для того, чтобы разрешить преобразование объектов чужого класса или величин примитивных типов в объект нашего класса необходимо задать для него конструкторы преобразования. Конструктор преобразования – это конструк-

тор с одним аргументом. Тип аргумента определяет тот тип, который может быть преобразован в объект класса. Например,

```
class A {
        int a;
  public:
        A(int x) { a = x; } /* конструктор с одним аргументом типа int */
};

void f( A x ); /* прототип функции с одним аргументом – объектом класса A */

int main() {
        A a1(10); /* вызывается конструктор с одним аргументом */
        f( 5 );       /* вызывается конструктор преобразования */
        return 0;
}
```

В этом примере при вызове функции f компилятор создаст временный объект класса A т.к. есть конструктор преобразования величины типа int в объект класса. Единственный конструктор в этом классе теперь выполняет две роли. Во-первых, это конструктор, вызываемый при создании новых объектов. Во-вторых, он задает преобразование целых значений в объекты класса. C++ позволяет объявлять конструкторы с одним аргументом так, чтобы их нельзя было использовать для преобразования типа. Для этого конструктор должен иметь модификатор explicit. Например,

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

```
void f( A x ); /* прототип функции с одним аргументом – объектом класса A */

    int main() {
        A a1(10); /* вызывается конструктор с одним аргументом */
    x   f( 5 );     /* Ошибка. Не определено преобразование int в A */
        return 0;
    }
```

IV. Деструктор класса.

Перед уничтожением объекта для него обязательно вызывается специальный метод класса, называемый деструктором. Имя деструктора начинается со значка тильда ~, после которого следует имя класса. Деструктор нельзя перегружать. Если он не задан для класса явно, то будет создан компилятором. Пример:

```
    class Person {
        char *name;
        int age;
      public:
        Person( char *n, int a) { /* конструктор */
            name = new char[ strlen(n) + 1 ]; /* захват дополнительной памяти */
            strcpy( name, n );
            age = a;
        }
        ~Person() { /* деструктор */
            delete[] name; /* освобождение памяти, захваченной конструктором */
        }
    };
```

V. Задания.

1. Создать структуру "Студент" содержащую следующие поля:
   - имя студента - произвольная длина (Си-строка)
   - отчество студента - произвольная длина (Си-строка)
   - фамилию студента - произвольная длина (Си-строка)
   - год рождения
   - группа.

2. Определить конструктор для инициализации полей структуры со значениями по умолчанию. Определить конструктор копирования и деструктор. Написать тестовый пример.

3. Изменить в описании структуры ключевое слово struct на class. Запустить программу. Какие возникли проблемы? Почему? Как их исправить?

4. Написать интерфейсные функции доступа к полям класса (получить/задать значение поля).

5. Внести в конструкторы и деструктор выдачу сообщений на экран о том, какая функция была вызвана. Модифицировать функцию main следующим образом:

```
void main(void)
{
   cout<<"Вход в функцию main()"<<endl;
   ...
   <тело_main()>
   ...
   cout<<"Выход из функции main()"<<endl;
}
```

Выяснить порядок вызова конструкторов и деструктора.

6. Описать глобальную функцию Student test(Student s){return s;} Вызвать ее в основной программе. Что произошло и почему?

7. Изменить передачу параметра ф-и test на передачу по ссылке.

Что изменилось?

8. Изменить возврат результата ф-и test на передачу по ссылке.

Что изменилось?

VI. Контрольные вопросы.

1. Каким образом инициализируются объекты класса в C++?
2. Может ли у класса быть несколько конструкторов?
3. Чем конструктор копирования отличается от других конструкторов класса?
4. Может ли конструктор копирования вызываться неявно?
5. Когда необходимо явно определять конструктор копирования для класса?
6. Как можно задать преобразование величины целого типа в объект нашего класса?
7. Как можно запретить использовать конструктор для преобразования типа?
8. Может ли у класса быть несколько деструкторов?
9. Когда вызывается деструктор класса?
10. Каково назначение деструктора?

Литература

1. Brian W. Kernighan, Dennis M. Ritchie. "The C programming language". Second edition – Prentice Hall. – 1988, 272 p.

2. Стэнли Б. Липпман. "Язык программирования C++. Вводный курс". – М.: Вильямс. – 2007, 896 с.

3. Г. Шилдт. "Самоучитель C++". – СПб.: БХВ-Петербург. – 2001, 688 с.

4. К.Арнольд, Дж.Гослинг, Д.Холмс. "Язык программирования Java". 3-е изд. – М.: Вильямс. – 2001, 624 с.