Динамично заделяне на памет. Деструктори. Оператор =.

гл.ас., д-р. Нора Ангелова

ОПЕРАТОР NEW/DELETE

- Създаване и разрушаване на динамично заделен масив от обект/и.

Пример:

```
point2 * dynamicPointsArr = new point2[10];
delete [10] dynamicPointsArr;
```

Как ще се разрушат обектите?

- Деструкторът ще се извика 10 пъти.

* delete [size]- they provide an optimization point for custom implementations:
they are called with the same *size* argument used in the call to the corresponding operator new.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3536.html

Деструктор

- Носи името на класа.
- Пред името стои знакът ~.
- Извиква се при разрушаване на обекти.
  - Разрушаване на обект чрез оператора delete
  - Излизане от блок, в който е бил създаден обект на класа
- Извикват се в обратен ред на конструкторите.

```
~point2() {
   //…
}
```

Ако конструкторът или някоя член-функция реализира динамично заделяне на памет за някоя член-данна, използването на деструктор е задължително, тъй като в този случай той трябва да освободи заетата памет.

ДЕСТРУКТОР

- **Излизане от блок, в който е бил създаден обект на класа**

```
class B {
  private:
    int b;
  public:
    B(int bData = 1){
        b = bData;
        cout << "B(" << bData << ")" << endl;
    }

    ~B() {
        cout << "~B" << b << endl;
    }
};

void print () {
  B test1;
}

int main() {
    print();
    return 0;
}
```

Резултат:
B(1)
~B1

ДЕСТРУКТОР

- **Разрушаване на обект чрез оператора delete**

```
class B {
  private:
    int b;
  public:
    B(int bData = 1){
        b = bData;
        cout << "B(" << bData << ")" << endl;
    }

    ~B() {
        cout << "~B" << b << endl;
    }
};

int main() {
    B * pointer = new B;
    delete pointer;

    return 0;
}
```

Резултат:
B(1)
~B1

ОПЕРАТОР =

- Извиква се при присвояване на обекти от съответния клас (не е конструктор и не създава обект).
- Връща className&
- Приема като параметър className const &

```
point2& point2::operator=(point2 const & p) {
    …
    return *this;
}
```

ОПЕРАТОР =

```
<име_на_клас>& <име_на_клас>::operator= (<име_на_клас> const & r) {
   if (this != &r) {
     // 1. освобождаване на динамичната памет на
     // компонентите на обекта, сочен от указателя this,
     // ако такава е отделена;
     // 2. копиране на компонентите на r в съответните
     // компоненти на обекта, сочен от указателя this
   }
   return *this;
}
```

Канонична форма на клас

канонична форма или голяма четворка на класа

=

конструруктор по подразбиране +

деструктор +

конструктор за присвояване +

операторна функция за присвояване

Каноничната форма е задължителна при класове, притежаващи член-данни, разположени в динамичната памет.

КОНТРОЛНА 3 - 1

```
class A {
  private:
    int a;
  public:
    A(int aData = 1) {
       a = aData;
       cout << "A(" << aData << ")" << endl;
    }
    A(A const & obj) {
       a = obj.a;
       cout << "Copy A(" << obj.a << ")" << endl;
    }
};

class B {
  private:
    int b;
    A objA;
  public:
    B(int bData = 1, int aData = 0) {
        b = bData;
        cout << "B(" << bData <<"," << aData << ")" << endl;
        objA = A(10);
    }
    B(B const & obj) {
       b = obj.b;
       cout << "Copy B(" << obj.b << ")" << endl;
    }
};
```

```
int main() {
    B test1(1,3);
    B test2 = test1;
    return 0;
}
```

КОНТРОЛНА 3 - 1

```
class A {
  private:
    int a;
  public:
    A(int aData = 1) {
       a = aData;
       cout << "A(" << aData << ")" << endl;
    }
    A(A const & obj) {
       a = obj.a;
       cout << "Copy A(" << obj.a << ")" << endl;
    }
};

class B {
  private:
    int b;
    A objA;
  public:
    B(int bData = 1, int aData = 0) {
        b = bData;
        cout << "B(" << bData <<"," << aData << ")" << endl;
        objA = A(10);
    }
    B(B const & obj) {
       b = obj.b;
       cout << "Copy B(" << obj.b << ")" << endl;
    }
};
```

```
int main() {
    B test1(1,3);
    B test2 = test1;
    return 0;
}
```

Резултат:

A(1)
B(1,3)
A(10)
A(1)
Copy B(1)

КОНТРОЛНА 3 - 2

```
class A {
  private:
    int a;
  public:
    A(int aData) {
       a = aData;
       cout << "A(" << aData << ")" << endl;
    }
    A(A const & obj) {
       a = obj.a;
       cout << "Copy A(" << obj.a << ")" << endl;
    }
};

class B {
  private:
    int b;
    A objA;
  public:
    B(int bData = 1, int aData = 0) {
        b = bData;
        cout << "B(" << bData <<"," << aData << ")" << endl;
        objA = A(10);
    }
    B(B const & obj) {
       b = obj.b;
       cout << "Copy B(" << obj.b << ")" << endl;
    }
};
```

```
int main() {
    B test1(1,3);
    B test2 = test1;
    return 0;
}
```

КОНТРОЛНА 3 - 2

```
class A {
  private:
    int a;
  public:
    A(int aData) {
       a = aData;
       cout << "A(" << aData << ")" << endl;
    }
    A(A const & obj) {
       a = obj.a;
       cout << "Copy A(" << obj.a << ")" << endl;
    }
};

class B {
  private:
    int b;
    A objA;
  public:
    B(int bData = 1, int aData = 0) {
        b = bData;
        cout << "B(" << bData <<"," << aData << ")" << endl;
        objA = A(10);
    }
    B(B const & obj) {
       b = obj.b;
       cout << "Copy B(" << obj.b << ")" << endl;
    }
};
```

Резултат:
Грешка

```
int main() {
    B test1(1,3);
    B test2 = test1;
    return 0;
}
```