ДЕСТРУКТОРИ. ОПЕРАТОР =. ДИНАМИЧНО ЗАДЕЛЯНЕ НА ПАМЕТ.

гл.ас., д-р. Нора Ангелова

CONST

- Четем от дясно на ляво

към е

```
int * const tempVariable; // не може да се променя адресът
```

към е

```
int const * tempVariable2; // не може да се променя ст/стта
```

към е

```
int const * const tempVariable2; // не се променят и двете
```

CONST

- Четем от дясно на ляво

```
int & const tempVariable; // не указва влияние
```
(int: към; & const: е)

```
int const & tempVariable2; // не може да се променя ст/стта
```
(int const: към; &: е)

```
int const & const tempVariable2; // не може да се променя ст/стта
```
(int const: към; & const: е)

CONST

```
int main()
{
    int a = 5;
    int * aPointer = &a;

    int b = 10;
    aPointer = &b;

    cout << *aPointer;

    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int * const aPointer = &a;

    int b = 10;
    aPointer = &b;

    cout << *aPointer;

    return 0;
}
```

Грешка

CONST

```
int main()
{
    int a = 5;
    int * const aPointer = &a;

    int b = 10;
    *aPointer = b;

    cout << *aPointer;

    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int const * aPointer = &a;

    int b = 10;
    *aPointer = b;
    cout << *aPointer;

    return 0;
}
```

Грешка

CONST

```
int main()
{
    int a = 5;
    int const * aPointer = &a;

    int b = 10;
    aPointer = &b;

    cout << *aPointer;

    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int const * const aPointer = &a;

    int b = 10;
    aPointer = &b; Грешка
    *aPointer = b; Грешка

    cout << *aPointer;

    return 0;
}
```

CONST

```
int main()
{
    int a = 5;
    int const * const aPointer = &a;

    int b = 10;
    aPointer = &b; Грешка
    *aPointer = b; Грешка

    cout << *aPointer;

    return 0;
}
```

CONST

```
int main()
{
    int a = 5;
    int const * const aPointer = &a;

    a = 10;

    cout << *aPointer;
    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int & aReference = a;

    int b = 10;
    aReference = b;

    cout << aReference;

    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int const & aReference = a;

    int b = 10;
    aReference = b;

    cout << aReference;

    return 0;
}
```

Грешка

CONST

```
int main()
{
    int a = 5;
    int const & aReference = a;

    a = 10;
    cout << aReference;

    return 0;
}
```

Резултат:
10

CONST

```
int main()
{
    int a = 5;
    int & const aReference = a;

    int b = 10;
    aReference = b;

    cout << aReference;

    return 0;
}
```

Резултат:
10

CONST

```
const int N;
```

Грешка - не може да се инициализира

КЛАС

```
class A
{
  private:
    int a = 5;

  public:
    ...
};
```

Грешка - памет се заделя при създаването на обект

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;
    }
};
```

Резултат:
Грешка

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

Динамични обекти

- Памет – програмен стек, област на динамичните данни (хийп)
  - програмен стек – кратковременна памет. Елементите му се наричат стекови рамки.
  - хийп – не се свързва с имена на променливи. Работи се с указатели.

ДИНАМИЧНИ ОБЕКТИ

- Структури от данни, за които операциите включване (добавяне) и изключване на елемент не са допустими, се наричат **статични**, в противен случай – **динамични.**

Динамични обекти

- Заделянето на памет по време на компилация се нарича **статично заделяне на памет.**
- Заделянето на памет по време на изпълнение на програмата се нарича **динамично заделяне на памет.**

Динамични обекти

- Създаване на динамични обекти – оператор new
- Разрушаване на динамични обекти – оператор delete

Паметта за динамичните променливи се заделя от оператора new.

Заделената по този начин памет остава свързана със съответната променлива докато не се освободи явно от програмиста.

ОПЕРАТОР NEW

- Използва се за създаване на динамични обекти.
- Заделя в хийпа необходимата памет и връща указател към нея.
- Пази се докато е необходимо.

new <име_на_тип>;

new <име_на_тип> [size];

new <име_на_тип> (<инициализация>);

<име_на_тип> - име на стандартен тип или име на клас.

size – израз с произволна сложност, която може да се преобразува до цял с положителна стойност. Показва броя на компонентите от типа, за които да се задели памет в хийпа и се нарича **размерност.**

<инициализация> - израз от тип <име_на_тип> или инициализация на обект според синтаксиса на конструктора на класа, ако <име_на_тип> е име на клас.

ОПЕРАТОР NEW

- Ако няма достатъчно място в хийпа, операторът new връща NULL или изключение.
- Проверка дали паметта е заделена

Пример:

```
int * dynamicArr = new int[10];
int * dynamicVar = new int(2+5*5);
rat * r = new rat(1,2);
rat** rPointers = new rat*[5];
```

ОПЕРАТОР DELETE

- Разрушава обекта, адресиран от указателя, като паметта, която заема този обект се освобождава.
- Ако обектът, адресиран от указателя, е обект на клас, първо се извиква деструкторът на класа и след това се освобождава паметта.

delete <указател_към_динамичен_обект>;
delete [] <указател_към_динамичен_обект>;
delete [size] <указател_към_динамичен_обект>;

<указател_към_динамичен_обект> - указател към диначен обект, създаден чрез оператора new

Пример:

```
int * dynamicVar = new int(2+5*5);
delete dynamicArr;
```

Оператор delete

- Разрушаване на масив

Пример:

```
int * dynamicVar = new int[10];
delete [] dynamicArr;
```

- Ако масивът съдържа в себе си указатели, които адресират динамично заделена памет, първо масивът трябва да бъде обходен и за всеки негов елемент да бъде извикан операторът delete.

Оператор delete

- Разрушаване на масив

Пример:

```
int * dynamicVar = new int[10];
delete [] dynamicArr;
```

- Ако масивът съдържа в себе си указатели, които адресират динамично заделена памет, първо масивът трябва да бъде обходен и за всеки негов елемент да бъде извикан операторът delete.

Оператор delete

- Разрушаване на масив от обекти

Пример:

```
rat * dynamicVar = new rat[10];
delete [10] dynamicArr;
```

Деструкторът ще се извика 10 пъти.

Динамични обекти

Задача

Да се създаде масив от указатели към цели числа (размерът се определя от променлива).

Нека стойностите да съответстват на индекса на елементите.

Да се изведат стойностите на екрана.

ДИНАМИЧНИ ОБЕКТИ

```
int size = 10;
int** arr = new int*[size];

if (!arr ) {
  cout << "Not enough memory!\n";
  return 1;
}

for (int i=0; i<size; i++) {
  arr[i] = new int;

  if (!arr[i] ) {
    cout << "Not enough memory!\n";
    return 1;
  }

  *arr[i] = i;
}

for (int i=0; i<size; i++) {
  cout << *arr[i] << " ";
}

for (int i=0; i<size; i++) {
  delete arr[i];
}

delete [] arr;
```

ASSERT

- `#include <cassert>`

```
assert(<твърдение>)
```

Ако е истина, изпълнението на програмата продължава, а ако не е истина, изпълнението на програмата завършва със съобщение за грешка, включващо мястото на възникване на грешката в кода (име на файл и номер на ред), както и самия текст на условието, което не е било изпълнено.

ДИНАМИЧНИ ОБЕКТИ

```
int** arr = new int*[size];

if (!arr ) {
  cout << "Not enough memory!\n";
  return 1;
}

-----------------------------------

assert(arr != NULL);
```

OPERATOR =

- Връща className&
- Приема като параметър className const &

```
rat& stack::operator=(rat const& r) {
    …
    return *this;
}
```

- Извиква се при присвояване на обекти от съответния клас
  (не е конструктор и не създава обект)