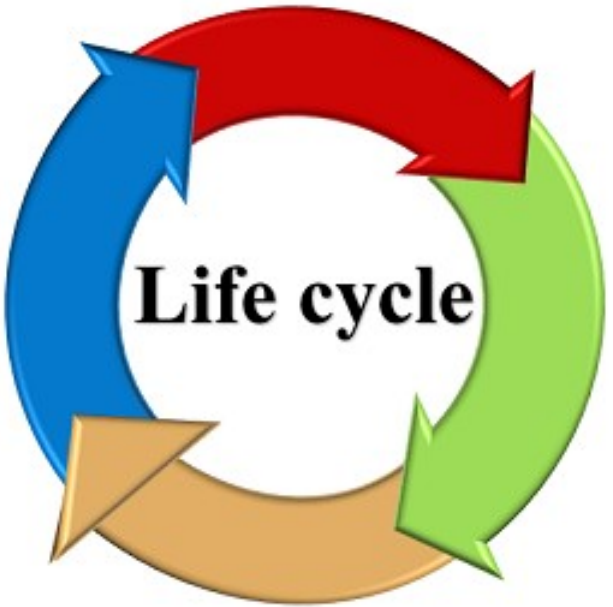
ООП Семинар 4

Предефиниране на оператори.
Жизнен цикъл на обекта!



Предефиниране на оператори

- Представете си следното нещо. Имате два обекта инстанции на ваш клас. Колко яко ще е да можем да ги съберем ей така с +. Искаме го, мечтаем си, а накрая ….

```
martin@martin-PC:~/Desktop$ g++ test.cpp
test.cpp: In function ‘int main()’:
test.cpp:18:20: error: no match for ‘operator+’ (operand types are ‘MyInt’ and ‘MyInt’)
     std::cout << a + b;
                  ~~^~~
```

Отидоха ни мечтите за бързо и лесно

- Реално тази грешка идва от там, че компилаторът идея си няма как да събере тези неща. Ето реакцията му

Така, а как да му кажем?

- За целта трябва да предефинираме операторът за дадените операнди.
- Синтаксис: <какъв ще е резултатът> operator<оператор>(<операнд>)

Пример

```
class MyInt
{
    private:
        int a;

    public:
        int getA() const;
        MyInt operator+(MyInt b);
        MyInt(int _a);
};
```

```
MyInt::MyInt(int _a)
{
    a = _a;
}
MyInt MyInt::operator+(MyInt b)
{
    MyInt c(a + b.a);
    return c;
}
int MyInt::getA() const
{
    return a;
}
```

Кратко въведение

- Погледнете телефоните, които така добре държите, а не внимавате тука…
- И те и имат жизнен цикъл
- Те биват проектирани от инженерите и бива създаден еталон
- После се пускат в масово производство и се правят много копия
- Всяко от тези копия изпълнява вярно службата си докато накрая не се развали.
- Същото е и с обектите

Жизнен цикъл на обекта

- Той се създава като се прави инстанция на някой клас.
- Изпълнява си службата
- По някое време трябва да се унищожи

Това пък за какво ни е?

- Много просто. До момента ние не се грижехме за това, защото компилаторът го правеше вместо нас с някакви методи по подразбиране, които са му заложени. Те са прекрасни докато не се намеси динамична памет.

Голямата 4-ка aka Канонично представяне

- Конструктор по подразбиране
- Конструктор за копиране
- Оператор за присвояване
- Деструктор
- Това е абсолютният минимум от курса по ООП. Ако не ги знаете няма З!!!

Конструктор по подразбиране

- Служи за да инициализира динамичната памет

```
const size_t DEFAULT_CAPACITY = 10;

class MyVector
{
    private:
        int *arr;
        size_t size;
        size_t capacity;

    public:
        MyVector();
}
```

```
MyVector::MyVector()
{
    size = 0;
    capacity = DEFAULT_CAPACITY;
    arr = new int[capacity];
}
```

Deep vs Shallow Copy

- При Shallow Copy ние копираме само референциите от стейта, но те продължават да сочат на едно и също място.
- При Deep Copy правим пълно копие на всичко, като за динамичната памет заделяме ново парче и правим копието там за да са на различни места и да няма неприятни последици.
- Конструктурът за копиране по подразбиране прави Shallow copy
- Същото се отнася и за оператора за присвояване по подразбиране

Конструктор за копиране

- Извиква се, **когато се създава инстанция**, която е копие на даден обект.
- Синтаксис <Име>(const <Име>& <идентификатор>)

Пример

```
MyVector::MyVector(const MyVector& a)
{
    size = a.size;
    capacity = a.capacity;
    arr = new int[capacity];
    for(int i = 0; i < size; i++)
    {
        arr[i] = a.arr[i];
    }
}
```

Указател this

- Всеки обект пази в себе си указател към себе си. Той се казва this
- Този указател си има приложения, като например ако данните във функция имат същото име като полетата, тогава за да ги различим пишем this→име = име

Пример

```
class MyInt
{
    private:
        int a;

    public:
        int getA() const;
        MyInt operator+(MyInt b);
        MyInt(int _a);
};
MyInt::MyInt(int a)
{
    this->a = a;
}
```

Оператор за присвояване

- Извиква се, когато **на вече създадена инстанция** на клас пробваме да присвоим стойността на друга инстанция.
- Съответно в този оператор трябва да изтрием паметта, която е заделена, да заделим нова и да направим Deep Copy на инстанцията, която присвояваме.

Пример

```
MyVector& MyVector::operator=(const MyVector& a)
{
    if(this != &a)
    {
        delete[] arr;
        size = a.size;
        capacity = a.capacity;
        arr = new int[capacity];
        for(int i = 0; i < size; i++)
        {
            arr[i] = a.arr[i];
        }
    }
    return *this;
}
```

WTF is this

- Може би ви се наби проверката if(this != &other)?
- Тя се прави от съображения да не изтрием паметта, която ще копираме в случай, че някой каже a = a;

Деструктор

- Стигнахме до края на жизнения цикъл.
- Деструкторът по подразбиране ще изтрие само указателя и динамичната памет ще увисне.
- За целта трябва да направим нещо, което като обектът стигне края на жизнеият си цикъл да се извика и да изчисти динамичната памет.
- Синтаксис ~<Име>()

Пример

```
MyVector::~MyVector()
{
    delete[] arr;
}
```

Задача 1

Да се дефинира клас Word, описващ дума, съставена от не повече от 20 символа от тип char. Класът да съдържа следните операции:

- [5 т.]оператор [] за намиране на i-тия пореден символ в думата
- [10 т.]оператори + и += за добавяне на един символ в края на думата. Ако думата вече има 20 символа, операторите да нямат ефект
- [10 т.]оператори < и == за сравнение на думи спрямо лексикографската наредба
- [10 т.]подходящи конструктори

Задача 2

Да се дефинира клас Sentence, описващ символен низ, състоящ се от произволен брой символи от тип char. Класът да поддържа следните операции

- [15 т.]функция addWord за добавяне на дума (обект от клас Word) към края на изречението
- [10 т.]конструктор за копиране
- [10 т.]други подходящи контруктори
- [10 т.]оператор за присвояване
- [5 т.]деструктор

Задача 3

Да се имплементира клас Polynomial, представляващ полином от произволна степен. Добавете:

- конструктор по подразбиране, копи конструктор, оператор равно и деструктор
- метод за отпечатване на полинома на стандартния изход
- operator() - по подадено число пресмята полинома
- метод за добавяне и премахване на коефициент
- оператор за събиране на два полинома

Задача 4

- Да се имплементира клас Task, представляващ една задача за изпълнение. Всяка задача се характеризира с име с произволна дължина и време(в часове) за изпълнение. Добавете:

- конструктор по подразбиране, копи конструктор, оператор равно и деструктор
- метод за отпечатване на задачата на стандартния изход
- оператор за сравнение на две задачи по име

- Имплементирайте и клас, описващ списък от задачи. Добавете:
- конструктор по подразбиране, копи конструктор, оператор равно и деструктор
- метод за отпечатване на всички задачи в списъка
- метод за добавяне на задача
- метод за премахване на задача