

15 reasons why

(записки от лекции по ООП)

**Темите тук са в
поредността, в която
сме ги взимали на
лекциите при
Димитриев, а не както в
конспекта. Конспект.**

ЛЕКЦИЯ 1.

???

- Пространства от имена (Namespace).
- Енумерации, структури и обединения.
- Видове енумерации и разлики
- Работа с инстанции: Инициализация, достъп до елементите, влагане, работа с функции, работа с масиви.
- Размер на обекти/инстанции.
- Подравняване и отместване.
- Endianness и проверка за big/little endian
- Линк към темата в [гитхъба на Никола](#)
- Линк към темата в [гитхъба на Ангел](#)

Имена I

Namespace. Члог би членовете дефинирани от по-редицата.

1. Namespace - пространство от имена

- инструмент за издаване на копии на имена

Пример:

namespace ns-name {

 декларации

 { } ns-name::f()

 оператор за резултат

! namespace може да се викат

using namespace ns-name;

{

! членовете на ns-name могат да се назват
без да имам ns-name:: в началото на този scope

using ns-name::f();

{

 f() → можем да назовем без да извикваме
 оператора за резултат

 g() x → не можем да назовем, тъй като
 не сме близки до f(), навсякътно е
 да напишем, ns-name::g()?

! пространствата от имета могат да предизвикват конфликт на имета

Пример:

```
namespace A{
```

```
void f(){};
```

```
}
```

```
namespace B{
```

```
void f(){};
```

```
}
```

```
using namespace A;
```

```
using namespace B;
```

f() → конфликт при извикването на f()

Структури - тип, който е рестричтиран до дадени от стойности, които характеризират специфично дефинирани константи.

Пример:

enum

```
enum Color{
```

```
red,
```

```
blue,
```

```
green
```

```
} ;
```

```
int main(){
```

```
Color myColor=Color::green;
```

```
if (myColor == green)
```

```
{
```

```
std::cout << "text";
```

color t; → описващ на член тело

! от C++ 20 ↑

```
using enum color;
```

```
color t=green;
```

?

Как членът стои зад стойностите?

Пример:

enum X {

a,

b,

c

}

} енумератори

Стойности:

a = 0

b = 1

c = 2

enum Y {

a = 11

b

c = 20

d = 10

e

f

g = e + f

h

i

} енумератори

Стойности:

a = 11

b = 12

c = 20

d = 10

e = 11

f = 12

g = 23



- enum е unscooped, член-датите му се третират като "глобални променливи" т.е. е возможен конфликт на имета извън scope-а им.

enum X

{

red;

blue

green

yellow

enum Y {

red,

purple

}

конфликт на имета



Проблем при употребата на enum - явно преобразуване на стойностите във вид числа.

Пример:

```
enum Color{
```

```
    red,
```

```
    blue
```

```
}
```

```
enum Fruit{
```

```
    apple,
```

```
    orange
```

```
}
```

```
int main(){
```

```
    Color myColor = Color::Blue;
```

```
    Fruit myFruit = Fruit::orange;
```

```
    if(myColor == myFruit)
```

```
{
```

Свойства:

red=0

blue=1

apple=0

orange=1

ме се използва, когато трябва да се
} до неочаквано поведение, кое да предизвика
тъй като константи от различен тип

трябва да се сравнят

Enum Class

- ! НИМА явно преобразуване от енумератор към члено число!

Пример:

```
if(myColor == myFruit)  
{  
}
```

→ НИМА да се компилира този кодо сравниване, где идатици от различни класове, т.е. съвсем сравниване где притежани променливи от различни типове.

enum class A

```
{  
    x,  
    y  
}
```

int n = A::x → НИМА да се компилира, този кодо название enum class, който има явно разграничение между енумератор и члено число.

- ! размер на променлива от тип enum/enum class, която размера на някой членовиден тип, в който могат да се записват стойностите на енумераторите.

- В некои компилатори идент и следното изисаване: $\leq \text{sizeof}(\text{int})$

Структури - посредствителност от идент, която се **запазва** в определен рег.

Пример:

struct Point

```
int x;
int y;
};
```

Структура \rightarrow идентификатор

Класове \rightarrow обекти

! Важна разлика:

point p;

point p {3,7}; \rightarrow създаване на стойности

point* ptr = new point; (initializer list)

\hookrightarrow зарезане на инициализацията за point

! goction go enenfritute

Пример:

point p;

p.x = 10;

p.y = 11;

cout << (p.x);

point* ptr = new point;

(*ptr).y = 7;

(*ptr).y = 10;

ptr->x = 7;

ptr->y = 10;

delete ptr;

} експансионно
значение

! Погавате във функции: $f(\text{point } p)$ $\text{g}(\text{point}^*\& p)$ $\text{k}(\text{point}^* p)$

копие

!! Винаги, ако нощем, че използваме да погаваме по копие!

? • как да се ползват референции?

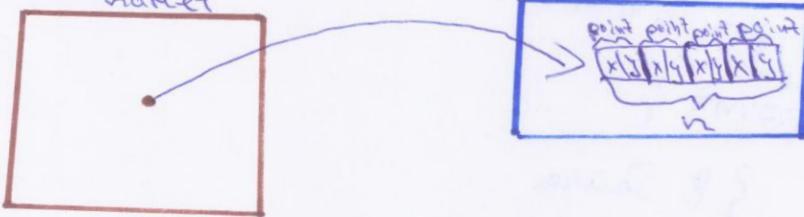
!! Винаги, ако нощем, че спаравме `const`, ако не правим промени по обекта/инстанцията!

? • как да се ползват константи?

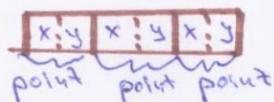
! масиви от инстанции

Пример:

`point* arr = new point[n];`

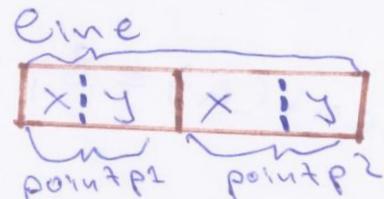


!! `delete [] arr;`
`point arr[3];`



! ВЛАГАТЕ НА ИНСТАНЦИИ

`Struct Line{
 point p1;
 point p2;
};`



`line myLine {x1,y1,x2,y2}` } експансионен
`line myLine{1,2,3,4}` } замене

✗

Пример за лоша абстракция:

struct Triangle {

```
int x1;  
int y1;  
int x2;  
int y2;  
int x3;  
int y3;
```

}

✓

Пример за добра абстракция:

struct Triangle {

```
point p1;  
point p2;  
point p3;
```

}

!

Размер и представление в паметта

Пример:

struct point {

```
int x;  
int y; } 8 байта
```

};

sizeof(MyStruct) ≥ \sum sizeof(member)



member → член на MyStruct

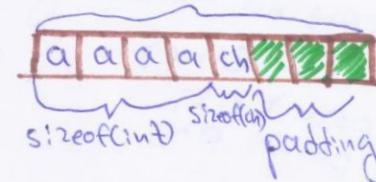
struct A {

int a; → 4 байта

char ch; → 1 байт

};

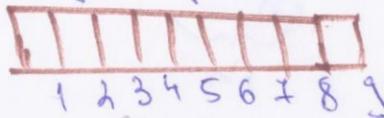
sizeof(A) = 8; →





• Всеки тип си има Alignment requirement.
регистрата между дългото и последователният адрес, който можем да "разделим" променлива от този тип.

Примери правилни:



нosiаване a на нодуц, (агрес)
които се деля на 4.

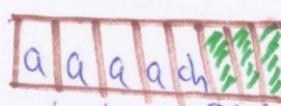
int a = 12; → 4 Bytes (0, 4, 8, 12)
alignof(int) → 4

Пример:

struct X {

 int a; → 4 Bytes

 char ch; → 1 Byte } →
 { представяе
 6 паметта



които се деля на 8.



Големината на структурата трябва да се дели на така-така "применив тип".
В ида.



Зад-променлива трябва да е същ агрес, който се дели на големината ѝ.



Променливите са декларирани в реда, в който са декларирани в структурата.

Пример:

```
struct A { sizeof: 8
    int a; alignof: 4
    { char ch;
}
```

struct A2 {

char ch; sizeof: 8
int a; alignof: 4
}

struct A3 {

char ch; sizeof: 12
int a; alignof: 4
char ch2;

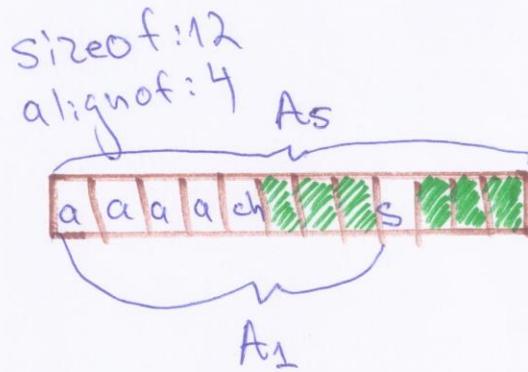
struct A4 {

char ch; sizeof: 8
char ch2; alignof: 4
int a;



struct A5

{
A1 obj;
char s;
}

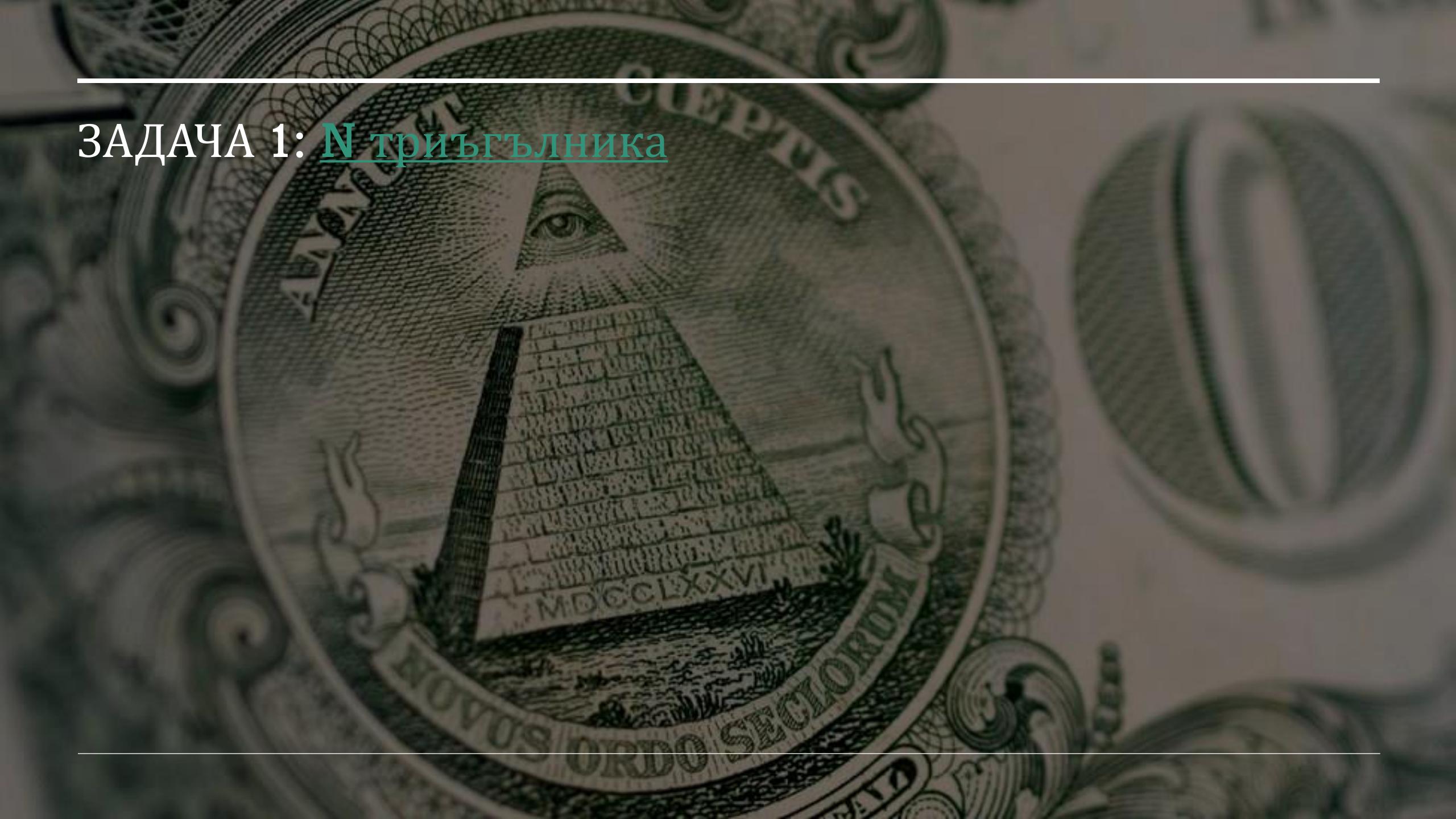


struct A6

{
char ch; → 1 byte
int arr[5]; → 5 × 4 bytes
char ch2; → 1 byte
unsigned int j; → 8 bytes

sizeof: 40
alignof: 8

ЗАДАЧА 1: N триъгълника



```
1 #include <iostream>
2
3 namespace Points
4 {
5     struct Point
6     {
7         int x = 0;
8         int y = 0;
9     };
10
11     void readPoint(Point& point)
12     {
13         std::cin >> point.x;
14         std::cin >> point.y;
15     }
16
17     double getDist(const Point& p1, const Point& p2)
18     {
19         int dx = p1.x - p2.x;
20         int dy = p1.y - p2.y;
21
22         return sqrt(dx * dx + dy * dy);
23     }
24
25     void printPoint(const Point& point)
26     {
27         std::cout << "(" << point.x << " " << point.y << ")";
28     }
29 }
30
```

```
31 namespace Figures
32 {
33     using namespace Points;
34     struct Triangle
35     {
36         Point p1;
37         Point p2;
38         Point p3;
39     };
40
41
42     void readTriangle(Triangle& triangle)
43     {
44         readPoint(triangle.p1);
45         readPoint(triangle.p2);
46         readPoint(triangle.p3);
47     }
48
49
50     double getArea(const Triangle& triangle)
51     {
52         double sideA = getDist(triangle.p1, triangle.p2);
53         double sideB = getDist(triangle.p2, triangle.p3);
54         double sideC = getDist(triangle.p1, triangle.p3);
55
56         double halfPer = (sideA + sideB + sideC) / 2;
57
58         return sqrt(halfPer * (halfPer - sideA) * (halfPer - sideB) * (halfPer - sideC));
59     }
60 }
```

```
61 void sortTrianglesByArea(Triangle* triangles, size_t N)
62 {
63     double* areas = new double[N];
64     for (int i = 0; i < N; i++)
65         areas[i] = getArea(triangles[i]);
66
67     for (int i = 0; i < N - 1; i++)
68     {
69         int minAreaTriangleIndex = i;
70
71         for (int j = i; j < N; j++)
72         {
73             if (areas[j] < areas[minAreaTriangleIndex])
74                 minAreaTriangleIndex = j;
75         }
76
77         if (minAreaTriangleIndex != i)
78         {
79             std::swap(triangles[i], triangles[minAreaTriangleIndex]);
80             std::swap(areas[i], areas[minAreaTriangleIndex]);
81         }
82     }
83     delete[] areas;
84 }
85
86 void printTriangle(const Triangle& triangle)
87 {
88     printPoint(triangle.p1);
89     printPoint(triangle.p2);
90     printPoint(triangle.p3);
91     std::cout << std::endl;
92 }
93 }
```

ЛЕКЦИЯ 2

- Поток. Стандартни потоци.
- Текстови файлове.
- Йерархия на потоците.
- Интерфейс на потоци.
- Потоци за вход/изход от файл.
- Режими на работа. Флагове на състоянията на потока.
- Позициониране във файл.
- Линк към темата в [гитхъба на Никола](#)
- Линк към темата в [гитхъба на Ангел](#)

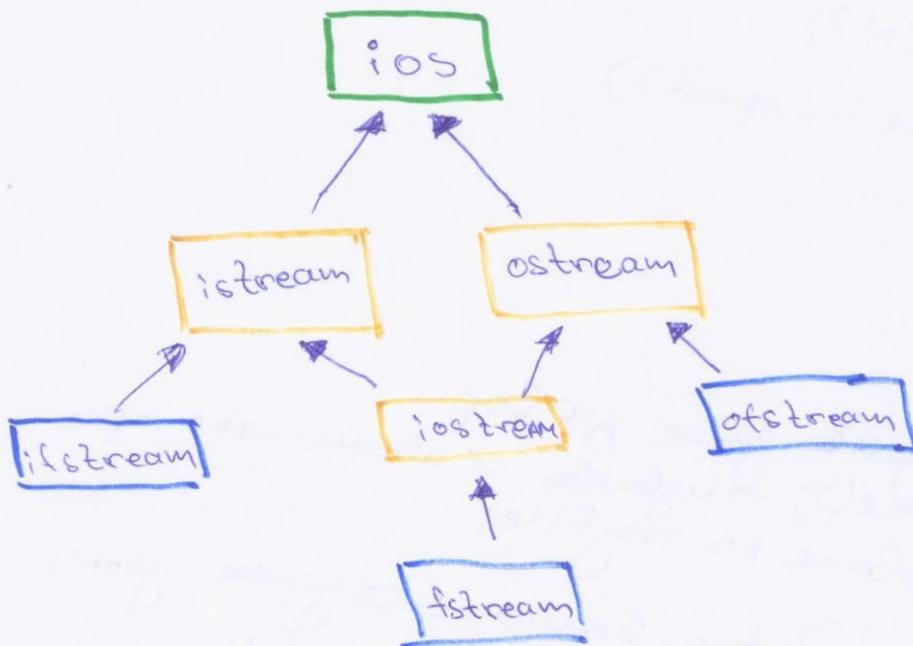
Лекция II

Формат \rightarrow ненавязчивость от данных, наоборот
к определению потока

$\text{bxog} \gg>$  $\ll<\text{uzxog}$

Пример:
`cout`
`cin`
`cerr`

Примерная иерархия:



! Типы для ввода ostream

!!! обратите внимание $\ll \rightarrow \text{cout} \ll \dots \ll \dots$,

!!! методы $\rightarrow \text{put}(\text{char ch})$
 $\rightarrow \text{write}(\text{const char* str}, \text{size_t bytes})$

Пример:

`ofstream os << <обект>`
 ↓ ↓ ↓
 метод оператор обект
 OS

`ofstream` → метод за писане във файл

Пример:

```
ofstream myFile("име на файл", режим на работа)  
myFile << Задача<< endl;  
myFile.put('a');  
if (!myFile.is_open())  
{  
}  
}
```



Помогай трябва да се затвори
`myFile.close();` → извиква се автоматично при
използване на `myFile`;



`myFile` има буфер, който се съгръща автоматично
с използването → `myFile.flush();`



`close()` извиква `flush()`;

Позициониране

! `put` указател

→ `file` га започва

→ следваща свободна позиция

1	7	3	
---	---	---	--

`↑
put`

!! `tellp()` → `file` е `put` указателят

!!! `seekp(idx)` → `file` га смигне `put` указателят

!!! `seekp(offset, direction)` → `file` га смигне
път указателят,
спрямо мястото и посока

! `direction:`

!! `:ios::beg` → начало на файла

!! `:ios::end` → края на файла

!!! `:ios::cur` → текущата позиция във файла

Пример:

`myFile.seekp(1, :ios::beg)` → `put` указателът е
пъти с 1 "offset" позиция
от началото на файла

! Позиционирането не се използва при `"out"`, `"in"`.

`ifstream` → възможност за нуцане Bob отваря (exog)

Пример:

`ifstream ("---", речник за падора)`

↑

file
`if (!file.isopen()) {
 cout << "Error!"
}`



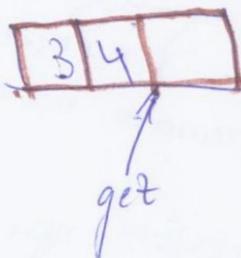
! Намогък за выход ifstream

!! оформителен възможност >> обработка

!! Неоформителен възможност

`→ get()` } → глобални
`→ read()` } → обработка

Тип `ifstream` → get упражнение

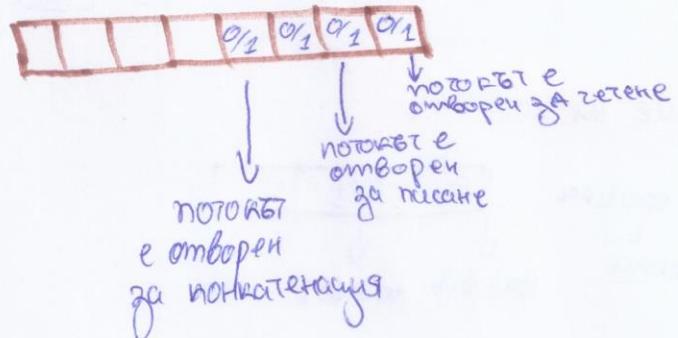


!! `seekg(idx)` → премества get указателя
на позиция idx

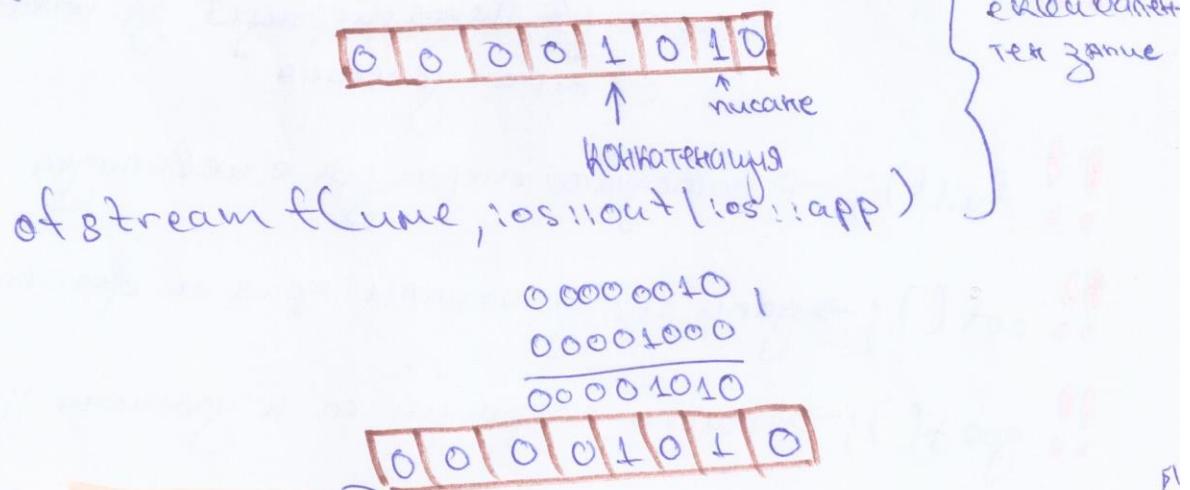
!! `seekg(offset, direction)` → синтаксис

!! `tellg()` → връща позицията
на междууказания символ
в потока за четене

Речим на работе \rightarrow чете и създава
 ostream (line, Речим са работа)



Пример:
 ofstream f (line, 10)



Видове речими на работа

- 1- ios::in \rightarrow покрай е отворен за четене
- 2- ios::out \rightarrow покрай е отворен за писане
- 4- ios::ate \rightarrow покрай е отворен и насока рут указател на края. Позицията винаги е на производствена позиция.
- 8- ios::app \rightarrow отваря потока и насока рут указателя на края на файла. Не позволява създаване на производствена позиция.
- 16- ios::trunc \rightarrow ако даденото съдържание, то се изтрие съдържанието
- 32- ios::binary \rightarrow "отваря файла в двоичен формат"

64 - `ios::nocreate` → отваря джанга, само ако
съществува

128 - `ios::noneplace` → отваря джанга, само ако
не съществува

Състояния на поток

0 - нормална	0/1	0/1	0/1
1 - грешка	↓	↓	↓

Bad bit tail bit eof

!!! `bad()`; → данни bad bit е валидни; Ако ги, ита запътна
информация. Нека операција за четене/писане
не е била успешна

!!! `fail()`; → последната операција е невалидна

!!! `eof()`; → данни сме достигнали края на джанга

!!! `good()`; → всички операции са извършени успешно

!!! `clear()`; → "изчисти" всички грешки; следващият
`good()` ще бъде успешна

Проведе се шестото пребояване на пеликаните в Югоизточна Европа

Agro.bg

🕒 01 June 2023, 16:11 🗂 606

ЗАДАЧА 2: Функция, която връща големината на файл; Функция,
която връща брой редове в текстов файл



```
1 #include <iostream>
2 #include <fstream>
3
4
5 size_t getFileSize(std::ifstream& file)
6 {
7     size_t currentPos = file.tellg();
8     file.seekg(0, std::ios::end);
9     size_t fileSize = file.tellg();
10    file.seekg(currentPos);
11    return fileSize;
12 }
13
14 int main()
15 {
16 }
```

```
1 #include <iostream>
2 #include <fstream>
3
4 unsigned getCharCountFromFile(std::ifstream& ifs, char ch)
5 {
6     size_t currentPosition = ifs.tellg();
7     ifs.seekg(0, std::ios::beg);
8
9     if (!ifs.is_open())
10        return 0;
11
12     unsigned int count = 0;
13
14     while (true)
15     {
16         char current = ifs.get();
17         if (ifs.eof())
18             break;
19
20         if (current == ch)
21             count++;
22     }
23
24     ifs.clear();
25     ifs.seekg(currentPosition);
26     return count;
27 }
28 unsigned getLinesCount(const char* fileName)
29 {
30     std::ifstream myFile(fileName);
31
32     if (!myFile.is_open())
33         return 0;
34
35     return getCharCountFromFile(myFile, '\n') + 1;
36 }
```

ЛЕКЦИЯ 3:

БУКВАР

- Работа с `fstream` - особености при отваряне на файл за четене и писане.
- Двоични файлове.
- Запазване на обекти в двоичен файл.
- Четене на обекти от двоичен файл.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

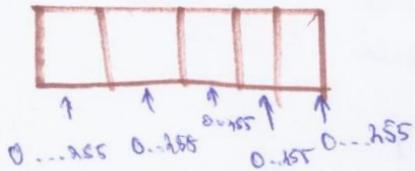
за първи клас



Глава III

Двоичные файлы → содержат в машинном виде
данные

Пример



Файл

ostream f (out.dat, ios::out | ios::binary)

f.write (указатель ROMБМ конко байта данных для записи)
(const char*) → запись в единицах байт

int a=16



Пример:

f.write (const char*)&a, sizeof(a));

Чтение

Пример:

int x;

ifstream it (file.dat, ios::in | ios::binary);

if.read (указатель ROMБМ загружаемой памяти, конко байта
(char*)
ga iporete)

if.read (char*)&x, sizeof(x));

! Выведение

!! write (- , -)
указатель на конк. | конк.
данные | байца

!! read (- , -)
указатель | конк.
FROM заг.ИМЕЕ | байца
| npozere

! Запись/чтение в объект/контейнер в файл

объект

Пример:

struct A { }

bool b;

int a;

{

A obj{false, 77};

ots.write(const char*)& ob, sizeof(A));

! Писане/четене на масив от обекти членувачи
от един обект.

Работа с fstream → В интерфейса му
има възможността
на ifstream и ofstream

```
ifstream("File.txt");
fstream("File.txt", ios::out);
```

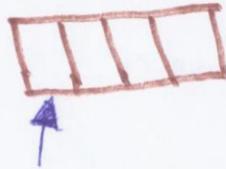
! ofstream → узход;

! ifstream → вход;

! fstream → вход + узход;

! ЗАНЕСВАТЕ НА СИМВОЛ В МЕЖДУ ОБА У
С ДРУГ СИМВОЛ. (ЗАГРАДА)
→ fstream-четене и пищене едновременно

! fstream има един указател



! Когато прилагаме входна операция след
изходна в fstream, предва да съдоврем
буферът.

!! flush();

!! seekp/tellp();

! Запись в стеке на одинарных кавычках, как это называется, "одинарные кавычки"

Пример:

```
struct Person {
```

```
    char* name;
```

```
    int age;
```

```
};
```

```
Person p;
```

```
p.age = 3;
```

```
p.name = new char [5];
```

```
strcpy(p.name, "Ivan");
```

```
;
```

```
delete[] p.name;
```

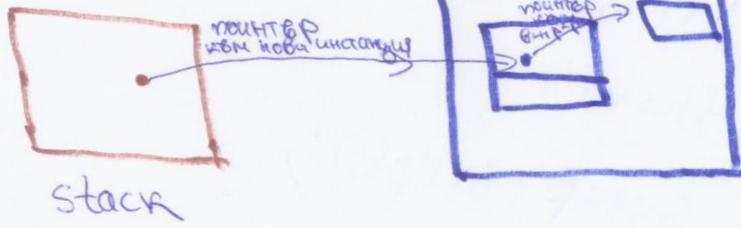
Пример:

```
Person* ptr = new Person;
```

```
;
```

```
delete[] ptr->name;
```

```
delete ptr;
```



Апаша посегнаха на малинов масив край Средноград

ЗАДАЧА 3: Пример за запазване/чтение на масив от обекти(от един тип) във файл;
Пример за заместване на символ с друг символ във файл (чтение и писане с fstream)

Вчера В Районно управление-Казанлък
17.12.2022 г. до 19.12.2022 г., от международни
била извършена кражба на 800 подготвени

Образувано е досъдебно производство

```
1 #include <iostream>
2 #include <fstream>
3
4
5 size_t getFileSize(std::ifstream& file)
6 {
7     size_t currentPos = file.tellg();
8     file.seekg(0, std::ios::end);
9     size_t fileSize = file.tellg();
10    file.seekg(currentPos);
11    return fileSize;
12 }
13
14 int main()
15 {
16     std::ifstream ifs("myfile.dat", std::ios::in | std::ios::binary);
17
18     if(!ifs.is_open())
19         return -1;
20
21     size_t fileSize = getFileSize(ifs);
22     size_t arrSize = fileSize / sizeof(int);
23     int* arr = new int[arrSize];
24
25     ifs.read((char*)arr, fileSize);
26
27     for (int i = 0; i < arrSize; i++)
28         std::cout << arr[i] << " ";
29
30     delete[] arr;
31     ifs.close();
32 }
```

```
1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 void replaceInFile(char ch, char ch2, fstream& file)
6 {
7     while (!file.eof())
8     {
9         char currentSymbol = file.get();
10
11         if (currentSymbol != ch)
12             continue;
13
14         file.seekp(-1, ios::cur);
15         file.put(ch2);
16         file.flush();
17     }
18 }
19
20 int main()
21 {
22     fstream file("treasureTrail.txt", ios::in | ios::out);
23
24
25     if(!file.is_open())
26     {
27         std::cout << "Error while opening the file!" << std::endl;
28         return -1;
29     }
30
31     replaceInFile('a', 'X', file);
32     return 0;
33 }
```



ages/年齢

8-12

4250

Tiananmen Square
Unknown Rebel

cont. **563** pcs/皮斯

Building Toy
建築玩具

ЛЕКЦИЯ 4:

- Указател `this`.
- Член-функции. Конструктори и деструктор.
- Извикване на конструктори и деструктори.
- Конвертиращи конструктори.
- Извикване на конструктори и деструктори при създаване масиви (статични и динамични).
- Модификатори за достъп.
- Абстракция.
- Капсуляция.
- `Mutable`.
- [Линк към гитхъба на Никола](#)
- [Линк към гитхъба на Ангел](#)



Меню IV

Слен-функции \rightarrow функции в метода на
клас/структурата

Пример:

```
struct Point {
```

```
    int x;
```

```
    int y;
```

```
};
```

```
struct Point {
```

```
    int x;
```

```
    int y;
```

```
    bool isInFirstQuadrant()
```

```
{
```

```
    return x >= 0 && y >= 0;
```

```
}
```

```
};
```

```
Point pt;
```

```
pt.isInFirstQuadrant;
```

! Слен функциите:

!! работят директно с члените гатните на обекта

!! извикват се от обект.

!! Компилаторът го преобразува до нормална
функция, с допълнителен параметър - указател
към обекта, от който се извиква.

Пример:

```
struct Point {  
    int x;  
    int y;  
    void Print();  
};  
cout << x << y;
```

Интерпретация
на компилятора

```
struct Point  
{  
    int x;  
    int y;  
};  
void print(Point this)  
{  
    cout << this.x << this.y;  
}
```

! Константные члены объектов → гарантируем, что
ни одна из переменных не меняется.

Пример:

```
struct A {  
    int n;  
    void f();  
    int i;  
};  
void printf() const {  
    cout << n;  
}  
};
```

void func(const A& ref)

{

ref.f() X → не се компилира, тъй като "f" не е константна функция - дължи се, а подаване на като константа

ref.printf(); ✓ → се се компилира, тъй като "printf" е константна функция.

struct A{

void f();

void g() const;

}

f(A* const this);

g(const A* const this);

!

2 специални кон-дължини

първи:

struct A{

int x;

int y;

}

int main(){
A obj{3,4};}

задействие на памет;
инициализация на
датите на обекта;

}

известяване за датите на
обекта; освобождаване на паметта;



конструктор

→ обявява, че се извиква при създаването на обект

- ! същото име като клас / структурата;
- ! не се означава единствено тук на връхните;
- ! конструктор без параметри се назва default (по подразбиране)
- ! Ако в класа / структурата не разпишем тук един конструктор, то компилаторът създава default-ен, без никакво генериране.

Пример:

struct A {

^{Член претенция}
 на компилатора

}

Struct A {
 A();
};

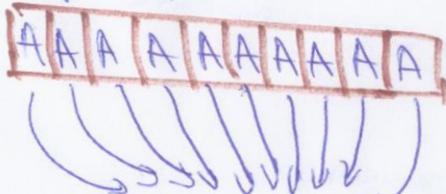
A obj; → def. конструктор

A obj(); → обявява;



Массив от инициализации

Пример: A arr[10];



def конструктор;



Конструкторы при возврате

Пример:

Struct A{

 A (int A);

};

Struct B{

 B(); } ; → создается default

};

Struct C{

 C(); } ; → создается default

};

Struct X{

 B obj1; } ; → вызывается констр. по умолчанию B;

 C obj2; } ; → вызывается констр. по умолчанию C;

 X(); } ; → создается default констр.

};



! Деструктор

→ глен-функция, която се извиква при изтриването на обект.

- ! Има стърго има нато структурата, но с'н пред него
- ! Има тип на връзка
- ! той е единствен
- ! Ако не го разпишем, то компилаторът създава такъв

Пример:

Struct A{

}

интиригращ
на конструктора

Struct A{
A();
~A();
}

! Деструктор при композиция

Пример:

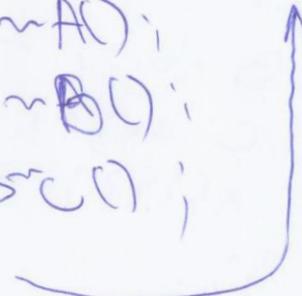
Struct X{

A obj1; → ~A();

B obj2; → ~B();

C obj3; → ~C();

{ ~X() }





Клас



За „no-copy“ обекти



В класа „no default“ всичко е private;



Структура



За „no-mutable“ обекти



В структурата „no default“ всичко е public

Модифициращ член-датчик \rightarrow член датчици, които могат да се променят от константни објекти

Пример
Struct A{

private:

mutable int x;

public :

void f() const{

x++}; ✓

}

};

- mutable член датчици не биваат на външното състояние на обекта;
- използване само в краен случаи;

! ~ $x()$ }

{ → деструкторът извиква
обратно в обратен ред
на десериализацията



Лансиране → ограничаване на достъпа

! конструктори за достъп

!!! Private → данни достъпни само в рамките на тази структурата

!!! Protected → имат достъп от класа и наследниците

! Интерфейс за достъп

!!! get/set методи

Пример:

int getAge() const

return age; → връщане

void setAge(int) → осъществяване на
има валидните
на подадените
данни

ЗАДАЧА 4: Пример за клас Person с име (наз с дължина най-много 20 символа) и години [5...90].



ЛЕКЦИЯ 5:

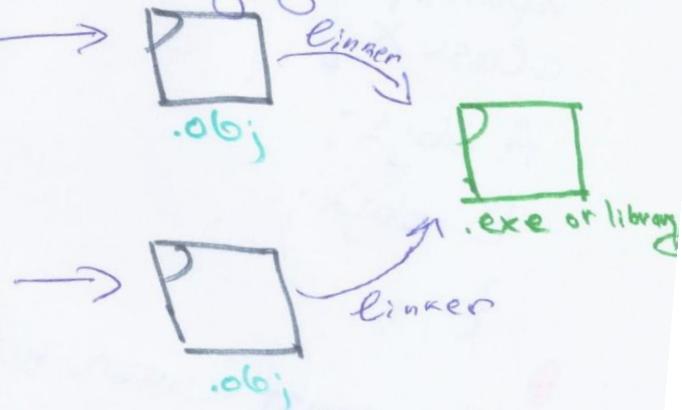
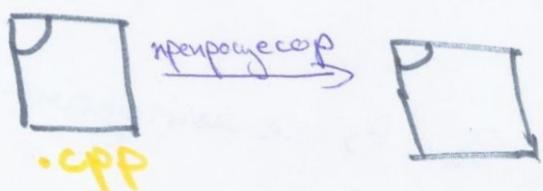
- Разделна компиляция.
- Препроцесор.
- Копиращ конструктор и оператор=.
- Композиция и агрегация.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Лекция V

Разгледа на компиляцията



→ обяснение, която се комбинира
при издаването със
има груп



! Препроцесор → текстоиздадовка на самите обяснение

Пример:

#include "file.txt" → може ред се замени със съдържанието на "file.txt";

#define Pi 3.14 → от може ред наставя, когато има Pi, че се замени с 3.14;

! Трансформация → промяна всички интервали, нови
редове, табулатури

! Синтаксичен анализ → дали кодът е с правилен
синтаксис

! Семантичен анализ → дали даден кодът има смисъл

! Оптимизация

! Assembly code

! Linking

Композиция и агрегация

! Композиция

Пример:

```
class X {
```

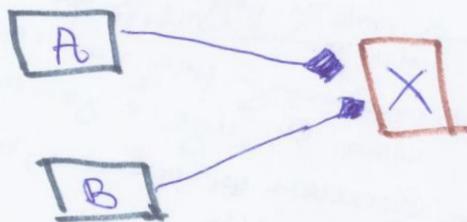
```
    A obj1;
```

```
    B obj2;
```

```
};
```

! Нижненият член на А и В се контролира от X.

Class Diagram (UML diagram)



! Агрегация

class X { } → X не отговаря за низнения член

 A* ptr;

};

X(A* temp): ptr (temp)

Y(A* obj): ret (obj)

class Y {

 A* ref; } → реферирането трябва да има зададена стойност

!

Копирате на обекти

- Копирай конструтор
- Оператор =

Копирай конструтор

→ конструтор, който приема обект от същия клас и тягущият обект става негово значение.

Пример:

A (const A& other);

- Ако не разширим R.R., то компилаторът генерира такъв

Пример:

A myObj;

A myObj2 (myObj); → приема две
создадена
инстанции

Оператор= → Применяется от source класс и между ними ставят несколько копий.

! При оператор= подразумевают все существующие объекты

! При R-R. создание новых объектов

Пример:

A obj;

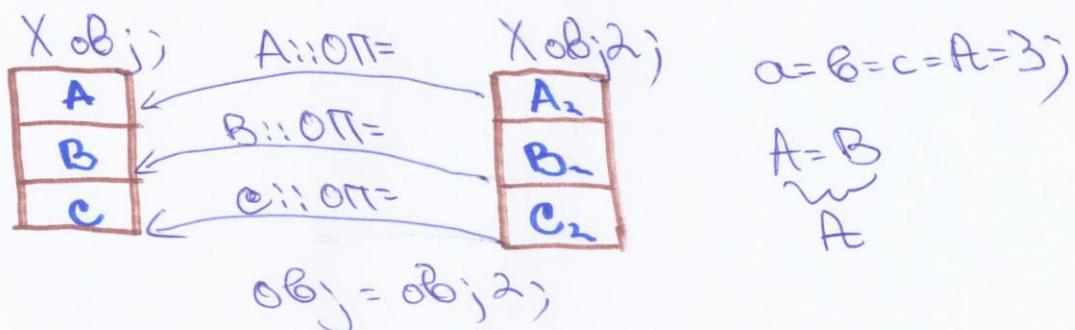
A obj2;

obj = obj2; → оператор=

! A obj;

A obj2; obj2 = obj; → инициализируется с помощью конструктора

! При ОП=

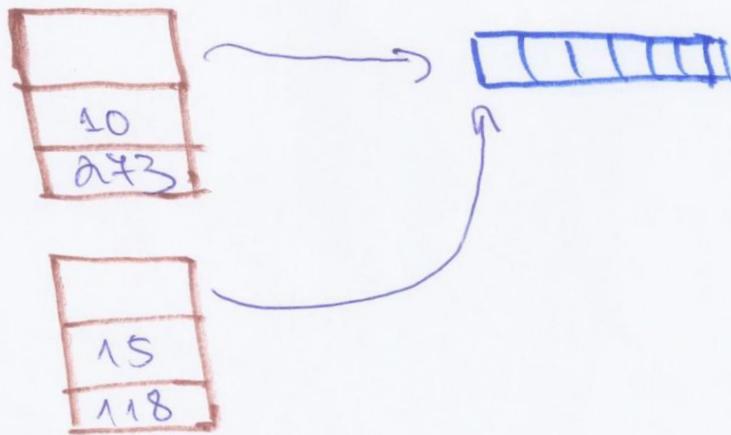


! Ако ОП= не е разписан експлицитно, то компилаторът създава кансъб по подразбиране



Shallow copy

→ при работе с вектором ресурс
и default-^{ко} генерируется R.R.W.



! Решение на shallow copy

!! Задоргавяне на копиратето

Пример:

A(const A&)= delete;

A& operator=(const A&)= delete;

!! Експлицитно различаване на копиратето
→ Deep copy

ЗАДАЧА 5: Пример с клас Event (използваме клас Time и Date на готово).

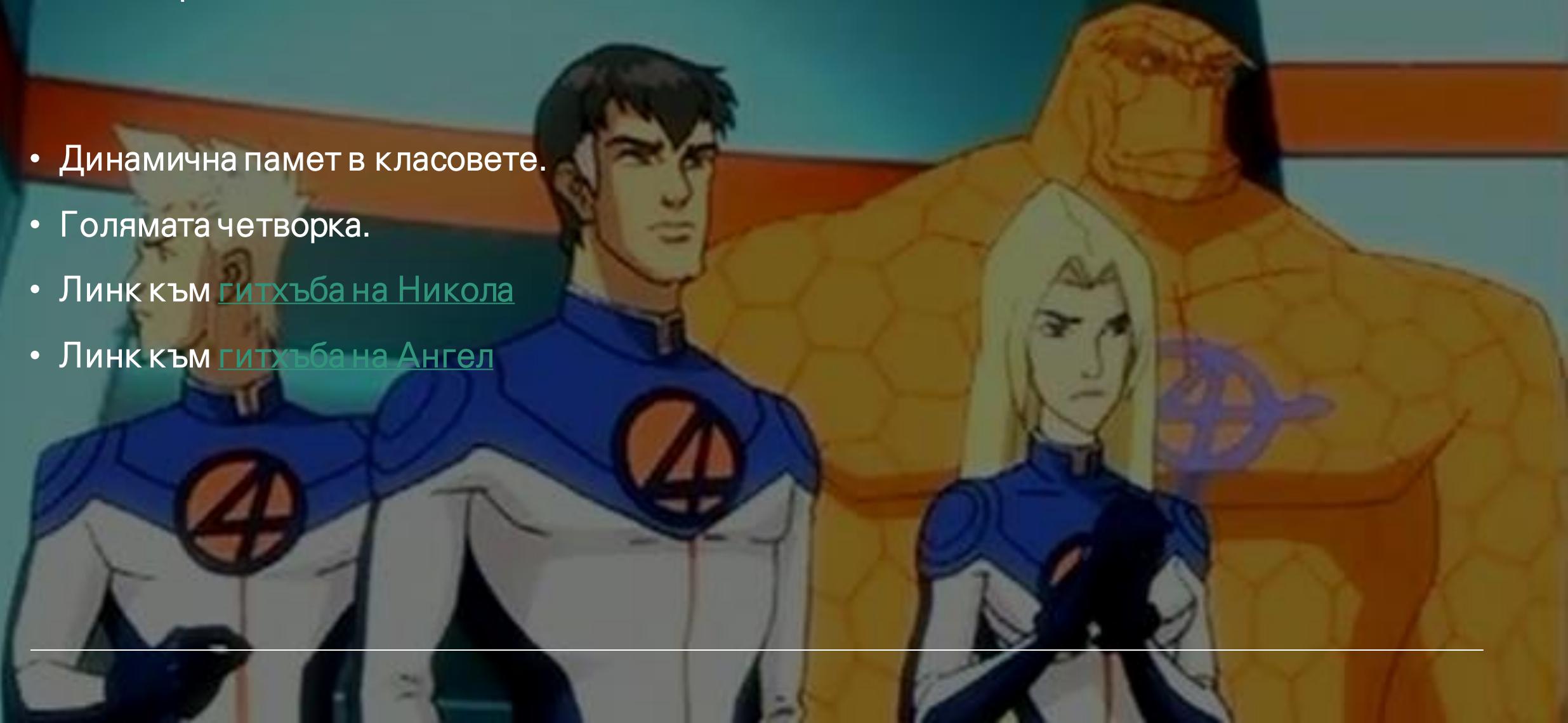
```
1 #pragma once
2 #include "../Time/Time.h"
3 #include "../Date/Date.h"
4 class Event
5 {
6     char _name[21];
7     Date _date;
8     Time _startTime;
9     Time _endTime;
10
11     void setName(const char* str);
12     void validateTimes();
13
14 public:
15     Event();
16     Event(const char* name, const Date& date, const Time& startTime, const Time& endTime);
17
18     Event(const char* name, unsigned day, unsigned month, unsigned year,
19           unsigned startTimeHours, unsigned startTimeMins, unsigned startTimeSecs,
20           unsigned endTimeHours,   unsigned endTimeMins,   unsigned endTimeSecs);
21
22     const char* getName() const;
23     const Date& getDate() const;
24     const Time& getStartTime() const;
25     const Time& getEndTime() const;
26
27
28
29 };
```

```
1 #include "Event.h"
2 #pragma warning(disable : 4996)
3
4
5 Event::Event(const char* name, const Date& date, const Time& startTime, const Time& endTime) : _date(date), _startTime(startTime), _endTime(endTime)
6 {
7     setName(name);
8     validateTimes();
9 }
10
11 Event::Event(const char* name, unsigned day, unsigned month, unsigned year,
12                 unsigned startTimeHours, unsigned startTimeMins, unsigned startTimeSecs,
13                 unsigned endTimeHours, unsigned endTimeMins, unsigned endTimeSecs) : _date(day, month, year),
14                                         _startTime(startTimeHours, startTimeMins, startTimeSecs),
15                                         _endTime(endTimeHours, endTimeMins, endTimeSecs)
16 {
17     setName(name);
18     validateTimes();
19 }
20 const char* Event::getName() const
21 {
22     return _name;
23 }
24 const Date& Event::getDate() const
25 {
26     return _date;
27 }
28 const Time& Event::getStartTime() const
29 {
30     return _startTime;
31 }
32 const Time& Event::getEndTime() const
33 {
34     return _endTime;
35 }
```

```
37     void Event::setName(const char* str)
38     {
39         if (strlen(str) > 20)
40             return;
41         else
42             strcpy(_name, str);
43     }
44     void Event::validateTimes()
45     {
46         if (_startTime.compare(_endTime) >= 1)
47             std::swap(_startTime, _endTime);
48     }
49
50     Event::Event() : Event("", 1, 1, 1, 0, 0, 0, 0, 0, 0) {}
```

ЛЕКЦИЯ 6:

- Динамична памет в класовете.
- Голямата четворка.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)



Лекция VII

! Голямата гетворка

! Директори конструктор \rightarrow създава се ~~създава~~ от компютъра, когато не е разписан друг конструктор.

! Копиращ конструктор \rightarrow създава се от компютъра, когато не е избрани или не са разписаны експлицитно темпори имена и т.н. на след. данните.

! OTT = \rightarrow създава се от компютъра, когато не е избрани или разписан експлицитно темпори + OR на име данните.

! Деструктор \rightarrow генерира се по подразбиране когато не сме разписан експлицитно;



Разписване голямата гетворка
САМО, когато имаме динамична
маса в индексирана структурата

Примери:

За извиквате:

$f(X \& obj)$

{ } → конст. констр. на X ;

{ } → дескр. на X ;

$g(X \& obj)$ { } → ползване едини obj ;

{

$t(X^* \& obj)$ { } → ползвай как едини obj ;

{

$X \& obj$; → геоф. констр.

$X \& obj_2 = obj_1$; → конст. констр. (започна инициализация);
исследование obj_2 ;

$X \& obj_3(obj_1)$; → конст. констр.;

$obj_2 = obj_1 \Rightarrow OPT = \left. \begin{array}{l} \text{присваиване стойност на всички} \\ \text{създадени и инициализирани обекти,} \\ \text{които са} \\ \text{известни са} \\ \text{конструирани} \end{array} \right\}$

$f(obj)$; → RR + Дескр.;

(за времето

$g(obj)$; → X } (обекти, които използват) ↓
 $t(& obj)$; → X } (броят на t) (на времето
един, константа obj);



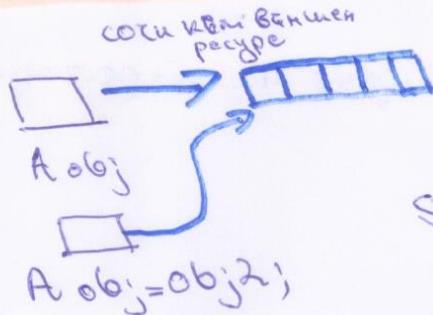
Проблем с default generated K.K, ОП = , Decrp.

Пример за shallow copy:

struct A {

char* str;

}



Копира се поинтъра; така поинтъра в obj и obj2 сочат към едно и също място; поради тази причина при копирането се изригат проблеми.

Пример:

class X {

A obj;

B obj2;

int main() {

1) X obj; // A(), B(), X()

2) X obj2 = obj; // CC A(), CC B(), CC X()

3) obj = obj2; // OP=X, OP=A, OP=B

ЗАДАЧА 6: Пример за клас студент с име (с произволна дължина) и масив от оценки(с произволна дължина)

M

N

O

Над среден

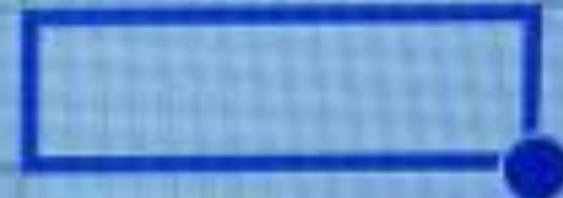
20

Допуснати

38

Без шанс

164



```
1 #pragma once
2 #include <iostream>
3
4 class Person {
5 private:
6     char* name = nullptr;
7     int age = 0;
8 public:
9     Person() = default;
10    Person(const char* name, int age);
11    Person(const Person& other);
12    Person& operator=(const Person& other);
13    ~Person();
14
15    const char* getName() const;
16    int getAge() const;
17
18    void setName(const char* name);
19    void setAge(int age);
20
21    void print() const;
22 private:
23     void CopyFrom(const Person& other);
24     void Free();
25 }
```

```
1 #pragma once
2 #include "Person.h"
3 #include <iostream>
4 #include <cstring>
5
6 Person::Person(const char* name, int age) {
7     setName(name);
8     setAge(age);
9 }
10
11 Person::Person(const Person& other) {
12     CopyFrom(other);
13 }
14
15 Person& Person::operator=(const Person& other) {
16     if (this != &other) {
17         Free();
18         CopyFrom(other);
19     }
20     return *this;
21 }
22
23 Person::~Person() {
24     Free();
25 }
26
27 const char* Person::getName() const {
28     return name;
29 }
30
31 int Person::getAge() const {
32     return age;
33 }
```

```
35     void Person::setName(const char* name) {
36         if (name == nullptr || this->name == name) {
37             return;
38         }
39
40         delete[] this->name;
41         size_t nameLen = strlen(name);
42         this->name = new char[nameLen + 1];
43         strcpy(this->name, name);
44     }
45
46     void Person::setAge(int age) {
47         this->age = age;
48     }
49
50     void Person::print() const {
51         std::cout << name << " " << age << std::endl;
52     }
53
54     void Person::CopyFrom(const Person& other) {
55         name = new char[strlen(other.name) + 1];
56         strcpy(name, other.name);
57         age = other.age;
58     }
59
60     void Person::Free() {
61         delete[] name;
62     }
```

ЛЕКЦИЯ 7:

Роми нападнаха репортер и оператор на Нова телевизия

| 06.10.2015 / 14:35



Сподели в:



- Предефиниране на оператори.
- Приятелски класове и функции.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Тема VII



Преодоляване на оператори

Пример:

```
class A {
```

```
    int a;
```

```
    int b;
```

```
}
```

```
A obj1;
```

```
A obj2;
```

```
obj1 + obj2; // иска също да е разрешено
```



Операторите са 3 вида:

- унарни (на 1 аргумент) ++
- бинарни (на 2 аргумента) -
- тернарен оператор (?: :)



Операторите се характеризират по:

- ассоциативност

• лево $\rightarrow +, -, /, *$

• дясно $\rightarrow =$

• приоритет $* > +$

- позицията на операндите спрямо оператора

• преднастъпки $\rightarrow ++a, --a, *a$

• индексни $\rightarrow [, - , *, /]$

• скобници $\rightarrow (,)$

Пример: Преобразуване на +

class A {

 int a1;

 int a2;

}

operator +

!! Въвежда оп-з

!! използва оп-з

1) A operator+(const A& lhs, const A& rhs)

{

 A res { lhs.a1 + rhs.a1, lhs.a2 + rhs.a2};

 return res;

}

2) class A {

 private:

 int a1;

 int a2;

 public:

 A operator+(const A& rhs)

{

 return A{a1 + rhs.a1, a2 + rhs.a2};

}

int main() {

 A obj1;

 A obj2;

 A res = obj1 + obj2; ✓

}

Пример: Переопределение на $\text{t} =$

$a \text{ t} = b$

вн
a (value)

!!! Всегда дружелюбно

!!! не всегда дружелюбно

1) A& operator $\text{t} =$ (A&lhs, const A&rhs) {
A res {lhs.a₁.t = rhs.a₁; lhs.a₂.t = rhs.a₂};
return res;
}

2) ~~A&~~ operator $\text{t} =$ (const A&rhs){

a₁.t = rhs.a₁;
a₂.t = rhs.a₂;
return *this;

}



Уговорка:

!!! Операторы $\text{t} =, -=, *=, /=, \% =$ → Гарантия что слен-дружелюбны

!!! Операторы $\text{t}, -, /, \%$ → Гарантия что они дружелюбны

Пример: Добавяне на ++

rvalue \rightarrow $a++ \rightarrow \text{operator}++()$

lvalue \rightarrow $a++ \rightarrow \text{operator}++(\text{int})$

dummy
parameter
(за да покаже,
че са различни
объекти)

X& operator++()

num++;

return (*this);

}

X operator++(int){}

num++;

return X;

}



Приметска ставка / функция \rightarrow приложена
функция е

външна от-я за клас, който
има достъп до private/protected
член-даници.

\rightarrow приложени клас е външна клас, който има
достъп до private/protected членовете на
издадената клас

Пример: Преопределение на операторите за поток

```
A obj;  
cout << obj;  
cin >> obj;
```

ostream operator << (ostream& os, const A& obj) com

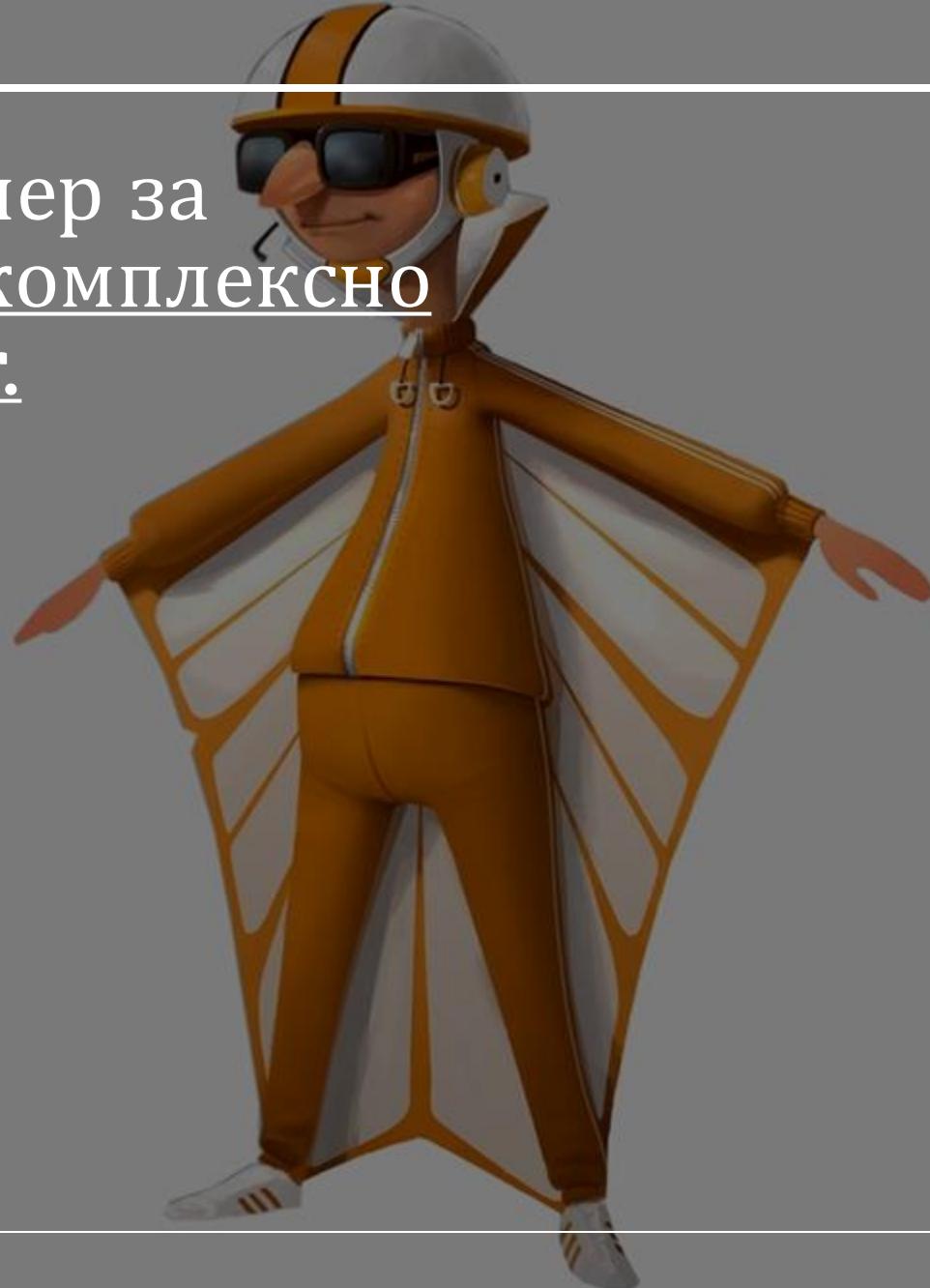
```
{  
    :  
    :  
    return os;  
}
```

! Този е левият аргумент \Rightarrow предъявява
га е левата сторона. Ако беше
правата, то левият параметър щеше
да е обект.

istream operator >> (istream& is, const A& obj)
{
 :
 :
 return is;

! Втората сторона, преняните специални

ЗАДАЧА 7: Пример за
реализация на комплексно
число и Nvector.



```
1
2 #include <iostream>
3 class Complex
4 {
5     private:
6         double real;
7         double im;
8
9     public:
10    Complex();
11    Complex(double real, double im);
12
13    Complex getConjugated() const; //връща комплексно спрегнатото число на текущия обект.
14
15    //assignment operators
16    // a+=b Към a се добавя стойността на b. a се променя , b не се. Операцията ще върне a за да може да се правят верижни изрази (a+=b+=c)
17    Complex& operator+=(const Complex&);
18    Complex& operator-=(const Complex&);
19    Complex& operator*=(const Complex&);
20    Complex& operator/=(const Complex&);
21
22
23
24    //stream opeartors ( std::cout << c1 ) (std::cin >> c1 )
25    friend std::ostream& operator<<(std::ostream&, const Complex&);
26    friend std::istream& operator>>(std::istream&, Complex&);
27
28};
29 // a+b. Операторът връща НОВА променлива(Complex), която е резултатът от операцията.
30 Complex operator+(const Complex&, const Complex&);
31 Complex operator-(const Complex&, const Complex&);
32 Complex operator*(const Complex&, const Complex&);
33 Complex operator/(const Complex&, const Complex&);
```

```
1 #include "Complex.h"
2
3 Complex::Complex()
4 {
5     real = 0;
6     im = 0;
7 }
8 Complex::Complex(double real, double im)
9 {
10    this->real = real;
11    this->im = im;
12 }
13 Complex& Complex::operator+=(const Complex& other)
14 {
15    real += other.real;
16    im += other.im;
17    return *this;
18 }
19 Complex& Complex::operator-=(const Complex& other)
20 {
21    real -= other.real;
22    im -= other.im;
23    return *this;
24 }
25 Complex& Complex::operator*=(const Complex& other)
26 {
27    double oldReal = real;
28    real = real*other.real - im*other.im;
29    im = oldReal*other.im + im * other.real;
30    return *this;
31 }
```

```
--  
32     Complex& Complex::operator/=(const Complex& other) // a/=b  
33     {  
34         Complex Conjugated = other.getConjugated(); //взимаме комплексно спрегнатата на другата дроб (b)  
35  
36         Complex otherCopy(other); // копираме другата (b), за да не я промянаме.  
37  
38         //Умножаваме двете по комплексно спрегнатата.  
39         *this *= Conjugated;  
40         otherCopy *= Conjugated; //Тук трябва да остане само реална част.  
41  
42         if (otherCopy.real != 0)  
43         {  
44             real /= otherCopy.real;  
45             im /= otherCopy.real;  
46         }  
47  
48         return *this;  
49     }  
50  
51  
52     //След дефинирането на +=, -=, *= и /=, можем лесно да дефинираме +, -, * и /, преизползвайки вече написаните оператори.  
53     Complex operator+(const Complex& lhs, const Complex& rhs) { //a+b  
54  
55         Complex result(lhs); // Създаваме нов обект, който е копие на a (използваме копи-конструктора)  
56         result += rhs; //към копието на a, добавяме b (Използваме вече дефинирания оператор +=)  
57  
58         return result;  
59     }  
60  
61     //Операторите -, *, / са аналогични с +.  
62     Complex operator-(const Complex& lhs, const Complex& rhs) {  
63         Complex result(lhs);  
64         result -= rhs;  
65  
66         return result;
```

```
69     Complex operator*(const Complex& lhs, const Complex& rhs) {
70
71         Complex result(lhs);
72         result *= rhs;
73
74         return result;
75     }
76
77     Complex operator/(const Complex& lhs, const Complex& rhs) {
78
79         Complex result(lhs);
80         result /= rhs;
81
82         return result;
83     }
84
85     Complex Complex::getConjugated() const
86     {
87         Complex result(*this);
88         result.im *= -1;
89
90         return result;
91     }
92
93     // за да можем да правим cout<<c1 или да запазваме във файл и др.
94     std::ostream& operator<<(std::ostream& ofs, const Complex& r) {
95
96         return ofs << r.real << ' ' << r.im << 'i';
97     }
98
99     // за да можем да правим cin>>c1 или да четем от файл и др.
100    std::istream& operator>>(std::istream& ifs, Complex& r) {
101
102        return ifs >> r.real >> r.im;
103    }
```

ЛЕКЦИЯ 8:

- Статични член-данни. Изключения. Обработка на изключения.
- Йерархия на изключенията и примери.
- Изключения в конструктори и деструктори.
- Нива на exception safety.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

If you choose an answer to this question at random,
what is the chance that you will be correct?

♦ A: 25%

♦ C: 50%

♦ B: 0%

♦ D: 25%

Тема VIII

Изключени

Пример:

- static unwinding
- перарти на изключени
- изключени в конструктора

Пример:

```
int main()
```

```
f()
```

```
?
```

```
f() {
```

```
[A obj(...)]
```

```
{
```

```
AC() {
```

```
:
```

```
:
```

```
throw 3();
```

```
{
```

→ При грешка се извличат всички

обекти над A, но

не и на самого A.



При изключени конструктора не трябва да съдържа незавършени външни ресурс.

Пример:

```
class X {
```

```
X(): obj1(--), obj2(--), obj3(--)
```

```
    A obj1;
```

```
    B obj2;
```

```
    C obj3;
```

```
}
```

грешка

Съществува построено, всички се на AC()



При грешка се извличат обекти. всички съществуващи построени обекти.

! Пример за тема

struct A {

}

A* ptr = new A[5];

конструктори:

AC(), AC(), AC(), AC()

AAAAAA

хвърля exception

броя

деструктори:

~A(), ~AC(), ~AC()

броя

Използване в деструкторите \rightarrow Три возможаване на

2 изключения, всичката се възниква std::terminate()
и програмата спира.

! Не хвърляне изключения в деструктори

static

! Пра зефінісум!

!! static local variable

Пример:

```
f() {  
    static A obj;  
}
```

$f(1) \{ AC \rightarrow 13$
 $f(1) \}^1$
 $f(1) \} \text{ non same}$
 $f(1) \} \text{ equal to}$
 $f(1) \} \text{ A obj same}$
 $f(1) \} \text{ representation}$

А об'єкт
созадає
само відчуття
при поганого
успіхів
на др-зі.

!! static function → не имеет своего
своего состояния
и может
использоваться
всеми объектами

 static member

```
class A{  
    static int x; → не е обявян с конкретен  
    int g; → различно  
} → за всеки  
онджен съект A;
```



It's not the people who vote that count. It's the people who count the votes.

ЗАДАЧА 8: Пример с клас, който брои инстанциите си.

— *Joseph Stalin* —

```
1 #pragma once
2
3
4 class SelfCounting
5 {
6
7     //a constant, which may be different
8     //for different objects
9     //so it is "per object"
10    const int const_val;
11
12    static unsigned liveObjectsCount;
13    static unsigned createdCount;
14
15 public:
16    SelfCounting();
17    SelfCounting(int val);
18    SelfCounting(const SelfCounting& other);
19    ~SelfCounting();
20
21    //methods which aren't
22    //related with particular
23    //instance
24    static unsigned getLiveObjectsCount();
25    static unsigned getCreatedCount();
26
27};
```

```
5     unsigned SelfCounting::liveObjectsCount = 0;
6     unsigned SelfCounting::createdCount = 0;
7
8     SelfCounting::SelfCounting() : SelfCounting(42) //initialization of constants
9                                     //can only be done while creating them
10                                    //so the only possible way is in
11                                    //the initialization list
12 {
13 }
14
15 SelfCounting::SelfCounting(int val) : const_val(val)
16 {
17     liveObjectsCount++;
18     createdCount++;
19 }
20 SelfCounting::SelfCounting(const SelfCounting& other) : const_val(other.const_val)
21 {
22     liveObjectsCount++;
23     createdCount++;
24 }
25
26 SelfCounting::~SelfCounting()
27 {
28     liveObjectsCount--;
29 }
30
31 unsigned SelfCounting::getLiveObjectsCount()
32 {
33     return liveObjectsCount;
34 }
35
36 unsigned SelfCounting::getCreatedCount()
37 {
38     return createdCount;
39 }
```

ЛЕКЦИЯ 9:

- Масиви от указатели към обекти.
- Move семантики - ползи, lvalue, rvalue, move конструктор/op=, std::move.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Тема IX

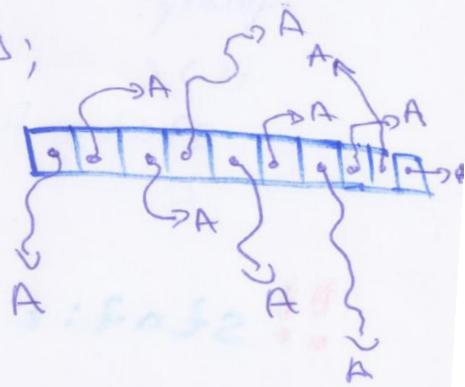
Копирование и обработка



2 вида:

!! $A^* \text{ptr} = \text{new } A[10];$

!! $A^{**} \text{ptr2} = \text{new } A^*[10];$



A^*



locality (сочетан. крит.)



A^{**}



нужен позиц.



но-безде swapable



хардк. но-много памят



не e righter default construction

Move семантика

- создание - конструктор
- разрушение - деструктор
- копирование - конструктор копирования
- Oⁿ =

→ наследование

Пример:

```
int main() {  
    String s1; // nullptr  
    f(s1);  
    // Удаление move семантики  
    // в f(s1);  
}  
{ ~s1  
    delete nullptr; // handled by delete  
}
```

f(string s) {
 string s;
}

delete nullptr → handled by delete

Множественные статичные в C++

● 3 основные типы

!! rvalue → имея на променлива/функция (специальная)

→ изъятие из функции, когда возвращается референция

Пример:

```
int f()  
{
```

f() = 9; // Важно, понятие врыва rvalue;

!! prvalue → литерал (т, nullptr, false)

→ изъятие из лп-я, когда врыва по

конечной статичности

!! xvalue (expiring value) → обекти, ком кратна

нижнейся си цепоч

prvalue + xvalue = rvalue

Пример за xvalue:

a. P → зоне гарни
↙
 временно
обекти

! xvalue не може да
е на областта
на полето

f1(int a){

 Konue

 - value ✓
 - rvalue ✓

{

f2(int& a){

 - value ✓
 - rvalue ✗

}

f3(const int& a){

 - value ✓
 - rvalue ✓

}

f4(int&& a){

 - value ✗
 - rvalue ✓

{ f4(4); f4(8) }

! &&-value reference
приема само value, не
могат конуе, а директно
изменя го.

void g4(A&& obj){

 int matrix[2]

 A obj;

 g4(obj); ✗

 g4(A{---}); ✓ → не опади конуе на A

 ↳ конструктор на A

}

`std::move(obj)` → преобразува от `lvalue` в `rvalue`

Пример:

A obj;

`f4(std::move(obj));`

`std::move()`

→ obj става в `xvalue`, кое то е `long raw`

→ възможният, без да прави конве

• не използва деструктор

{ mAC }



move OP =

Пример: `String: OP=(String& other)?`

```
delete[] data;  
data=other.data;  
other.data=nullptr;
```

}



• move конструиращ конструктор



• moveFrom(String& other) обект

RAII → Всеки валиден ресурс, който използва класа ня трябва да е обврзан с инициализирането му чрез констру

• за добавяне на валидност

!! add(const T&obj);

!! add(T&& obj);

→ value reference,
не оставяме подадените
обекти да се използват
след извикването на
функцията

Пример:

int main()

AC---> ако няма име е value,
често удаляемо на същия ред)

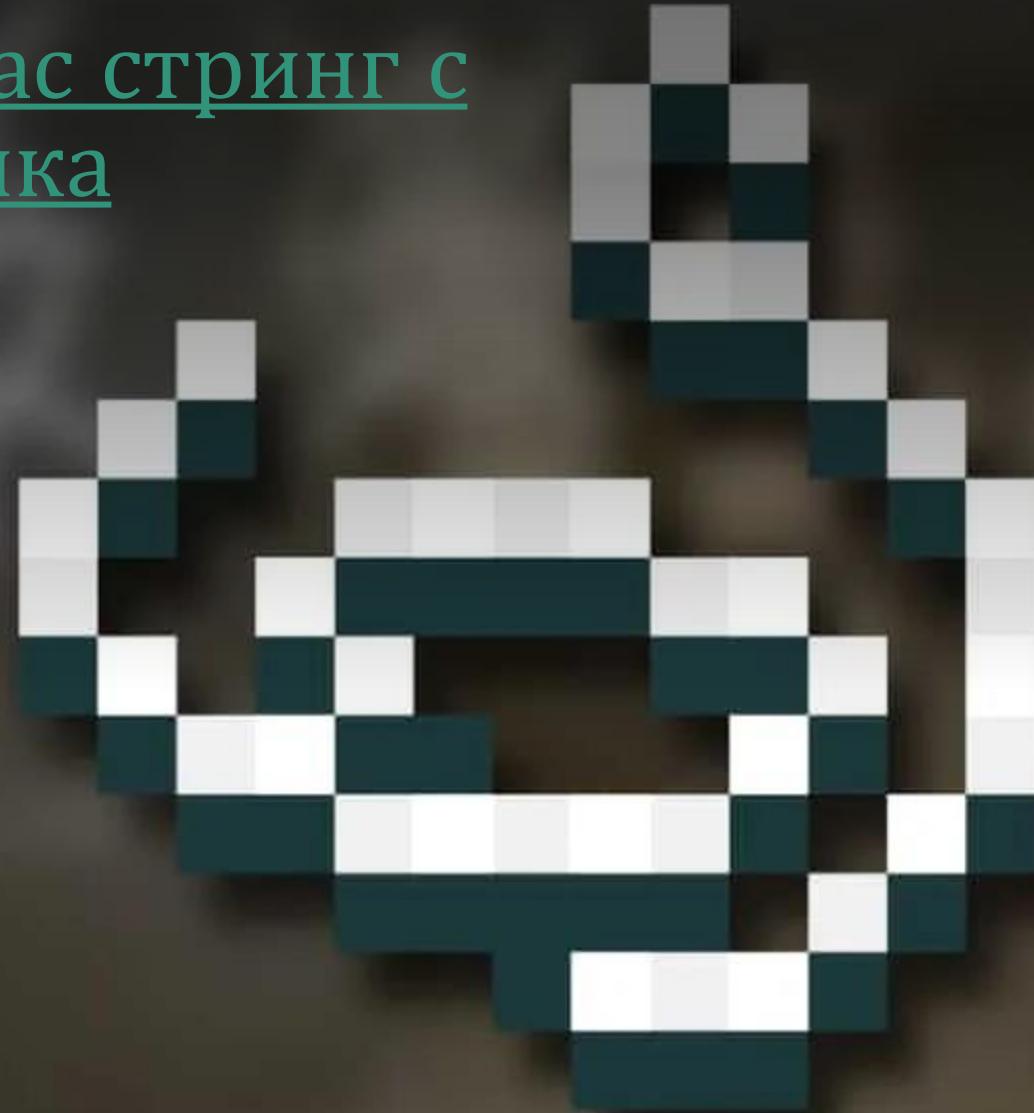
}

int main()

A&S = AC--->;

{ → ако даваме редоверенчка, тъй като на обекта че
се уверява до края на scope-а.

ЗАДАЧА 9: Клас стринг с move семантика



```
1 #pragma once
2 #include <iostream>
3
4 class MyString
5 {
6     char* _data;
7     size_t _length;
8
9     void copyFrom(const MyString& data);
10    void move(MyString&& other);
11    void free();
12
13    explicit MyString(size_t capacity); //for memory allocation. How much bytes to allocate
14 public:
15
16    MyString();
17    MyString(const char* data);
18    MyString(const MyString& other);
19    MyString& operator=(const MyString& other);
20    ~MyString();
21
22    MyString(MyString&& other) noexcept;
23    MyString& operator=(MyString&& other) noexcept;
24
25    size_t length() const;
26    MyString& operator+=(const MyString& other);
27
28    MyString substr(size_t begin, size_t howMany) const;
29
30    char& operator[](size_t index);
31    char operator[](size_t index) const;
32
33    const char* c_str() const;
34
35    friend MyString operator+(const MyString& lhs, const MyString& rhs);
36    friend std::istream& operator>>(std::istream&, MyString& str);
--
```

```
38
39     std::ostream& operator<<(std::ostream& os, const MyString& str);
40
41     bool operator<(const MyString& lhs, const MyString& rhs);
42     bool operator<=(const MyString& lhs, const MyString& rhs);
43     bool operator>=(const MyString& lhs, const MyString& rhs);
44     bool operator>(const MyString& lhs, const MyString& rhs);
45     bool operator==(const MyString& lhs, const MyString& rhs);
46     bool operator!=(const MyString& lhs, const MyString& rhs);
```

```
1 #include "MyString.h"
2
3 MyString::MyString(size_t capacity)
4 {
5     _length = capacity - 1;
6     _data = new char[capacity];
7 }
8
9 MyString operator+(const MyString& lhs, const MyString& rhs)
10 {
11     MyString result(lhs._length + rhs._length + 1);
12
13     result[0] = '\0';
14     strcat(result._data, lhs._data);
15     strcat(result._data, rhs._data);
16
17     return result;
18 }
19
20 MyString& MyString::operator+=(const MyString& other)
21 {
22     char* result = new char[(_length += other._length) + 1];
23     result[0] = '\0';
24     strcat(result, _data);
25     strcat(result, other._data);
26
27     delete[] _data;
28     _data = result;
29
30     return *this;
31 }
32
```

```
38 MyString::MyString(const char* data) : MyString(strlen(data) + 1)
39 {
40     strcpy(_data, data);
41 }
42
43 MyString::MyString(const MyString& other)
44 {
45     copyFrom(other);
46 }
47
48 MyString& MyString::operator=(const MyString& other)
49 {
50     if (this != &other)
51     {
52         free();
53         copyFrom(other);
54     }
55     return *this;
56 }
57
58
59 void MyString::free()
60 {
61     delete[] _data;
62     _data = nullptr;
63 }
64
65 MyString::~MyString()
66 {
67     free();
68 }
69
```

```
70     size_t MyString::length() const
71     {
72         return _length;
73     }
74
75     void MyString::copyFrom(const MyString& other)
76     {
77         _length = other._length;
78         _data = new char[_length + 1];
79         strcpy(_data, other._data);
80     }
81
82     void MyString::move(MyString&& other) {
83         _data = other._data;
84         other._data = nullptr;
85         _length = other._length;
86     }
87
88     MyString::MyString(MyString&& other) noexcept
89     {
90         move(std::move(other));
91     }
92
93     MyString& MyString::operator=(MyString&& other) noexcept
94     {
95         if (this != &other)
96         {
97             free();
98             move(std::move(other));
99         }
100        return *this;
101    }
102
```

```
103     char& MyString::operator[](size_t index) //Неконстантен достъп
104     {
105         return _data[index];
106     }
107
108     char MyString::operator[](size_t index) const //Константен достъп
109     {
110         return _data[index];
111     }
112
113     MyString MyString::substr(size_t begin, size_t howMany) const
114     {
115         if (begin + howMany > _length)
116             throw std::length_error("Error, Substr out of range");
117
118
119         MyString res(howMany + 1);
120         for (int i = 0; i < howMany; i++)
121             res._data[i] = _data[begin + i];
122         res[howMany] = '\0';
123         return res;
124     }
125
126     const char* MyString::c_str() const
127     {
128         return _data;
129     }
130
131     std::ostream& operator<<(std::ostream& os, const MyString& str)
132     {
133         return os << str.c_str();
134     }
135
```

```
136     std::istream& operator>>(std::istream& is, MyString& str)
137     {
138         char buff[1024];
139         is >> buff; // is.getLine(buff, 1024);
140
141         delete[] str._data;
142         str._length = strlen(buff);
143         str._data = new char[str._length + 1];
144         strcpy(str._data, buff);
145         return is;
146     }
147
148
149     bool operator<(const MyString& lhs, const MyString& rhs)
150     {
151         return strcmp(lhs.c_str(), rhs.c_str()) < 0;
152     }
153
154     bool operator<=(const MyString& lhs, const MyString& rhs)
155     {
156         return strcmp(lhs.c_str(), rhs.c_str()) <= 0;
157     }
158     bool operator>=(const MyString& lhs, const MyString& rhs)
159     {
160         return strcmp(lhs.c_str(), rhs.c_str()) >= 0;
161     }
162     bool operator>(const MyString& lhs, const MyString& rhs)
163     {
164         return strcmp(lhs.c_str(), rhs.c_str()) > 0;
165     }
166     bool operator==(const MyString& lhs, const MyString& rhs)
167     {
168         return strcmp(lhs.c_str(), rhs.c_str()) == 0;
169     }
170     bool operator!=(const MyString& lhs, const MyString& rhs)
171     {
172         return strcmp(lhs.c_str(), rhs.c_str()) != 0;
173     }
```

ЛЕКЦИЯ 10:

- Наследяване.
- Видове наследяване.
- Параметри на функции (указатели и референции).
- Конструктори и деструктори при наследяване.
- Копиране при наследяване.
- Move семантики при наследяване.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Наследование

Наследование → гетониприятие наследует члены своего
предка и дополняет его. Все наследуемые
члены получают к нему доступ, а это означает для наследников.

Пример:

```
class A {  
public:  
    int x;  
    int y;  
    void f() const;  
}
```

```
class X: A {  
    double c;
```

```
int main() {  
    X obj;
```

obj.x++; ✓
obj.y++; ✓
obj.f(); ✓

X



⚠ Консюмист → has a relationship

⚠ Наследование → is a relationship

Одни примеры:

f1(A obj);

f2(A& obj);

f3(const A& obj);

f4(A&& obj);

f1 obj(1, 2, 3);

f1(A obj){ A(A)

f1(obj);

{ ~A() }

A obj(b...);

f2(obj); ✓

f2(A...); ✗

f2(s28::move(obj)); ✗

f2(3); ✗

! При наследството:

- наследникът се наследява и от претходните
- наследника се даватът от този от предишните

struct A {

```
    int x;  
    private:  
    int y;  
}
```

struct X: A {

```
    f() { y++; } X
```

↓ Нема гетър

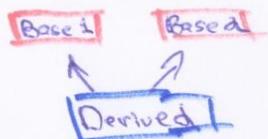
}

• protected: private, но е гетър за наследниците

! Derived is a Base



! Може да има нискоетерен наследяване



3. Інкапсуляція

private

protected

public

Приклад:

struct A {

public:

int x;

protected:

int y;

private:

int z;

{

struct X1: public A {

→ x **public**

→ y **protected**

→ z **private (член групи)**

struct X2: protected A {

→ x **e protected**

→ y **e protected**

→ z **член групи**

struct X3: private A {

→ x **private**

→ y **private**

→ z **член групи**

! Разлика между class и struct → при наследствете наследяването по подразбиране е private, докато при структурите е public.

! Ако имате Base или външен ресурс, когато наследниците му не се интересуват като, този работи с него, Base трябва сам да се покрие за външния си ресурс.

Примери за побеждане на базов наследствен клас при подаване като аргумент на обект

f1(Base b);
f2(const Base&);
f3(const Base*&p);
f4(Der &);
f5(const Der &);
f6(const Der*&);

! Класовете наследници могат да дават подаванни
като параметри на обект, който приемат обект от базовия клас.



f1(&); ✓ } не поддържа за съответствието
f2(&); ✓ } за Der, тък като сам Base застъпва.
f3(&); ✓ }
f4(&); ✓ } работят с членово Der
f5(&); ✓ }
f6(&); ✓ }

Base b;

f1(b); ✓

f2(b); ✓

f3(b); ✓

f4(b); ✗

f5(b); ✗

f6(b); ✗

Derived:

Base

int x;
·f();



Der

int x;
·g();

f1(const Base* ptr);

Der d;

f1(&d),

ptr → f(); ✓

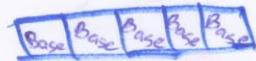
ptr → g(); ✗

ptr → &x++; ✓ increments x in Base class

• MacBu
Пример:

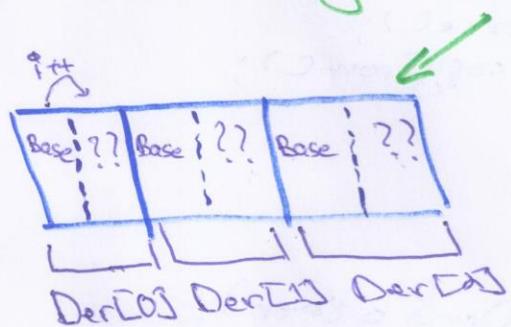
```
f(Base* ptr, size_t size) {  
    ptr[0].x++;  
    ptr[1].x--;  
    cout << ptr[2].x;  
}
```

!! Есть ограничение
по компонентам
sizeof(Base).



Der* ptr=new Der[3];

f(ptr, 3); \rightarrow valid syntax, but



\Rightarrow X Нарушено наследование на массиве от наследуемого

Конструктор при наследовании \rightarrow конструктор от Derived
избрана конструктор от Base. Ако не е указано,
то се избира конструктор по подразбиране.

```
Base  
↑  
Derived  
Der : Base {  
    A obj1;  
    B obj2;  
}
```

!! Конструктор на Derived;
Der(...); Base(...), obj1(-), obj2(-)
→
Base(-), A(-), B(-)

Деструктор при наследование → деструкторы на класс наследник
вызываются деструкторами на базовом классе, без го удалять
не нужно.

~Der{

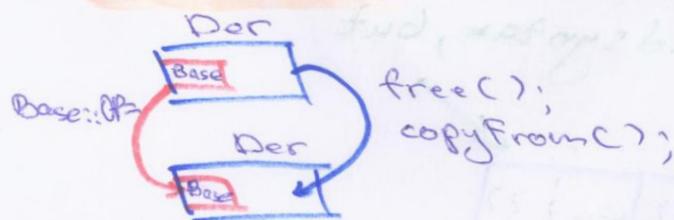
②

~Base()

~AC()

~Base()

One operator =



Пример:

```
OP=(const Der& other){  
    if (this != &other){  
        Base::operator=(other); // Оператор на other  
        free();  
        copyFrom(other);  
    }  
    return *this  
}
```

Move при наследование

Пример:

OP = Der(Der& other): Base (std::move(other))

```
{  
    if (this != &other){  
        std::move(other);  
        Base::operator=(other);  
        free();  
        std::move(other);  
        copyFrom(other);  
    }  
    return *this;  
}
```

Без move
запись Base

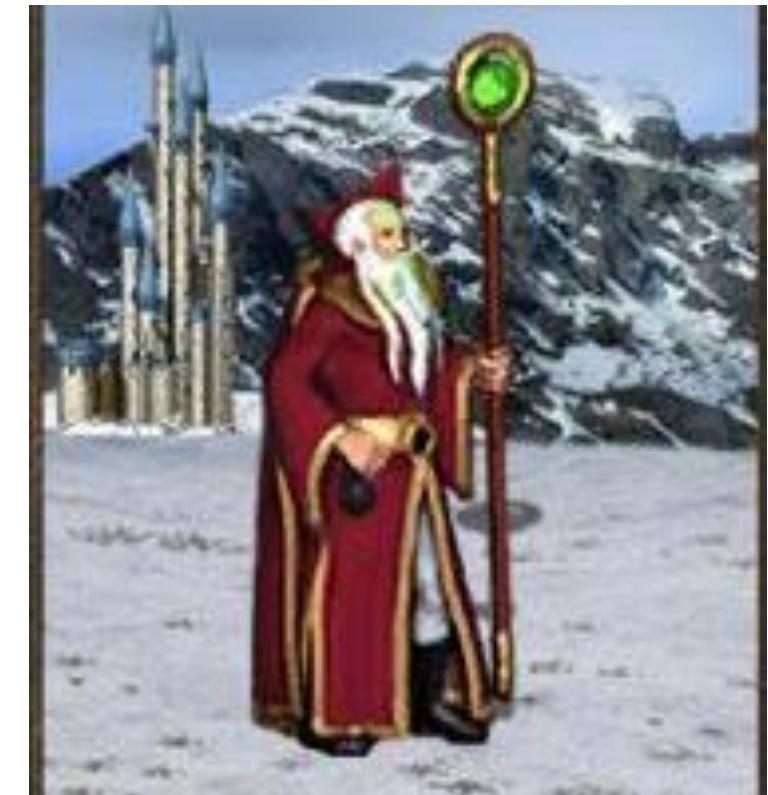
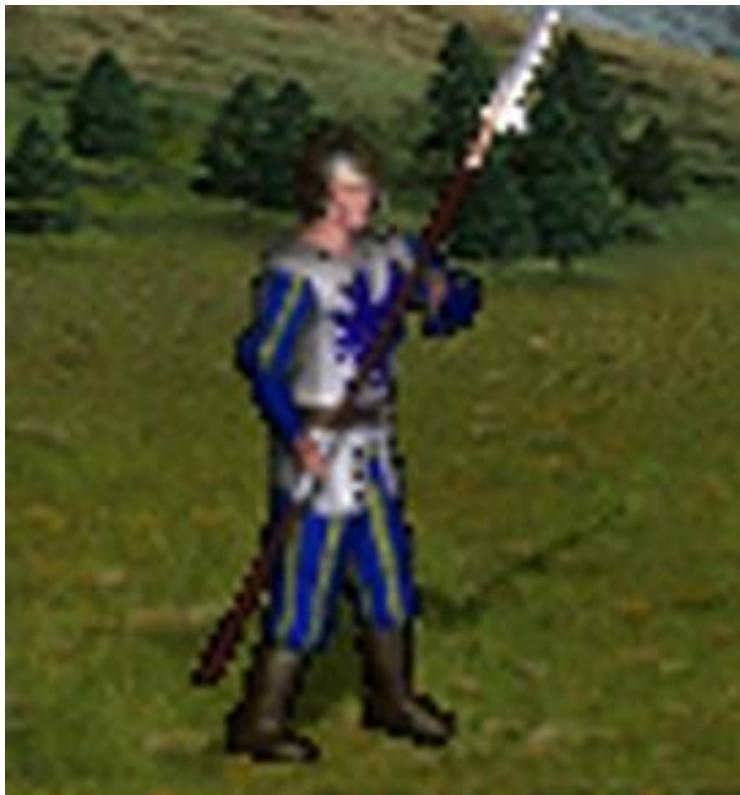
Der(Der& other): Base (std::move(other))

```
{  
    moveFrom(std::move(other));  
}
```

Без move,
запись с Base

Без move,
запись с Derived

ЗАДАЧА 10: Пример с човек, студент и преподавател



```
1 #pragma once
2 #include <iostream>
3 class Person
4 {
5     char* name = nullptr;
6     int age = 0;
7
8     void copyFrom(const Person& other);
9     void free();
10    void moveFrom(Person&& other);
11
12 public:
13     Person() = default;
14     Person(const char* name, int age);
15
16     Person(const Person& other);
17     Person& operator=(const Person& other);
18
19     Person(Person&& other);
20     Person& operator=(Person&& other);
21
22     const char* getName() const;
23     int getAge() const;
24
25     ~Person();
26     void print() const;
27
28 protected:
29     void setName(const char* name);
30     void setAge(int age);
31 };
```

```
1 #pragma once
2 #include "Person.h"
3 #include <cstring>
4 #pragma warning (disable:4996)
5
6
7
8 void Person::copyFrom(const Person& other)
9 {
10     name = new char[strlen(other.name) + 1];
11     strcpy(name, other.name);
12     age = other.age;
13 }
14
15 void Person::free()
16 {
17     delete[] name;
18 }
19
20 Person::Person(const char* name, int age)
21 {
22     setName(name);
23     setAge(age);
24 }
25
26 Person::Person(const Person& other)
27 {
28     copyFrom(other);
29 }
30
31 Person& Person::operator=(const Person& other)
32 {
33     if (this != &other)
34     {
35         free();
36         copyFrom(other);
37     }
38 }
```

```
38     return *this;
39 }
40
41 void Person::moveFrom(Person&& other)
42 {
43     name = other.name;
44     other.name = nullptr;
45     age = other.age;
46 }
47
48
49 Person::Person(Person&& other)
50 {
51     moveFrom(std::move(other));
52 }
53
54 Person& Person::operator=(Person&& other)
55 {
56     if (this != &other)
57     {
58         free();
59         moveFrom(std::move(other));
60     }
61     return *this;
62 }
63 const char* Person::getName() const
64 {
65     return name;
66 }
67
68 int Person::getAge() const
69 {
70     return age;
71 }
72
```

```
//  
73     void Person::setName(const char* name)  
74     {  
75         if (name == nullptr || this->name == name)  
76             return;  
77  
78         delete[] this->name;  
79         size_t nameLen = strlen(name);  
80         this->name = new char[nameLen + 1];  
81         strcpy(this->name, name);  
82     }  
83  
84     void Person::setAge(int age)  
85     {  
86         this->age = age;  
87     }  
88  
89  
90     Person::~Person()  
91     {  
92         free();  
93     }  
94  
95     void Person::print() const  
96     {  
97         std::cout << name << " " << age << std::endl;  
98     }  
99
```

```
1 #include <iostream>
2 #include "../Person/Person.h"
3
4 class Student : public Person
5 {
6     size_t fn = 0;
7 public:
8     Student() = default;
9     Student(const char* name, int age, size_t fn);
10
11    size_t getFn() const;
12    void setFn(size_t fn);
13
14
15    // No need for big 4, since there is NO dyn. memory in Student
16    // The compiler will generate valid:
17
18    //Student(const Student& other);
19    //Student& operator=(const Student& other); //no need
20    // ~Student()
21
22    //See on the .cpp file how the compiler creates them
23
24};
```

```
1      #include "Student.h"
2
3  Student::Student(const char* name, int age, size_t fn) : Person(name, age)
4  {
5      setFn(fn);
6  }
7
8  size_t Student::getFn() const
9  {
10     return fn;
11 }
12
13 void Student::setFn(size_t fn)
14 {
15
16     if (fn < 100)
17         fn = 100;
18     this->fn = fn;
19 }
20
21 // The compiler will make:
22
23 //Student(const Student& other) : Person(other), fn(other.fn), grade(other.grade)  OK!!!
24 //{}
25
26 //Student& operator=(const Student& other)          OK!!!!
27 //{
28 //    if (this != &other)
29 //    {
30 //        Person::operator=(other); //handles the deletion and copying of the base class
31 //        fn = other.fn;
32 //    }
33 //    return *this;
34 //}
```

```
1 #include "../Person/Person.h"
2
3 class Teacher : public Person
4 {
5     char** subjects;
6     size_t subjectsCount;
7
8     void free();
9     void copyFrom(const Teacher& other); // само нещата, които са от Teacher
10    void moveFrom(Teacher&& other);
11
12 public:
13     Teacher(const char* name, int age, const char* const * subjects, size_t subjectsCount);
14
15     Teacher(const Teacher& other);
16     Teacher& operator=(const Teacher& other);
17
18     Teacher(Teacher&& other);
19     Teacher& operator=(Teacher&& other);
20     ~Teacher();
21 }
```

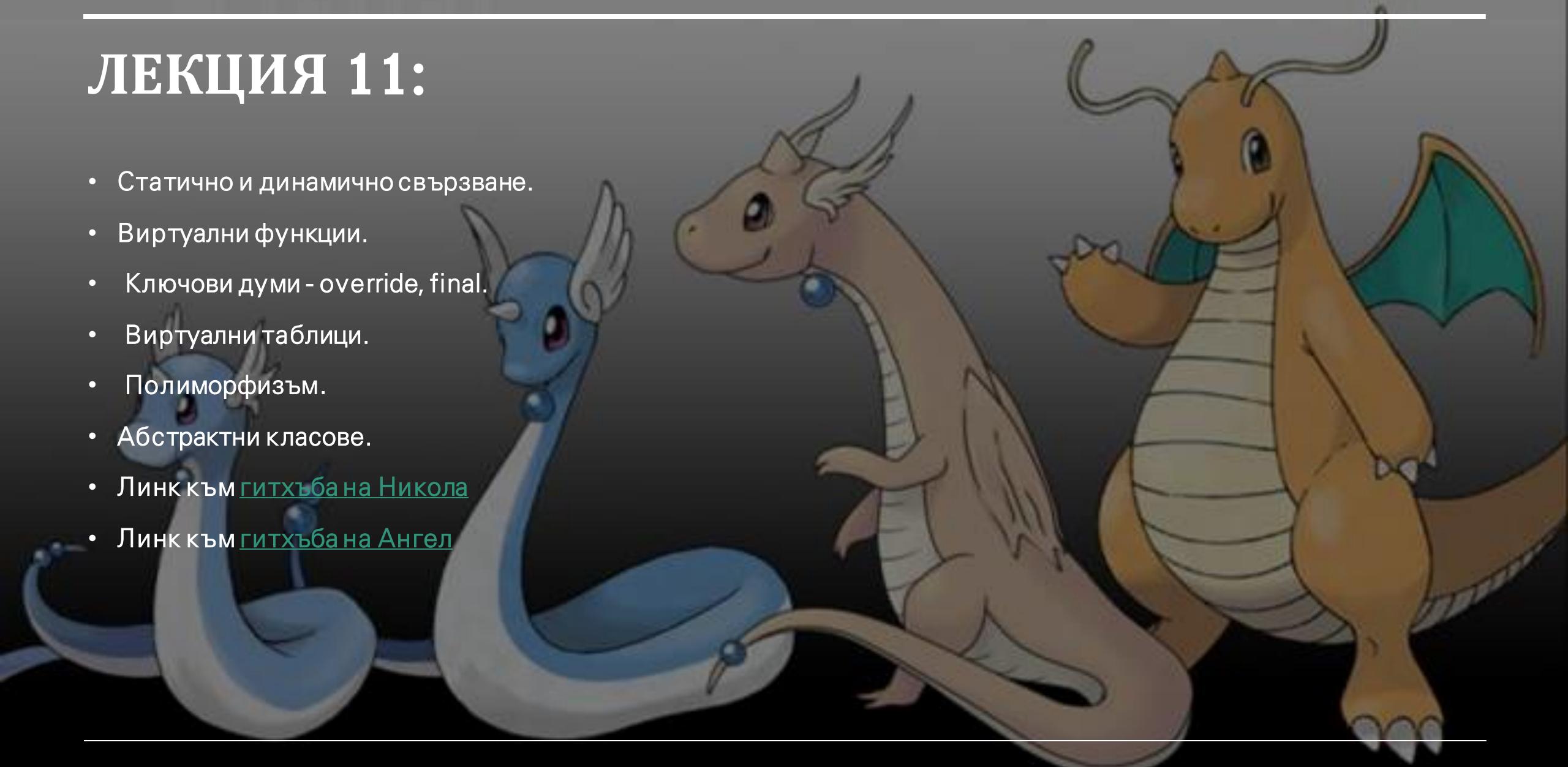
```
1 #include "Teacher.h"
2
3 Teacher::Teacher(const char* name, int age, const char* const* subjects, size_t subjectsCount) : Person(name, age)
4 {
5     this->subjects = new char* [subjectsCount];
6
7     for (size_t i = 0; i < subjectsCount; i++)
8     {
9         this->subjects[i] = new char[strlen(subjects[i]) + 1];
10        strcpy(this->subjects[i], subjects[i]);
11    }
12    this->subjectsCount = subjectsCount;
13 }
14
15 void Teacher::free()
16 {
17     for (size_t i = 0; i < subjectsCount; i++)
18         delete[] subjects[i];
19     delete[] subjects;
20 }
21 void Teacher::copyFrom(const Teacher& other)
22 {
23     subjects = new char* [other.subjectsCount];
24
25     for (size_t i = 0; i < other.subjectsCount; i++)
26     {
27         subjects[i] = new char[strlen(other.subjects[i]) + 1];
28         strcpy(subjects[i], other.subjects[i]);
29     }
30
31     subjectsCount = other.subjectsCount;
32 }
33
```

```
34 Teacher::Teacher(const Teacher& other) : Person(other)
35 {
36     copyFrom(other); //only the teacher stuff is copied
37 }
38
39 Teacher& Teacher::operator=(const Teacher& other)
40 {
41     if (this != &other)
42     {
43         Person::operator=(other); //delete data of person + copy the data from person
44         free(); //Teacher::free();
45         copyFrom(other); //Teacher::copyFrom()
46     }
47     return *this;
48 }
49 void Teacher::moveFrom(Teacher&& other)
50 {
51     subjects = other.subjects;
52     subjectsCount = other.subjectsCount;
53     other.subjects = nullptr;
54     other.subjectsCount = 0;
55 }
56
57
58 Teacher::Teacher(Teacher&& other) : Person(std::move(other))
59 {
60     moveFrom(std::move(other));
61 }
62
```

```
63     Teacher& Teacher::operator=(Teacher&& other)
64     {
65         if (this != &other)
66         {
67             Person::operator=(std::move(other));
68             free();
69             moveFrom(std::move(other));
70         }
71         return *this;
72     }
73
74     Teacher::~Teacher()
75     {
76         free();
77     } // destr person
```

ЛЕКЦИЯ 11:

- Статично и динамично свързване.
- Виртуални функции.
- Ключови думи - `override`, `final`.
- Виртуални таблици.
- Полиморфизъм.
- Абстрактни класове.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)



Тема XI

Полиморфизм → это же наследование по методу
различных наследников/имплементаций

Compile-time polymorphism

function overloading (перегрузка на стадии компиляции)

Пример:

```
void f();
void f(int);
void f(string);
int max(int, int);
int max(int, int, int);
```

operator overloading

Пример:

3+4

"ABC" + "XY"

3,5+4

Пример:

```
Base
class Base {
    int x;
}
Derived
class Derived : public Base {
    int x;
}
```



Der d;

d.x; // Derived::x

```
Base* ptr = &d;
ptr->x; // Base::x
```

Пример с virtual

```
class Base {  
    virtual void f() {}...}
```

{

```
class Der: Base {
```

```
    void f() override
```

```
    {}
```

```
}
```

Der d;

d.f(); // Der::f()

Base* ptr=d;

ptr->f();

Результат:

```
class Base {
```

```
    virtual void cout << "Base::f()"; }
```

```
void g() { f(); }
```

```
Base(); f(); }
```

```
~Base(); f(); }
```

```
class Der: Base {
```

```
    void f() override { cout << "Der::f()"; }
```

```
}
```

```
int main() {
```

Der d;

d.g(); // Der::f()

Base* ptr=d;

ptr->g(); // Der::f()

Derived

Base -> f()

ако f() е Superfunction
може надстроен но-всичко
допълнително тук f().

Пример:

```
class Base {  
    void f() {}  
}
```

```
class Der::Base {
```

```
    void f() {} override
```

```
}
```

```
Der d;
```

```
d.f(); Der::f()
```

```
Base* ptr = &d;
```

```
ptr->f(); // Base::f()
```



! **Динамично свързване** → изборът на функцията, която да се изпълни става по време на компилация, изборът етапа от членовете на редоверенция, откъдето се изпълнява функцията.

! **Динамично свързване** → изборът на функцията, която да се изпълни става по времето на изпълнение на програмата (run time polymorphism).

Изпълнява се чрез **Виртуални функции**.

Абстрактен клас → клас, програмиран за наследяване.

Неможем да иницират обекти от абстрактни клас. Също така може една абстрактна функция

! **Абстрактна функция** → pure virtual function
(също виртуална функция)

Синтаксис: virtual void f() = 0;

! Всички наследници трябва да имат реална реализация на тази функция, за да са полезни. Ако някой наследник не имплементира – става абстрактен, заместо че наследи поине една абстрактна функция.

! При полиморфен преговор на синтаксиса трябва да имаме виртуален деструктор!

Base* ptr = new Der(...);

delete ptr; // virtual ~Base();

тук $\downarrow \rightarrow$ ~Der()
надесно и написа

Икономска зона final → никой не

може да прегазища дадената

функция, оттук че тя е финална. За класове и структури final значи, че те немогат да се наследяват.

Base | virtual f()

Derived | f() override;

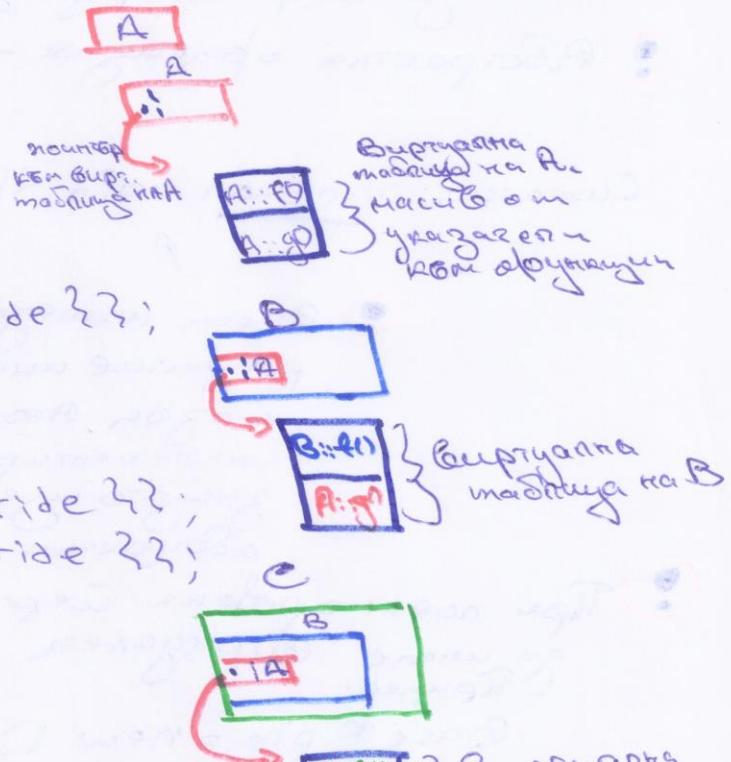
Derived2

Derived3 | f() final;

Derived4 | ~~f() override~~;

Виртуални методи → пример на синтаксиса на динамично свързване

```
struct A {  
    virtual f() {}  
    virtual g() {}  
};  
  
struct B : A {  
    void f() override {};  
};  
  
struct C : B {  
    void f() override {};  
    void g() override {};  
};
```



⚠ 2 генерализации
и 1 cast

- 1 генерализация → напиране на гадината
- 2 генерализации → напиране на обектната
- 1 cast → замяна на времето на обекта

Хетерогенен контейнер → клас, който съхранява колекция от предмети от различни абстракции клас.

⚠ Клониране → ~~заруменка~~, която всяки обекти правят вече да са си.

Пример:

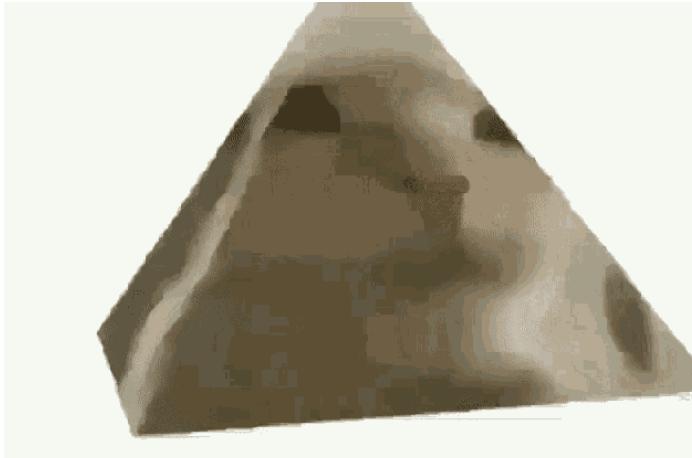
```
copyFrom (const Container & other)
```

```
for each:  
arr[i] = other.arr[i] → clone();
```

Будете clone опасни

ЗАДАЧА 11:

Пример за фигури



```
1 #pragma once
2 #include <iostream>
3
4 class Shape // Abstract class - no instances of Shape are allowed!
5 {
6 protected:
7     struct point
8     {
9         point() :x(0), y(0){}
10        point(int x, int y) :x(x), y(y){}
11        int x;
12        int y;
13        double getDist(const point& other) const
14        {
15            int dx = x - other.x;
16            int dy = y - other.y;
17
18            return sqrt(dx*dx + dy*dy);
19        }
20    };
21    const point& getPointAtIndex(size_t index) const;
22 private:
23     point* points;
24     size_t pointsCount;
25
26     void copyFrom(const Shape& other);
27     void free();
28
29
30 public:
31     Shape(size_t pointsCount);
32
33     Shape(const Shape& other);
34     Shape& operator=(const Shape& other);
35     virtual ~Shape(); //!!!!!!
36
37     void setPoint(size_t pointIndex, int x, int y);
```

```
~~~  
37     void setPoint(size_t pointIndex, int x, int y);  
38  
39     virtual double getArea() const = 0; //pure virtual  
40     virtual double getPer() const = 0; // pure virtual  
41     virtual bool isPointIn(int x, int y) const = 0;  
42  
43 };
```

```
1 #include "Shape.h"
2
3 Shape::Shape(size_t pointsCount) : pointsCount(pointsCount)
4 {
5     points = new point[pointsCount]; // [0,0], [0,0]...
6 }
7
8 void Shape::copyFrom(const Shape& other)
9 {
10    points = new point[other.pointsCount];
11
12    for (int i = 0; i < other.pointsCount; i++)
13        points[i] = other.points[i];
14
15    pointsCount = other.pointsCount;
16 }
17 void Shape::free()
18 {
19     delete[] points;
20 }
21
22 Shape::Shape(const Shape& other)
23 {
24     copyFrom(other);
25 }
26 Shape& Shape::operator= (const Shape& other)
27 {
28     if (this != &other)
29     {
30         free();
31         copyFrom(other);
32     }
33     return *this;
34 }
35 Shape::~Shape()
36 {
37     free();
```

```
38     }
39
40     const Shape::point& Shape::getPointAtIndex(size_t index) const
41     {
42         if (index >= pointsCount)
43             throw std::exception("Invalid point index!");
44
45         return points[index];
46     }
47
48     void Shape::setPoint(size_t pointIndex, int x, int y)
49     {
50         if (pointIndex >= pointsCount)
51             throw std::exception("Invalid point index!");
52
53         points[pointIndex] = point(x, y);
54     }
```

```
1 #pragma once
2 #include "Shape.h"
3
4 class Circle : public Shape
5 {
6     double radius;
7
8 public:
9     Circle(int x, int y, double radius);
10
11    double getArea() const override;
12    double getPer() const override;
13    bool isPointIn(int x, int y) const override;
14
15};
```

```
1 #include "Circle.h"
2
3 const double PI = 3.1415;
4 Circle::Circle(int x, int y, double radius) : Shape(1), radius(radius)
5 {
6     setPoint(0, x, y);
7 }
8
9 double Circle::getArea() const
10 {
11     return PI * radius * radius;
12 }
13 double Circle::getPer() const
14 {
15     return 2 * PI * radius;
16 }
17 bool Circle::isPointIn(int x, int y) const
18 {
19     Shape::point p(x, y);
20     return p.getDist(getPointAtIndex(0)) <= radius;
21 }
```

```
1 #pragma once
2 #include "Shape.h"
3
4 class Rectangle : public Shape
5 {
6 public:
7     Rectangle(int x1, int y1, int x3, int y3);
8     double getArea() const override;
9     double getPer() const override;
10    bool isPointIn(int x, int y) const override;
11
12};
```

```
1 #include "Rectangle.h"
2
3 Rectangle::Rectangle(int x1, int y1, int x3, int y3) : Shape(4)
4 {
5     setPoint(0, x1, y1);
6     setPoint(1, x1, y3);
7     setPoint(2, x3, y3);
8     setPoint(3, x3, y1);
9 }
10 double Rectangle::getArea() const
11 {
12     const Shape::point& p0 = getPointAtIndex(0);
13     const Shape::point& p1 = getPointAtIndex(1);
14     const Shape::point& p3 = getPointAtIndex(3);
15
16     return p0.getDist(p1) * p0.getDist(p3);
17 }
18
19 double Rectangle::getPer() const
20 {
21     const Shape::point& p0 = getPointAtIndex(0);
22     const Shape::point& p1 = getPointAtIndex(1);
23     const Shape::point& p3 = getPointAtIndex(3);
24
25     return 2*(p0.getDist(p1) + p0.getDist(p3));
26 }
27
28 bool Rectangle::isPointIn(int x, int y) const
29 {
30     Shape::point p(x, y);
31     return p.x >= getPointAtIndex(0).x && p.y >= getPointAtIndex(1).x &&
32             p.y <= getPointAtIndex(0).y && p.y >= getPointAtIndex(2).y;
33 }
```

```
1 #pragma once
2
3 #include "Shape.h"
4
5 class Triangle : public Shape
6 {
7
8 public:
9     Triangle(int x1, int y1, int x2, int y2, int x3, int y3);
10    double getArea() const override;
11    double getPer() const override;
12    bool isPointIn(int x, int y) const override;
13
14};
```

```
3
4     Triangle::Triangle(int x1, int y1, int x2, int y2, int x3, int y3) : Shape(3)
5     {
6         setPoint(0, x1, y1);
7         setPoint(1, x2, y2);
8         setPoint(2, x3, y3);
9     }
10    double Triangle::getArea() const
11    {
12        const Shape::point& p1 = getPointAtIndex(0);
13        const Shape::point& p2 = getPointAtIndex(1);
14        const Shape::point& p3 = getPointAtIndex(2);
15
16        return abs(p1.x*p2.y + p2.x*p3.y + p3.x*p1.y - p1.y * p2.x - p2.y*p3.x - p3.y*p1.x) / 2.00; //formula with the determinant
17    }
18    double Triangle::getPer() const
19    {
20        const Shape::point& p1 = getPointAtIndex(0);
21        const Shape::point& p2 = getPointAtIndex(1);
22        const Shape::point& p3 = getPointAtIndex(2);
23
24        return p1.getDist(p2) + p2.getDist(p3) + p3.getDist(p1);
25    }
26
27    bool Triangle::isPointIn(int x, int y) const
28    {
29        Shape::point p(x, y);
30        Triangle t1(getPointAtIndex(0).x, getPointAtIndex(0).y, getPointAtIndex(1).x, getPointAtIndex(1).y, p.x, p.y);
31        Triangle t2(getPointAtIndex(2).x, getPointAtIndex(2).y, getPointAtIndex(1).x, getPointAtIndex(1).y, p.x, p.y);
32        Triangle t3(getPointAtIndex(2).x, getPointAtIndex(2).y, getPointAtIndex(0).x, getPointAtIndex(0).y, p.x, p.y);
33
34        return abs(t1.getArea() + t2.getArea() + t3.getArea() - getArea()) <= std::numeric_limits<double>::epsilon();
35    }
36}
```

ЛЕКЦИЯ 12 (ТЕМА 13):

- Шаблони. Необходими функции в шаблонен клас/шаблонна функция.
- Темплейтни специализации.
- Примери за шаблонни класове/функции от стандартната библиотека.
- Умни указатели.
- Употреба и идея на `shared_ptr`, `weak_ptr`, `unique_ptr`.
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Тема XII

Параметричен полиморфизъм (шаблони)

• Пример за шаблони от лекцията:

функция max → сравняване на производълни обекти

template < типен аргумент T >

const T& max(const T& lhs, const T& rhs) {

входящина ← return (lhs > rhs) ? lhs : rhs;

случай, т.е.,

за да извикаме max<A>(A left), А трябва да има дефиниран оператор >.

max < int > (9, 3);

↳ копира кога на max и замества
навсякото T с int.

max < string > ("ABC", "XY");

↳ копира кога на max и замества
навсякото T с string.

• Пример за шаблонен клас

• Класове, в които на всички елементи не
има достъп

Опака (Container)

• ADS → абстрактна структура от данни

→ има също дефинирано поведение,
което е в имплементация

→ push() → добавяне

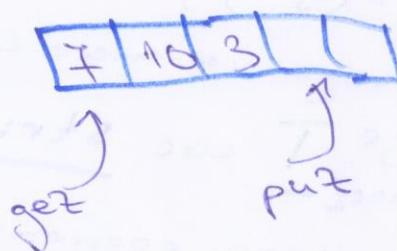
→ peek() → "предизвикателство"

→ pop() → премахване

`queue <int> q`
`q.push(7);`
`q.push(10);`
`q.push(3);`
`q.pop(); // 7`
`q.peek(); // 10`
`q.pop(); // 3`

• имплементация

- свързана (linked list) \rightarrow не е част от курса X
- през масива - чиркулярна структура ✓



• **Внимание!**
 .h .cpp \rightarrow веде за новите
 имплементации на .h

queue <int>

• 2 решения

queue <Apple> \rightarrow 1. .hpp обща
 Стандартизиране на .h

• \rightarrow 2. квад на .cpp
 обща назоваване
 с зарез между бегаче
 тествира.

**ЗАДАЧА 12: Пример за стек
(с шаблонен капацитет) и опашка(с функция `resize`)**



```
1
2 #include <iostream>
3
4 template <typename T, const unsigned S>
5 class MyStack
6 {
7 private:
8     T arr[S];
9     size_t size = 0;
10 public:
11     void push(const T& obj);
12     void push(T&& obj);
13
14     const T& peek() const;
15     void pop();
16
17     bool isEmpty() const;
18 };
19
20
21
22 template <typename T, const unsigned S>
23 void MyStack<T,S>::push(const T& obj)
24 {
25     if (size >= S)
26         return;
27     arr[size++] = obj;
28 }
29
30 template <typename T, const unsigned S>
31 void MyStack<T,S>::push(T&& obj)
32 {
33     if (size >= S)
34         return;
35     arr[size++] = std::move(obj);
36 }
```

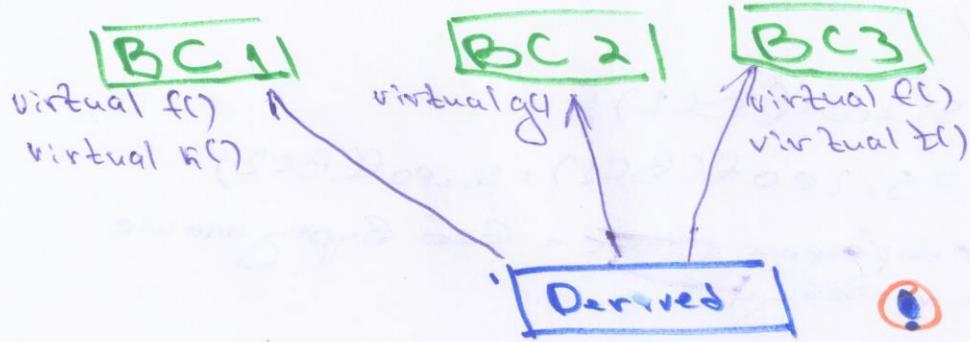
```
37
38     template <typename T, const unsigned S>
39     const T& MyStack<T,S>::peek() const
40     {
41         if (isEmpty())
42             throw std::out_of_range("Error!");
43
44         return arr[size - 1];
45     }
46
47     template <typename T, const unsigned S>
48     void MyStack<T,S>::pop()
49     {
50         if (isEmpty())
51             throw std::out_of_range("Error!");
52         size--;
53     }
54
55     template <typename T, const unsigned S>
56     bool MyStack<T,S>::isEmpty() const
57     {
58         return size == 0;
59     }
```

ЛЕКЦИЯ 13 (ТЕМА 12):

- Множествено наследяване.
- Диамантен проблем.
- Колекции от обекти в полиморфна йерархия.
- Копиране и триене
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

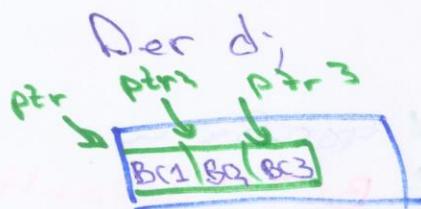
Лекция VIII

Множествено наследование



Override, → дадени са го опре-
делираме f(), го
које се ползу-
ва функцијата.

Пример:



BC1* ptr1 = &d;

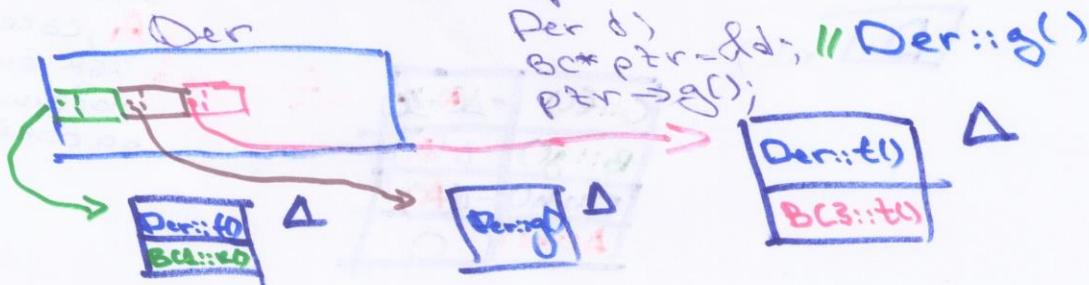
BC2* ptr2 = &d;

BC3* ptr3 = &d;

Спомените Δ → какво означаваат го напиши
јазарен, за го напиши правилни
одеки.

Пример:

Per d)
.BC* ptr=&d; // Der::g()
ptr->g();



$\Delta(\text{BC1}) \rightarrow$ можно отнести к BC1 или нет
на Der

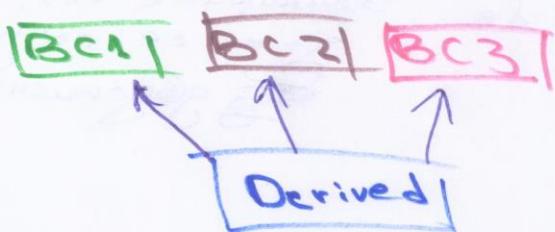
$$\Delta(\text{BC1}) = 0;$$

$$\Delta(\text{BC2}) = \text{sizeof}(\text{BC1});$$

$$\Delta(\text{BC3}) = \text{sizeof}(\text{BC1}) + \text{sizeof}(\text{BC2})$$

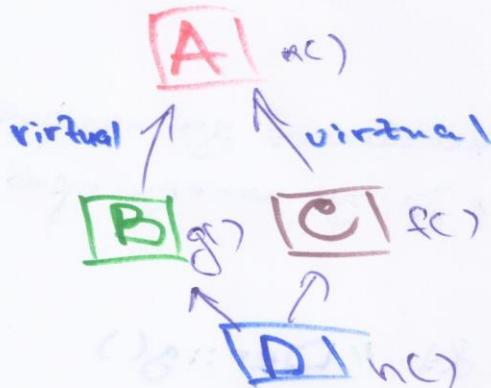
! $\Delta X \star \rightarrow$ наследник глобал в бб бирегулание
модулю

Пример:



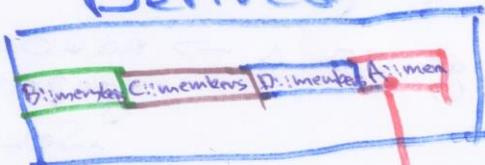
Der(): BC1(...), BC2(...), BC3() default

Бирегуланин модуль при гиант



! Важен пример за темата

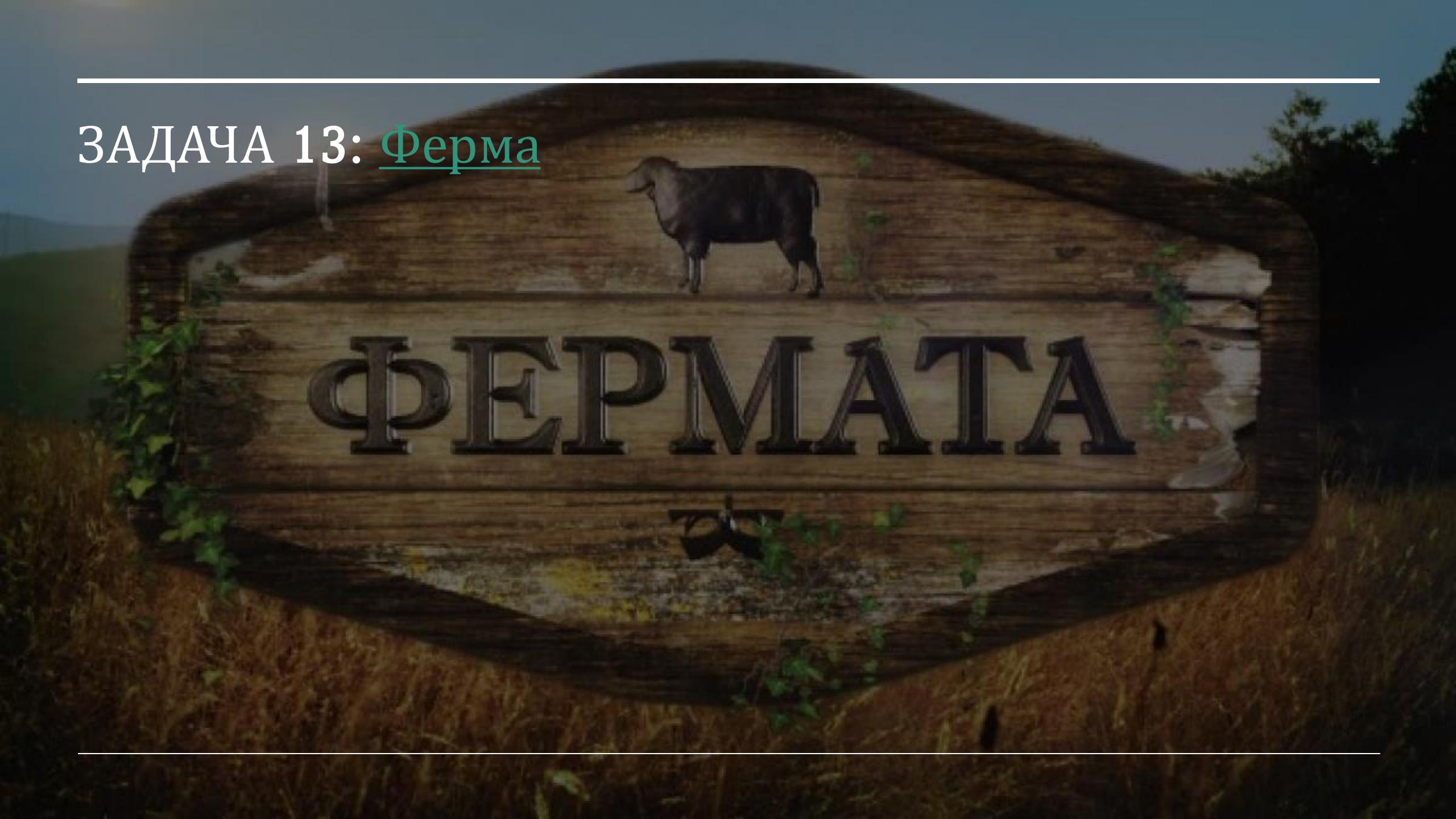
Derived



помощь B
известно
A, създава
всички
модули
на обекта

C::fc	-Δ(A+fc)
B::gc	-Δ(A)
D::nc	-Δ(A)
A::ac	0

ЗАДАЧА 13: Ферма



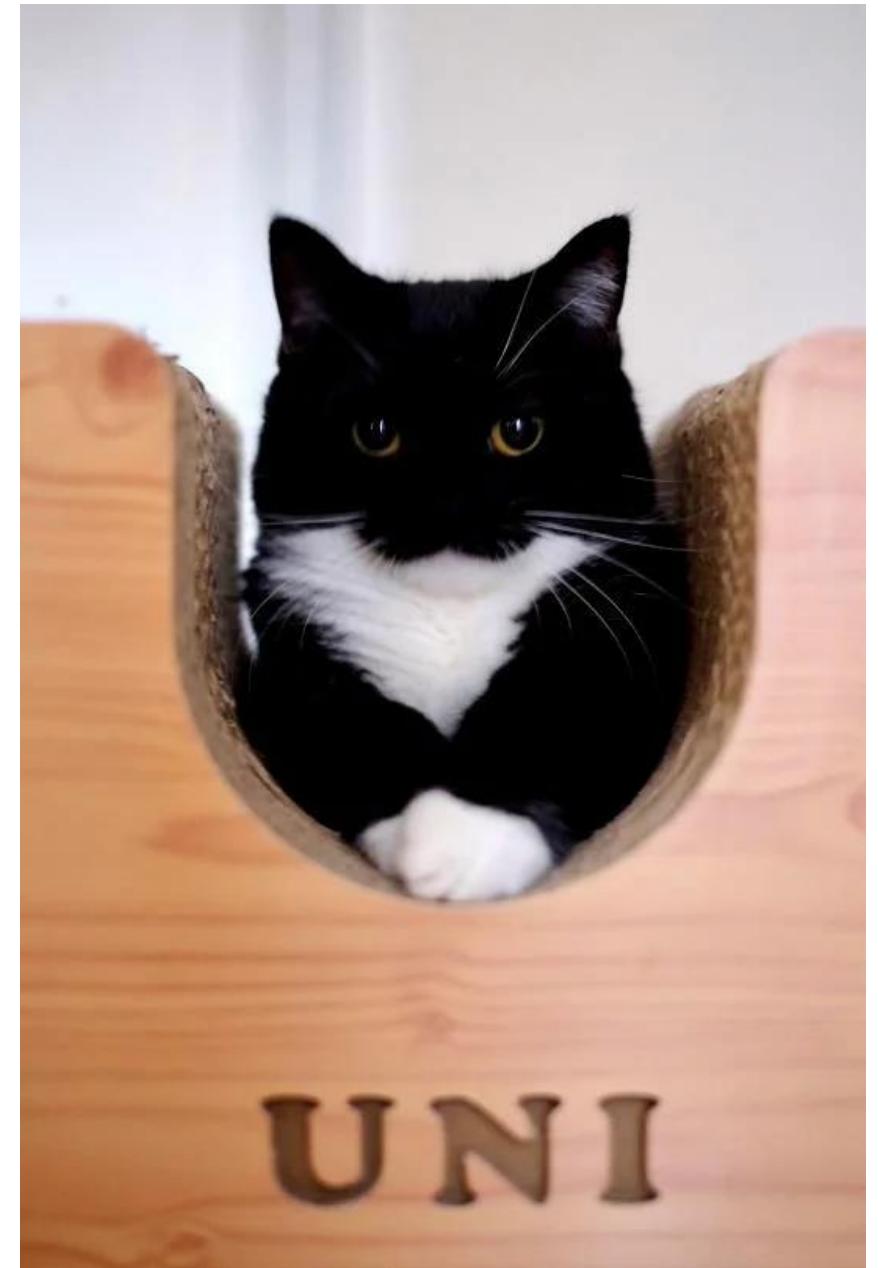
ФЕРМАТА

```
1 #pragma once
2
3
4 enum class AnimalType
5 {
6     Dog,
7     Cat,
8     Cow
9 };
10 class Animal
11 {
12     AnimalType type;
13 public:
14
15     Animal(AnimalType type) : type(type) {};
16     virtual void roar() const = 0;
17     virtual Animal* clone() const = 0; //създава копие на текущия обект.
18     virtual ~Animal() = default;
19
20     AnimalType getType() const
21     {
22         return type;
23     }
24 };
```



```
1 #pragma once
2 #include "Animal.h"
3 class Cat : public Animal
4 {
5 public:
6     Cat();
7     void roar() const override;
8     Animal* clone() const override;
9 }
```

```
1 #include "Cat.h"
2
3 Cat::Cat() : Animal(AnimalType::Cat)
4 {}
5
6 void Cat::roar() const
7 {
8     std::cout << "Meow meow!" << std::endl;
9 }
10
11 Animal* Cat::clone() const
12 {
13     Animal* newObj = new Cat(*this); //copy constr of Mouse
14     return newObj;
15 }
```



```
1 #pragma once
2
3 #include "Animal.h"
4
5 class Cow : public Animal
6 {
7     public:
8         Cow();
9         void roar() const override;
10        Animal* clone() const override;
11    };
12
13 #include "Cow.h"
14
15 Cow::Cow() : Animal(AnimalType::Cow)
16 {}
17
18 void Cow::roar() const
19 {
20     std::cout << "Muu muu!" << std::endl;
21 }
22
23 Animal* Cow::clone() const
24 {
25     Animal* newObj = new Cow(*this); //copy constr of Cow
26     return newObj;
27 }
```



```
1 #pragma once
2
3 #include "Animal.h"
4
5 class Dog : public Animal
6 {
7     public:
8         Dog();
9         void roar() const override;
10        Animal* clone() const override;
11    };
12
13 #include "Dog.h"
14
15 Dog::Dog() : Animal(AnimalType::Dog)
16 {}
17
18 void Dog::roar() const
19 {
20     std::cout << "Wof wof" << std::endl;
21 }
22
23 Animal* Dog::clone() const
24 {
25     Animal* newObj = new Dog(*this); //copy constr of Mouse
26     return newObj;
27 }
```



```
1 #pragma once
2 #include "../Animals/Animal.h"
3
4 class Farm
5 {
6     Animal** animals;
7     size_t animalsCount;
8     size_t capacity;
9
10    void free();
11    void copyFrom(const Farm& other);
12    void moveFrom(Farm&& other);
13    void resize();
14
15 public:
16    Farm();
17    Farm(const Farm& other);
18    Farm& operator=(const Farm& other);
19    ~Farm();
20
21    Farm(Farm&& other) noexcept;
22    Farm& operator=(Farm&& other) noexcept;
23
24    void addAnimal(AnimalType animalType);
25    void roarAll() const;
26
27    AnimalType getTypeByIndex(unsigned index) const;
28 }
```

```
1 #include "Farm.h"
2 #include "../Factory/AnimalFactory.h"
3 //The factory is in a seperate file, so we don't have to
4 //move include the animals in this file
5
6 void Farm::free()
7 {
8     for (size_t i = 0; i < animalsCount; i++)
9         delete animals[i]; //не се инт. какъв обект е. (вирт дестр)
10    delete[] animals;
11 }
12
13 void Farm::copyFrom(const Farm& other)
14 {
15     animals = new Animal*[other.capacity];
16     animalsCount = other.animalsCount;
17     capacity = other.capacity;
18
19     for (size_t i = 0; i < animalsCount; i++)
20         animals[i] = other.animals[i]->clone();
21 }
22
23 void Farm::moveFrom(Farm&& other)
24 {
25     animalsCount = other.animalsCount;
26     capacity = other.capacity;
27
28     animals = other.animals;
29     other.animals = nullptr;
30
31     other.animalsCount = other.capacity = 0;
32 }
33
```

```
63     Farm& Farm::operator=(const Farm& other)
64     {
65         if (this != &other)
66         {
67             free();
68             copyFrom(other);
69         }
70         return *this;
71     }
72
73     Farm::~Farm()
74     {
75         free();
76     }
77
78     Farm::Farm(Farm&& other) noexcept
79     {
80         moveFrom(std::move(other));
81     }
82     Farm& Farm::operator=(Farm&& other) noexcept
83     {
84         if (this != &other)
85         {
86             free();
87             moveFrom(std::move(other));
88         }
89         return *this;
90     }
91
```

```
34     void Farm::resize()
35     {
36         Animal** newCollection = new Animal * [capacity *= 2];
37         for (size_t i = 0; i < animalsCount; i++)
38             newCollection[i] = animals[i]; // !!не правим клониране
39         delete[] animals;
40         animals = newCollection;
41     }
42
43
44     void Farm::addAnimal(AnimalType animalType)
45     {
46         if (animalsCount == capacity)
47             resize();
48         animals[animalsCount++] = animalFactory(animalType);
49     }
50
51     Farm::Farm()
52     {
53         capacity = 8;
54         animalsCount = 0;
55         animals = new Animal * [capacity];
56     }
57
58     Farm::Farm(const Farm& other)
59     {
60         copyFrom(other);
61     }
62
```

```
92     void Farm::roarAll() const
93     {
94         for (size_t i = 0; i < animalsCount; i++)
95             animals[i]->roar();
96     }
97
98     AnimalType Farm::getTypeByIndex(unsigned i) const
99     {
100         if (i > animalsCount)
101             throw std::out_of_range("Invalid index");
102         return animals[i]->getType();
103     }
```

```
1 #include "../Animals/Animal.h"
2
3 Animal* animalFactory(AnimalType type);
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

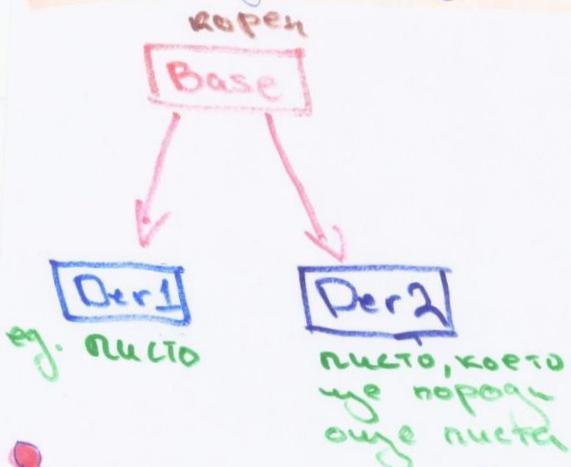
```
1 #include "AnimalFactory.h"
2 #include "../Animals/Cat.h"
3 #include "../Animals/Dog.h"
4 #include "../Animals/Cow.h"
5
6 Animal* animalFactory(AnimalType type)
7 {
8     switch (type)
9     {
10         case AnimalType::Dog:
11             return new Dog();
12         case AnimalType::Cat:
13             return new Cat();
14         case AnimalType::Cow:
15             return new Cow();
16     }
17     return nullptr;
18 }
```

ЛЕКЦИЯ 14 (ТЕМА 15):

- Дървовидна структура от обекти от полиморфна йерархия
- Линк към [гитхъба на Никола](#)
- Линк към [гитхъба на Ангел](#)

Лекция XIV

Двойная структура синтаксиса объектов



! Класс унаследует Der1 и Der2:



Derived2



Derived2



! Derived2 унаследует

от двух Base, поэтому может быть
один из двух Base или cả два

Пример:

Derived2::Base?

Base* ptr1;

Base* ptr2;

}

ЗАДАЧА 14: Логически изрази



```
1 #pragma once
2 #include "MyString/MyString.h"
3 #include "MyString/StringView.h"
4
5 class ExpressionCalculator {
6 private:
7     static const short CHARACTERS_COUNT = 26;
8
9     class BooleanInterpretation {
10 private:
11         // This contains the value of the character
12         bool variables[CHARACTERS_COUNT] = { false };
13 public:
14     bool getValue(char ch) const;
15     void setValue(char ch, bool value);
16
17     static bool isValidCharacter(char ch);
18     static BooleanInterpretation createFromNumber(size_t number, const bool variables[CHARACTERS_COUNT]);
19 };
20
21 class BooleanExpression {
22 public:
23     // Which characters are included in the expression
24     bool variables[CHARACTERS_COUNT] = { false };
25     short variablesCount = 0;
26
27     virtual bool evaluate(const BooleanInterpretation& interpretation) const = 0;
28     virtual BooleanExpression* clone() const = 0;
29     virtual ~BooleanExpression() = default;
30 };
31
32 class Variable : public BooleanExpression {
```

```
class Variable : public BooleanExpression {
private:
    char ch;
public:
    Variable(char ch);
    bool evaluate(const BooleanInterpretation& interpretation) const override;
    BooleanExpression* clone() const override;
};

class UnaryExpression : public BooleanExpression {
private:
    char operand;
    BooleanExpression* expression;
public:
    UnaryExpression(char operand, BooleanExpression* expression);
    bool evaluate(const BooleanInterpretation& interpretation) const;
    BooleanExpression* clone() const;
    ~UnaryExpression();
};

class BinaryExpression : public BooleanExpression {
private:
    char operand;
    BooleanExpression* left;
    BooleanExpression* right;
public:
    BinaryExpression(char operand, BooleanExpression* left, BooleanExpression* right);
    bool evaluate(const BooleanInterpretation& interpretation) const;
    BooleanExpression* clone() const;
    ~BinaryExpression();
};

BooleanExpression* expression = nullptr;
```

```
66     static bool checkAllVariations(const BooleanExpression* expression, bool expectedValue);
67     static BooleanExpression* parseExpression(const StringView& str);
68 public:
69     ExpressionCalculator(const MyString& str);
70     ExpressionCalculator(const ExpressionCalculator& other);
71     ExpressionCalculator(ExpressionCalculator&& other) noexcept;
72     ExpressionCalculator& operator=(const ExpressionCalculator& other);
73     ExpressionCalculator& operator=(ExpressionCalculator&& other) noexcept;
74     ~ExpressionCalculator();
75
76     bool isTautology() const;
77     bool isContradiction() const;
78
79 private:
80     void copyFrom(const ExpressionCalculator& other);
81     void move(ExpressionCalculator&& other);
82     void free();
83 }
```

```
1 #include "ExpressionCalculator.h"
2 #include <stdexcept>
3
4 namespace {
5     const char AND = '^';
6     const char OR = 'v';
7     const char IMPL = '>; //=>
8     const char IFF = '='; // <=>
9     const char XOR = '+';
10    const char NEG = '!';
11
12    bool isOperator(char ch) {
13        return ch == AND || ch == OR || ch == IFF || ch == IMPL || ch == XOR || ch == NEG;
14    }
15 }
16
17 /// BooleanInterpretation
18 bool ExpressionCalculator::BooleanInterpretation::isValidCharacter(char ch) {
19     return 'A' <= ch && ch <= 'Z';
20 }
21
22 ExpressionCalculator::BooleanInterpretation ExpressionCalculator::BooleanInterpretation::createFromNumber(size_t number, const bool variables[CHARACTERS_COUNT]) {
23     BooleanInterpretation result;
24
25     for (size_t i = 0; i < CHARACTERS_COUNT; i++) {
26         if (variables[i]) {
27             if (number & 1) { // (number % 2) != 0
28                 result.setValue('A' + i, true);
29             }
30
31             number >>= 1; // number /= 2
32         }
33     }
34
35     return result;
36 }
```

```
--  
38     bool ExpressionCalculator::BooleanInterpretation::getValue(char ch) const {  
39         if (!BooleanInterpretation::isValidCharacter(ch)) {  
40             throw std::exception("Invalid character");  
41         }  
42  
43         return variables[ch - 'A'];  
44     }  
45  
46     void ExpressionCalculator::BooleanInterpretation::setValue(char ch, bool value) {  
47         if (!BooleanInterpretation::isValidCharacter(ch)) {  
48             throw std::exception("Invalid character");  
49         }  
50  
51         variables[ch - 'A'] = value;  
52     }  
53  
54     ///  
55  
56     /// Variable  
57     ExpressionCalculator::Variable::Variable(char ch) : ch(ch) {  
58         variables[ch - 'A'] = true;  
59         variablesCount = 1;  
60     }  
61  
62     bool ExpressionCalculator::Variable::evaluate(const BooleanInterpretation& interpretation) const {  
63         return interpretation.getValue(ch);  
64     }  
65  
66     ExpressionCalculator::BooleanExpression* ExpressionCalculator::Variable::clone() const {  
67         return new Variable(*this);  
68     }  
69
```

```
74     ExpressionCalculator::UnaryExpression::UnaryExpression(char operand, BooleanExpression* expression)
75         : operand(operand), expression(expression)
76     {
77         for (size_t i = 0; i < CHARACTERS_COUNT; i++) {
78             variables[i] = expression->variables[i];
79         }
80
81         variablesCount = expression->variablesCount;
82     }
83
84     bool ExpressionCalculator::UnaryExpression::evaluate(const BooleanInterpretation& interpretation) const {
85         if (operand != NEG) {
86             return false;
87         }
88
89         return !expression->evaluate(interpretation);
90     }
91
92     ExpressionCalculator::BooleanExpression* ExpressionCalculator::UnaryExpression::clone() const {
93         return new UnaryExpression(operand, expression->clone());
94     }
95
96     ExpressionCalculator::UnaryExpression::~UnaryExpression() {
97         delete expression;
98     }
99     /**
100
```

```
102
103     ExpressionCalculator::BinaryExpression::BinaryExpression(char operand, BooleanExpression* left, BooleanExpression* right)
104         : operand(operand), left(left), right(right)
105     {
106         for (size_t i = 0; i < CHARACTERS_COUNT; i++) {
107             variables[i] = left->variables[i] || right->variables[i];
108             if (variables[i]) {
109                 variablesCount++;
110             }
111         }
112     }
113
114     bool ExpressionCalculator::BinaryExpression::evaluate(const BooleanInterpretation& interpretation) const {
115         switch (operand) {
116             case OR: return left->evaluate(interpretation) || right->evaluate(interpretation);
117             case AND: return left->evaluate(interpretation) && right->evaluate(interpretation);
118             case IMPL: return !left->evaluate(interpretation) || right->evaluate(interpretation);
119             case IFF: return left->evaluate(interpretation) == right->evaluate(interpretation);
120             case XOR: return left->evaluate(interpretation) != right->evaluate(interpretation);
121             default: return false; break;
122         }
123     }
124
125     ExpressionCalculator::BooleanExpression* ExpressionCalculator::BinaryExpression::clone() const {
126         return new BinaryExpression(operand, left->clone(), right->clone());
127     }
128
129     ExpressionCalculator::BinaryExpression::~BinaryExpression() {
130         delete left;
131         delete right;
132     }
133
134     /**
135
136 
```

```
138
139     bool ExpressionCalculator::checkAllVariations(const BooleanExpression* expression, bool expectedValue) {
140         size_t variationsCount = 1 << expression->variablesCount;
141
142         for (size_t i = 0; i < variationsCount; i++) {
143             if (expression->evaluate(BooleanInterpretation::createFromNumber(i, expression->variables)))
144                 != expectedValue) {
145                 return false;
146             }
147         }
148
149         return true;
150     }
151
152     ExpressionCalculator::BooleanExpression* ExpressionCalculator::parseExpression(const StringView& str) {
153         if (str.length() == 0) {
154             return nullptr;
155         }
156
157         if (str.length() == 1) {
158             return new Variable(str[0]);
159         }
160
161         StringView strWithoutBrackets = str.substr(1, str.length() - 2);
162         size_t length = strWithoutBrackets.length();
163         size_t bracketsCounter = 0;
164         for (size_t i = 0; i < length; i++) {
165             if (strWithoutBrackets[i] == '(') {
166                 bracketsCounter++;
167             }
168             else if (strWithoutBrackets[i] == ')') {
169                 bracketsCounter--;
170             }
171             else if (isOperator(strWithoutBrackets[i]) && bracketsCounter == 0) {
172                 if (strWithoutBrackets[i] == NEG) {
```

```
164     for (size_t i = 0; i < length; i++) {
165         if (strWithoutBrackets[i] == '(') {
166             bracketsCounter++;
167         }
168         else if (strWithoutBrackets[i] == ')') {
169             bracketsCounter--;
170         }
171         else if (isOperator(strWithoutBrackets[i]) && bracketsCounter == 0) {
172             if (strWithoutBrackets[i] == NEG) {
173                 return new UnaryExpression(
174                     NEG,
175                     parseExpression(
176                         strWithoutBrackets.substr(i + 1, length - i - 1)
177                     )
178                 );
179             }
180             else {
181                 return new BinaryExpression(
182                     strWithoutBrackets[i],
183                     parseExpression(strWithoutBrackets.substr(0, i)),
184                     parseExpression(strWithoutBrackets.substr(i + 1, length - i - 1))
185                 );
186             }
187         }
188     }
189
190     return nullptr;
191 }
192
193 ExpressionCalculator::ExpressionCalculator(const MyString& str) : expression(parseExpression(str)) { }
194
```

```
195     ExpressionCalculator::ExpressionCalculator(const ExpressionCalculator& other) {
196         copyFrom(other);
197     }
198
199     ExpressionCalculator::ExpressionCalculator(ExpressionCalculator&& other) noexcept {
200         move(std::move(other));
201     }
202
203     ExpressionCalculator& ExpressionCalculator::operator=(const ExpressionCalculator& other) {
204         if (this != &other) {
205             free();
206             copyFrom(other);
207         }
208
209         return *this;
210     }
211
212     ExpressionCalculator& ExpressionCalculator::operator=(ExpressionCalculator&& other) noexcept {
213         if (this != &other) {
214             free();
215             move(std::move(other));
216         }
217
218         return *this;
219     }
220
221     ExpressionCalculator::~ExpressionCalculator() {
222         free();
223     }
224
225     bool ExpressionCalculator::isTautology() const {
226         return checkAllVariations(expression, true);
227     }
```

```
229     bool ExpressionCalculator::isContradiction() const {
230         return checkAllVariations(expression, false);
231     }
232
233     void ExpressionCalculator::copyFrom(const ExpressionCalculator& other) {
234         expression = other.expression->clone();
235     }
236
237     void ExpressionCalculator::move(ExpressionCalculator&& other) {
238         expression = other.expression;
239         other.expression = nullptr;
240     }
241
242     void ExpressionCalculator::free() {
243         delete expression;
244     }
245
246     ///
```

ЛЕКЦИЯ 15 (ТЕМА 14):

- Type casting.
 - SOLID principles.
 - Design patterns - типове и примери (singleton, factory, flyweight, iterator, command).
 - Линк към [гитхъба на Никола](#)
 - Линк към [гитхъба на Ангел](#)
-

Лекция XV

Type casting

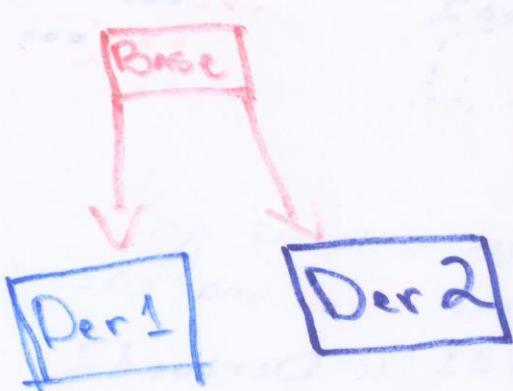
! Начиная с C++11, можно называть генератором

`int*` ~~~~ `[3]`

! 4 способа

I static cast → можно не проверять типы, но это не безопасно
→ No runtime check!!!

Пример:
`int a = 7;`
`double b = a; // static-cast`



`f(Base* ptr);`

`Der d;`

`f(&d);` → незначимое, т.к. правильный cast на d.

`f(static_cast<Base*>(&d));`

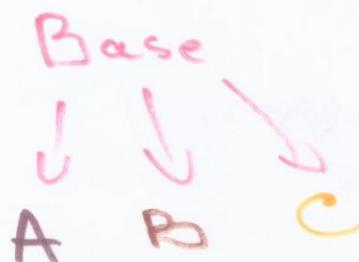
↑
меньше, чем можно
предусмотреть

Одно примеру:

$A^* \text{ptr} = \text{new } A();$

$B^* \text{ptr2} = \text{static_cast}\langle B^* \rangle(\text{ptr});$

ГРЕШКА!!!



$\text{Base}^* \text{ptr} = \text{new } B();$

:

:
ptr вон нахбю е настен



$\text{if}(\text{ptr} \rightarrow \text{get}(\text{type}) == 2)$
 $B^* \text{ptrB} = \text{static_cast}\langle B^* \rangle \text{ptr};$

1 - A

2 - B

3 - C

Пример с библиотека на C:

f(..., callback)

callback(нарареми)

callback(**void***) {

A();

A* myptr=static_cast<**A**>(ptr);

ищо пишем на C++

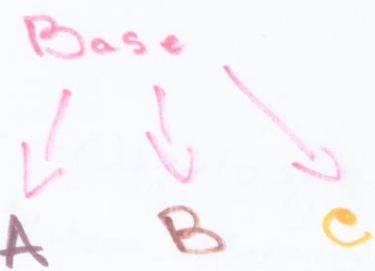
}

II

dynamic-cast → прави runtime check

→ no-джен

→ прави то, което сме
сигурни в наши



f(Base* ptr)

A ptrA=dynamic-cast<**A**>(ptr);

: Ако cast-а не е успешен,

! Ошибко за downcasting връща **nullptr**

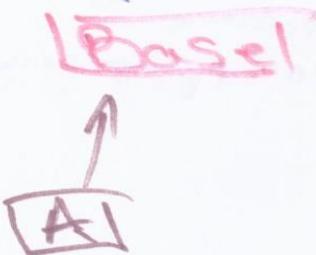
if (**A*** ptr=dynamic-cast<**A**>(ptr))
ptr-наме готов

?

ptr не конструира!



Не бояться down-cast-a падом
Пример:



struct **Base** {

int x

void f() { ... }

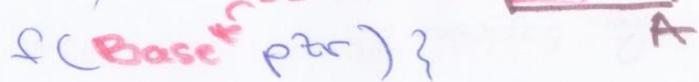
};

struct **A1:Base** {

int y;

{

};



func(Base* ptr) {

A* ptrA = dynamic-cast**A1**(ptr);

! fail! Так как

B Base НЕ ИМЕЕТ

никаких **виртуальных**

функций!!!

Следовательно, casting не ворнет

nullptr !!!

III

const - cast → можем да нанимодифираме
константност на обектъ,
които го дава
чрез употребата (реализация)
на , callback)

callback (const char* ptr) {

char* ptr2 = const_cast<char*>(ptr);
ptr[0] = 'a';

}

Design patterns

- ! Обобщени годри практики
- ! Стандартни решения на често възникващи
проблеми
- ! 3 вида:
 - ! creational patterns → осигуряват начин за
създаване на ~~не~~ обекти, сървантни посредни
по създаването им
 - Пример: factory
- ! structural patterns → осигуряват начин за създава-
не на по-сложни и пасажи
чрез инструменти като
наследяване и композиция
- ! behavior patterns → осигуряват начин за
комуникация между обекти

IV

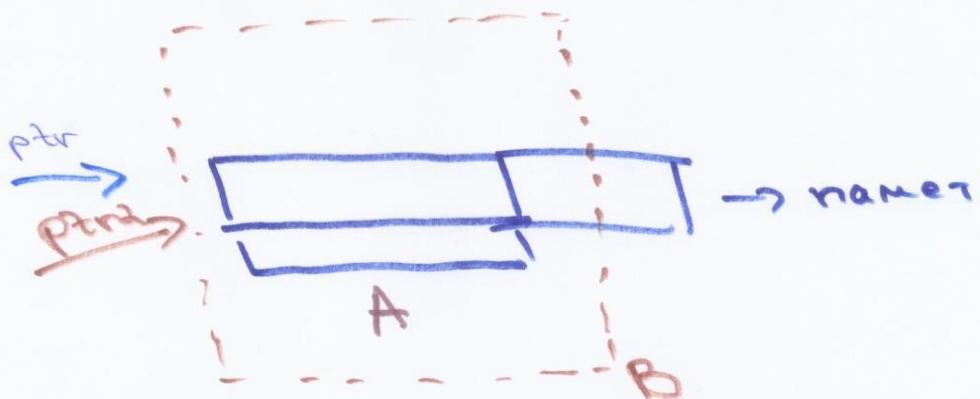
reinterpret-cast → преадресуване на
паметта от произволен
типор, като указател или
произволен тип, без да е
известно какът тип са имат този
один и същ паметници.

Пример:

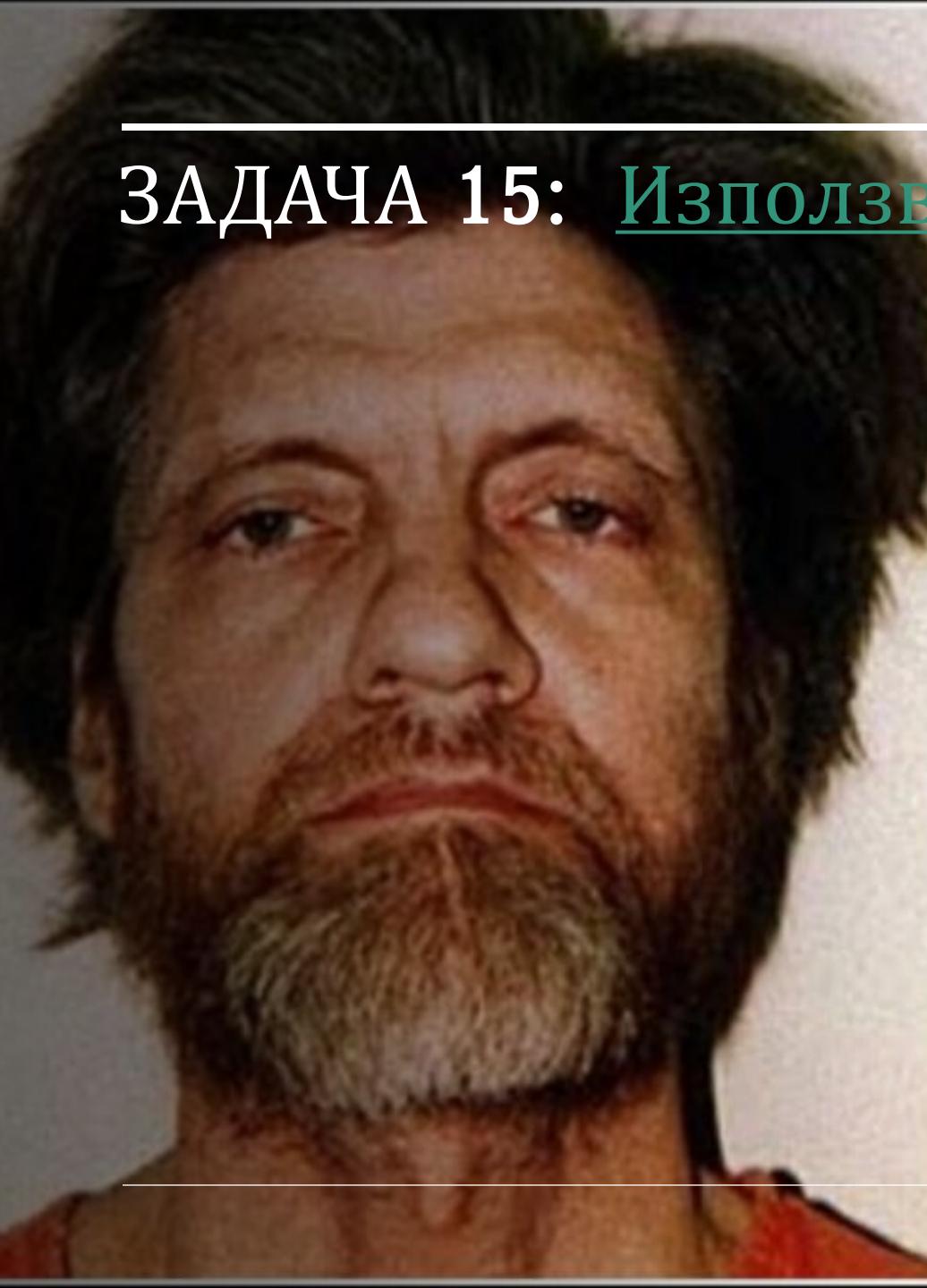
class A, class B

A* ptr = new A();

B* ptr2 = reinterpret-cast<B*>(ptr);



ЗАДАЧА 15: Използване на singleton + factory



The Industrial Revolution and its consequences have been a disaster for the human race.

— *Theodore Kaczynski* —

CREDITS:

- [@Justsvetoslavov](https://github.com/Justsvetoslavov)
- [Angeld55](https://github.com/Angeld55)
- <https://learn.fmi.uni-sofia.bg/course/view.php?id=9048>

**МУЗИКА И ТАНЦИ
КЪСМЕТ И БЕРЕКЕТ
И САМОНА ПРЕД!!!**

**КРАЙ!
УСПЕХ НА ИЗПИТА!!!**