

Тема 1 | Пространства от имена (Namespace). Видове  
енумерации и разлики. Работа с инициализация, достъп до членове, благане, работа с  
Функции, работа с масиви. Размер на обекти/инстанции.  
Поддръжване и отменяване. Обединение. Endianess и  
проверка за little/big endian

## Namespace

- инструмент за избегване на конфликт на имена
- наредба #include в скоби с дефиниции между .TXX

Пример: namespace ns {  
 void f();  
 int global = 5;  
}

main()  
 ns::f();

- using-ключ за резултатна  
настройка scope за добавяне на namespace &

⚠ Конфликт  
namespace A  
 f()

namespace B  
 f()

using namespace A;  
using namespace B;  
f(); // Compile time error t:f() || B::f()

 Анонимен namespace  
- механизъм за реструктуриране на видимостта  
на променливи, обекти, функции

Пример: namespace {  
 void internal();  
}

### Енумерация

- тип, който може да има стойности само от  
предварително дефинирани специални константи (енумератори)

- Енумератор обявява на узло зон

- ако не е указано се изброя следващ +1
- ако на нюней не е дадена стойност тогава = 0

Пример 1) enum class {

red, // 0

blue, // 1

orange // 2

}

2) enum + {

a, // 0

b, // 1

c = 5 // 5

d, // 6

g = a + b // 1

}

Код 2 това енумерации

### Unscoped

- енумераторите са глобални  
променливи, имената им  
са запазени

Пример: enum color {

red,

orange

,

enum fruit {

~~orange~~

,

int orange

Compile Time

• инициализирано преодразяване от енумератор като  
значение

Пример: enum color {  
 red, // 0  
 orange // 1  
}  
enum animal {  
 dog, // 0  
 cat // 1  
}

if (color::red == animal::dog)

// това ще е true

// тази смислова логика

## Unscoped (enum class)

- идентификаторите не са глобални

- няма инициализирано преодразяване

Пример:

enum class Person {  
 male,  
 female  
}

enum class subject {  
 OOP,  
 SDA  
}

int n = person::male;

if (person::male == subject::oop) {

// Compile Time error

}

# Работа с инстанциями

- user defined.
- наследуемость от класса, конто се наследува
- в определен рег.

## Инициализация

A obj;

A \* objPtr = new A();

delete objPtr;

Struct A {

char a[10];

int b;

double c;

}

доступ к элементам

obj.b++;

objPtr->b++;

синтаксична

захар от (\*objPtr).b++;

## Выворот

Struct B {

double va;

A a;

int ci;

}

Работа с объектами от дженерика

1. конк - f1(A obj); 1) ✓ ✓ ✓ ✓ ✓ ✓

2. конк. конк - f2(const A obj); 2) ✗ ✗ ✗ ✗ ✗ ✗

3. перф f3(A & obj); 3) ✓ ✓ ✓ ✓ ✓ ✓

4. конк. перф f4(const A& obj); 4) ✗ ✗ ✗ ✗ ✗ ✗

5. указател - f5(A\* obj); 5) ✓ ✓ ✓ ✓ ✓ ✓

6. конк. указател f6(const A\* obj); 6) ✓ ✗ ✗ ✗ ✗ ✗

Връзките на инициализации от ф-ции

1) A f() { - } ✓

2) A & f() { A obj - } умира в края на scope-a X

3) A \* f() { A obj; return &Obj; } X

(2,3 работи за статични обекти)

4) A \* f() { t \* ptr = new A(); return ptr; } ✓

(напредуващ се граден

5) const A & f() { A obj } ✓ Задължителна

възможност за избора на обекта)

Работа с масиви

A arr[10];

A \* ptr = new A[5];

delete [] ptr;

Размер на обекти/инстанции

Паметта, която можем да използваме.

1. Global (Static)-занизване на статичните, глобални променилни, лгна за члената програма.

2. Stack - памет съдържащ local-ните променилни - малка, бърза, неизпирена от OS (некамер контрол бърз), LIFO, locality

3. Heap (динамичка) - заета се по време на програмата, - При Base  
 - некоја locality (разпределба из паметта), но - давна, незвонка заетището на големи касети памет, неизбежна се от неподредене (с C++)

Size of

ноказва колко байта-а е паметта употребена за градежниот објект

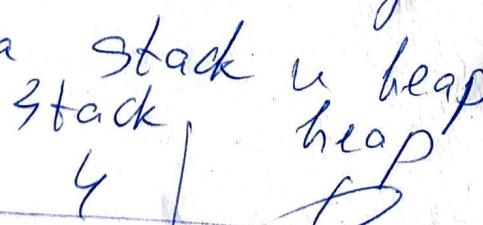
Пример: Struct T {

```
int a;
int b;
```

Size of(T)=8;

Пример: памет на стек и heap

int a=7;



char b='a';

1

bool\* b=new bool[1]

8

1

int\*\* m=new int\*[10]

8

10.8 (pointer)

int\* arr=new int[100];

100.4 (int)

char test[7] = "abc";

0

int arr[10]=80;

0

(6)

примитивните типови често се генерира:

int - 4 байта, double - 8 байти,  
 char 1 байт, bool 1 байт, short 2 байти  
 pointer 8 байти (for x64) and 4  
 bytes for x32)

## Подравяване и отдаване

- в паметта ще бъде място да се подравни инициализирано
- в паметта положението на структурата са предсказани така както са написани в кода.
- структурата ще бъде подравнена по най-голямия член ѝ (наречиен бит). Структура (OS specific)

## Alignment requirement

- разликата между 2 последователни адреса на които можем да разположим една променлива

Tip:  `приети винаги`

$$\text{alignof}(T) = \lceil \text{sizeof}(T) \rceil$$

## Пример:

Struct A {

char a; alignof(A)=4;

int b;

}

Δ Подредба от най-голям към най-малък член

одразу

"B" трябва да започва от адрес

Диаг



одразува се padding, когато е неизползван блок от който зададен от програмата с цел структурата да е правилно подредена

За да не се забърка със заглавията на функции и  
методи га

### △ char arr{} in struct

#### Struct AOptimize {

int b;

char a;

char arr{}; // каза, че ще се  
- използва т-кото бъдешо

масив в такъ

Quar pr. arr

т|т|т|т|т|т|

### △ struct B { }

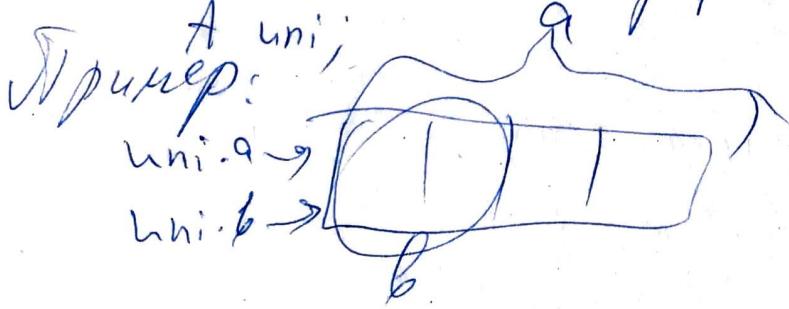
sizeof(B) = ~~11~~ 1

### Oдевищни

- не съговаряне от полета, използвани един и  
същи имена

- user defined

- при работе от 1 глоб. съвместим unique UB



union A {  
int a;  
short b;  
};

## Endianness

- Назад за напримерка на даните във формат int 

α	β	γ	δ
---	---	---	---
- Big endianness 

α	β	γ	δ
---	---	---	---
- little endianness - наименование също означава

ψ	γ	β	α
---	---	---	---

Пример: проверка за little/big endian и същевидно  
bool isLittleEndian()  
{ union A {

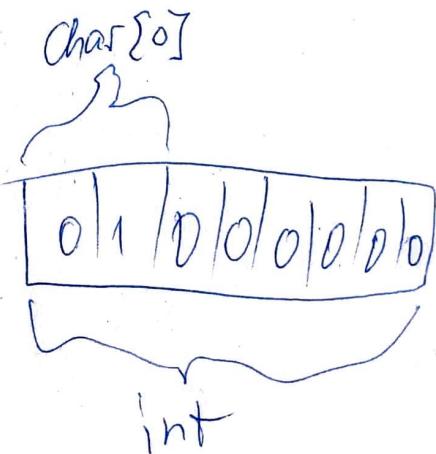
int a;

unsigned char bytes[4];

} myTest;

return myTest.bytes[0];

}



```
#include <iostream>
```

```
namespace Points
```

```
{
```

```
struct Point
```

```
{
```

```
    int x=0;
```

```
    int y=0;
```

```
}
```

```
void readPoint(Point& point)
```

```
{
```

```
    std::cin>>point.x;
```

```
    std::cin>>point.y;
```

```
}
```

```
double getDist(const Point& p1, const Point& p2)
```

```
{
```

```
    int dx = p1.x - p2.x;
```

```
    int dy = p1.y - p2.y;
```

```
    return sqrt(dx*dx + dy*dy);
```

```
}
```

```
void printPoint(const Point& point)
```

```
{
```

```
    std::cout<<"("<<point.x<<","<<point.y<<")";
```

```
}
```

hTriangles.cpp

# namespace Figures

{ using namespace Points;

Struct Triangle

{ Point p1;

Point p2;

Point p3;

}

void readTriangle (Triangle & triangle)

{

readPoint (triangle.p1);

readPoint (triangle.p2);

readPoint (triangle.p3);

}

double getArea (const Triangle & triangle)

{

double sideA = getDist (triangle.p1, triangle.p2);

double sideB = getDist (triangle.p2, triangle.p3);

double sideC = getDist (triangle.p3, triangle.p1);

double halfPer = (sideA + sideB + sideC)/2;

return sqrt(halfPer \* (halfPer - sideA) \* (halfPer - sideB) \* (halfPer - sideC));

}

void SortTrianglesByArea (Triangle\* triangles, int N)

(size\_t size = N)

```
double* areas = new double[N];  
for (int i = 0; i < N; i++)  
    areas[i] = getArea(triangles[i]);  
for (int i = 0; i < N - 1; i++)  
{  
    int minAreaTriangleIndex = i;  
    for (int j = i + 1; j < N; j++)  
    {  
        if (areas[j] < areas[minAreaTriangleIndex])  
            minAreaTriangleIndex = j;  
    }  
    if (minAreaTriangleIndex != i)  
    {  
        std::swap(triangles[i], triangles[minAreaTriangleIndex]);  
        std::swap(areas[i], areas[minAreaTriangleIndex]);  
    }  
    delete[] areas;
```

void printTriangle (const Triangle& triangle)

```
{  
    printPoint(triangle.p1);  
    printPoint(triangle.p2);  
    printPoint(triangle.p3);  
    std::cout << std::endl;
```

```
int main()
{
    size_t N;
```

```
std::cin >> N;
```

Figures::Triangle\* triangleArr = new Fig::Triangle[N];

```
for (size_t i = 0; i < N; i++)
```

Figures::readTriangle(triangleArr[i]);

Figures::sortTrianglesByArea(FIG::angleArr, N);

```
for (size_t i = 0; i < N; i++)
```

Figures::printTriangle(triangleArr[i]);

```
delete[] triangleArr;
```

```
}
```