

# ИЗБРОЕН ТИП. СТРУКТУРИ

доц. Петър Армянов, гл. ас. Дафина Петкова

1

# ИЗБРОЕН ТИП

## ОПИСАНИЕ. ДЕФИНИЦИЯ

---

Потребителски дефиниран тип.

### Логическо описание

- Съставен тип данни, който обединява множество именувани константи, логически свързани помежду си.

```
enum Colour
{
    BLACK,
    GRAY,
    WHITE
};
```

Множеството от стойности на тип **Colour** са всички именувани константи, описани в неговата дефиниция.

# ИЗБРОЕН ТИП

## МНОЖЕСТВО ОТ ДОПУСТИМИ СТОЙНОСТИ

С каква стойност се свързва всяка от константите?

**Без явна** инициализация:

```
enum Colour
{
    BLACK, // BLACK == 0
    GRAY,  // GRAY == 1
    WHITE  // WHITE == 2
};
```

Именуваните константи могат да се инициализират **ЯВНО**.

```
enum Colour
{
    BLACK = 3,
    GRAY,   // GRAY == 4
    WHITE = 7
};
```

# ИЗБРОЕН ТИП

## ПРОМЕНЛИВИ ОТ ТИПА

---

### Дефиниране на променливи

```
Colour colour = GRAY;
```

### Инициализация с цяло число

// стойност от множеството на изброения тип

```
Colour colour = (Colour) 2;
```

// допустимо е да се използва стойност, която не е от изброения тип

```
Colour colour = (Colour) 8;
```

Необходимо е явно преобразуване до типа на изброения тип.

# ИЗБРОЕН ТИП

## ОПЕРАЦИИ

---

### Вход и изход

 `std::cin >> colour;`

`std::cout << colour << std::endl;`

Стойността на променливата `colour` се преобразува неявно до цяло число.

Как да изведем текст?

# ИЗБРОЕН ТИП

## ОПЕРАЦИИ

---

**Операциите == и != са дефинирани.**

```
colour == RED
```

**Не използвайте сравнения от вида:**

```
colour == 7
```

Какво означава 7?

Какво ще се случи, ако бъде добавена нова стойност в дефиницията на изброения тип?

# ИЗБРОЕН ТИП

## КАК ДА ГО ИЗПОЛЗВАМЕ?

---

**Използване в оператора switch.**

```
switch (colour)
{
    case BLACK: std::cout << "Black" << std::endl; break;
    case WHITE: std::cout << "White" << std::endl; break;
    case GRAY: std::cout << "Gray" << std::endl; break;
    default: break;
}
```

Може да има предупреждение от компилатора, ако не са използвани всички стойности на изброяения тип.

# ИЗБРОЕН ТИП

## КАК ДА ГО ИЗПОЛЗВАМЕ?

---

Ако именуваните константите са с **последователни стойности**:

- Как да проверим дали дадена стойност е валидна за изброяения тип?

```
colour >= BLACK && colour <= WHITE
```

- Как да обходим всички именувани константи в изброяения тип?

```
for( int colour = BLACK; colour <= WHITE; ++colour)  
    print( (Colour) colour );
```

Какво ще се промени, ако бъде добавен нов елемент към изброяения тип?

Преди първия? След последния?



# ИЗБРОЕН ТИП

## КАК ДА ГО ИЗПОЛЗВАМЕ?

Ако именуваните константите са с **последователни стойности** и **първата валидна стойност е 0**, обхождането може да се унифицира така:

```
enum Colour
{
    UNKNOWN = -1, // невалидна стойност

    WHITE,
    RED,
    YELLOW,
    BLACK,
    GRAY,

    COLOUR_COUNT // има смисъл на брой на валидните стойности
};
```

```
for (int colour = UNKNOWN + 1;
     colour < COLOUR_COUNT;
     ++colour)
    ...
```

# СЪСТАВНИ ТИПОВЕ ДАННИ

---

Какво представляват съставните типове данни?

*Комбинират или използват фундаментални типове данни.*

Какви съставни типове данни са разгледани до тук?

*Указател;*

*Псевдоним;*

*Масив;*

*Изброен тип и др.*

# СТРУКТУРИ

---

## Логическо описание

- Крайна редица от елементи, които могат да бъдат от **различен** тип.
- Описват една обща характеристика, един общ обект.

## Защо са необходими?

- Организирант логически свързани данни в едно цяло, улеснява работата с данните.
- Всяка модификация е по-лесна.

# СТРУКТУРИ

## ДЕФИНИЦИЯ

```
struct Student
{
    char id[8];
    char name[36];
    char program[24];
    int year;
    double averageGrades;
};
```

- Структурата е **крайна** редица от елементи, които могат да бъдат от **различен тип**.
- Отделните елементи на структурата наричаме още **полета** на структурата.
- Типовете на полетата трябва да бъдат предварително определени.

# СТРУКТУРИ

## ПРОМЕНЛИВИ

```
Student me;
```

```
Student me = {  
    "01M1234",  
    "Me",  
    "Informatics",  
    1,  
    5.78 };
```

- Ако автоматична променливата **не е инициализирана**, то всяка от характеристиките ѝ има **случайна стойност**.
- Ако при **инициализация**, броят на изброените стойности е **по-малък** от броя на характеристиките на структурната променлива, **останалите** се инициализират с **нули**.

# СТРУКТУРИ

## ДОСТЪП ДО ПОЛЕТАТА НА СТРУКТУРАТА

```
strcpy( me.id , "03M1234" )  
std::cout << me.id << std::endl;
```

```
me.year++;
```

```
me.program[0] = "i";
```

Достъпът се осъществява с **оператор точка „.”**

- достъп за четене;
- достъп за писане.

Всяко поле може да се разглежда като променлива от съответния тип.

Всички операции, дефинирани над този тип, са възможни и за полето.

# СТРУКТУРИ

## ОПЕРАЦИИ НАД СТРУКТУРАТА

```
Student me;
```

```
std::cin >> me;
```

```
std::cout << me << std::endl;
```

Операциите за вход и изход

**не са дефинирани за**

потребителски дефинирани типове данни.

# СТРУКТУРИ

## ОПЕРАЦИИ НАД СТРУКТУРАТА

```
Student me;  
Student you = {  
    "01M1234",  
    "Me",  
    "Informatics",  
    1,  
    5.78 };
```

**me = you;**

Операцията за присвояване е дефинирана.

**me** и **you** трябва да бъдат от **един и същи тип!**

Копирането на информацията е побитово, едно към едно.

Какво ще се случи, ако в структурата има указател (външен ресурс)?



# СТРУКТУРИ

## УКАЗАТЕЛИ КЪМ СТРУКТУРИ

---

```
Student me = {"01M1234", "Me", "Informatics", 1, 5.78};  
Student* pMe = &me;
```

**Как достъпваме полетата на структурната променлива, към която сочи указателят?**

```
(*pMe).averageGrades = 5.00;
```

Операторът . е с по-висок приоритет от \*.

По-опростен синтаксис с **оператор ->**:

```
strcmp(pMe->name, "You")
```

# СТРУКТУРИ

## ПСЕВДОНИМИ

---

```
Student me = {"01M1234", "Me", "Informatics", 1, 5.78};  
Student& refMe = me;
```

**Достъпът до полетата се извършва с оператор .**

```
refMe.year = 2;  
strncmp(refMe.id, "01M", 3)
```

# СТРУКТУРИ

## КАТО ПАРАМЕТРИ ИЛИ РЕЗУЛТАТ ОТ ФУНКЦИИ

---

### Предаване по стойност

Създава се **копие** на фактическия параметър, подаден на функция.

Две променливи (оригиналната структурна променлива и копието).

```
void print(Student student);
```

### *Особености:*

- Не се работи с оригиналните данни, т. е. те не се променят.
- Потокът на информацията е еднопосочен (само към функцията).
- Ако входните данни са обемни, функциите нарастват значително, копирането отнема време, тежка операция.
- Могат да се подават литерали и временни обекти.

# СТРУКТУРИ

## КАТО ПАРАМЕТРИ НА ФУНКЦИИ

---

### Предаване по указател

Подава се **адресът** на структурната променлива.

Две променливи (оригиналната структурна променлива и указателят, който сочи към нея).

```
void read(Student* pStudent);
```

### Особености:

- Не се прави копие, получава се директен достъп до паметта, в която е съхранен фактическият параметър.
- Потокът на информация е двупосочен.
- Входните данни могат да бъдат променени.
- Не могат да бъдат подадени литерали или временни обекти.

Дали `nullptr` е валидна стойност за функцията?

# СТРУКТУРИ

## КАТО ПАРАМЕТРИ НА ФУНКЦИИ

---

### Предаване по референция

Подава се **самата** структурна променлива. Една променлива с две имена.

```
void read(Student& student);
```

*Особености:*

- Не се прави копие. Не се изисква допълнителна памет.
- Формалният параметър е друго име на фактическия. Директен достъп.
- Потокът на информация е двупосочен.
- Входните данни могат да бъдат променени.
- Не могат да бъдат подадени литерали или временни обекти.

# СТРУКТУРИ

## УКАЗАТЕЛИ И ПСЕВДОНИМИ

```
Student me = {...};  
// pointer to me  
Student* pMe = &me;  
// pointer to const me  
const Student* pCMe = &me;  
// const pointer to me  
Student* const cPMe = &me;  
// const pointer to const me  
const Student* const cpCMe = &me;
```

```
// reference to me  
Student& refMe = me;  
// reference to const me  
const Student& refConstMe = me;
```

Кои операции са възможни във всеки от случаите?

# СТРУКТУРИ

## КАТО РЕЗУЛТАТ ОТ ФУНКЦИИ

---

Как да върнем структурна променлива като **резултат от функция**?

```
Student read();
```

Ако структурната променлива е създадена **локално** във функцията, може да бъде върната само **по стойност**.

# ВЛАГАНЕ НА СТРУКТУРИ

Възможно е влагане на една структура в друга.

```
struct Address
{
    char city[32];
    char street[64];
    unsigned short number;
};

Address myAddress =
    {"Sofia", "Madrid", 35};
```

```
struct Student
{
    char id[8];
    char name[36];
    Address address;
    ...
};

Student me =
    {"123", "Me", myAddress, };


std::cout << me.address.city;
```



# ВЛАГАНЕ НА СТРУКТУРИ


*Ограничения:*

Една структура **не може да бъде вложена в себе си**. Не може да се определи размерът ѝ. Може да се дефинира обаче указател към тази структура.



```
struct Student
{
    char id[8];
    char name[36];
    Student mentor;
    ...
};
```

```
struct Student
{
    char id[8];
    char name[36];
    Student* mentor;
    ...
};
```



# ВЛАГАНЕ НА СТРУКТУРИ

*Ограничения:*

Една структура **не може да бъде вложена в себе си**. Не може да се определи размерът ѝ.

Може да се дефинира обаче указател към тази структура.

```
struct Student
{
    char id[8];
    char name[36];
    Student* mentor;
    ...
};
```

```
Student mentor =
{"1", "Ivan Ivanov", nullptr, };
// менторът трябва да е дефиниран,
// трябва да надживее студента ме
Student me =
{"123", "Me", &mentor, };
```

# ПОЛЕЗНИ ИЗТОЧНИЦИ

---

- <https://en.cppreference.com/w/cpp/language/typedef>
- <https://stackoverflow.com/questions/22646165/what-is-the-use-of-enums-in-c-and-c>
- <https://www.programiz.com/cpp-programming/enumeration>
- <https://en.cppreference.com/w/cpp/language/enum>
- <https://www.informatika.bg/lectures/variables-structs-arrays>
- <http://www.cplusplus.com/doc/tutorial/structures/>
- <http://www.informit.com/articles/article.aspx?p=31783&seqNum=2>
- <http://en.cppreference.com/w/cpp/language/union>
- [http://en.cppreference.com/w/cpp/language/bit field](http://en.cppreference.com/w/cpp/language/bit_field)
- <http://www.geeksforgeeks.org/bit-fields-c>
- <http://blog.aaronballman.com/2011/08/the-joys-of-bit-fields>
- <http://stackoverflow.com/questions/2310483/purpose-of-unions-in-c-and-c>