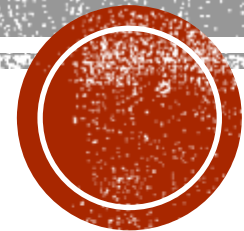


# ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

СРЕЩА 12 Структури и обединения



# Глобална променлива

Променлива, която може да има достъп както вътре във функция, така и извън `main()` функцията. Такава променлива се нарича още **extern** променлива.

Декларира се извън `main()` и има начална стойност 0

```
int x;
```

```
int main(){  
int x;
```

```
}
```

```
int main(){
```

```
extern
```

```
}
```



# Структури

- структурата комбинира елементи от различен тип
- позволяват влагане на структури
- структурата има членове, всеки член е от различен тип
- рекурсивен тип (може да има член указател към същата структура)
- за дефиниране се използва ключовата дума struct

```
struct tagStruct {  
    CType1 m_memeber1;  
    CType2 m_memeber2;  
    ...  
    CType3 m_memeber3;  
    struct tagStruct* m_pstNext;  
};
```



# Дефиниция, декларация, инициализация

- дефиниция и инициализация

```
struct tagSTest {  
    int m_iNum;  
    float m_fRate;  
};  
struct tagSTest a = { 1 , 5.5 };  
struct tagSTest b = { .m_fRate=15.15, .m_iNum=10000};
```

- декларация

```
extern struct tagSTest a;
```



# Достъп до елементите на структура

```
struct tagSTest {  
    int m_iNum;  
    float m_fRate;  
    struct tagSTest *test;  
};
```

```
struct tagSTest stVar = { 1, 5.5 };  
struct tagSTest* pstVal = &stVar;
```

```
stVar.m_iNum = 13;  
pstVal->m_fRate = 6.78; /* достъп чрез указател */
```



# Разполагане в паметта

- подравняване - членовете се разполагат последователно в паметта, като се **подравняват в зависимост от архитектурата на процесора**

```
struct tagTest {  
    char m_chValue;  
    unsigned short m_usValue;  
}; /* sizeof(struct tagTest) -> 4 Bytes */
```

- пакетиране

```
struct tagTest { /* sizeof(struct tagTest) -> 3 Bytes */  
    char m_chValue;  
    unsigned short m_usValue;  
} __attribute__((packed)); /* зависи от компилатора */
```



# Структура като параметър на функция

- предаване по стойност, подобно на базов тип (копира се цялата структура в стека)

```
void doCalc(struct tagData stData /*входен параметър*/);
```

- предаване чрез указател

```
void doCalc(struct tagData* pstData);
```



# Връщане на структура от функция

- връщане по стойност, подобно на базов тип (копира се цялата структура в стека)

```
struct tagData getData(void);
```

- предаване чрез указател (при използване на динамична памет)

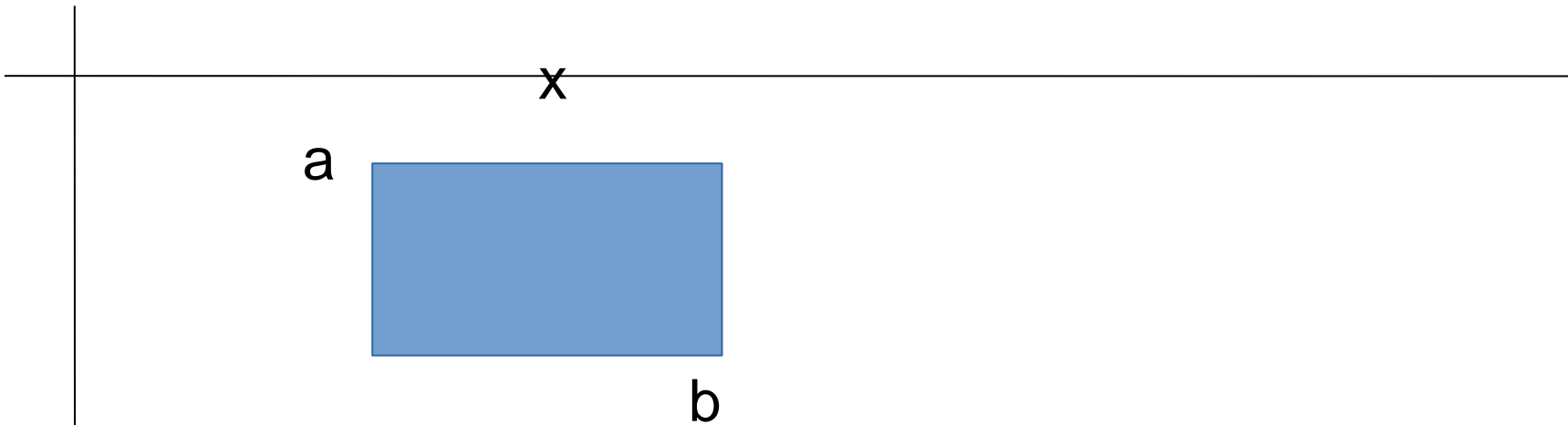
```
struct tagData* getPtrData(void);
```





# Структура

Структурите могат да се вграждат. Нашата реализация на правоъгълник представлява две точки, които обозначават противоположните ъгли по единия от диагоналите на правоъгълника:



```
struct rect {  
    struct point a;  
    struct point b;  
};
```



# Структура – упражнение

Задача 1. Напишете програма, в която информацията да бъде съхранявана в структура, описваща автомобил. Входната информация трябва бъде въведена от потребителя, като напишете меню с въпроси към него. Принтирайте въведената информация за описание на автомобила.

Задача 2. Напишете програма, която събира две дистанции, които са изразени в километри, метри, сантиметри. Дистанциите трябва да бъдат въведени от потребителя. Принтирайте изходните и резултатната дистанция.

**ДОМАШНО** Задача 3. Напишете програма, която калкулира разликата на два времеви периода, изразени във векове, години, месеци, дни, минути, секунди. Принтирайте изходните периоди и резултата.



# Структура – упражнение

Задача 4. Напишете произволна програма, която да демонстрира уменията ви да боравите с **nested structure**.

Задача 5. Напишете произволна програма, която демонстрира уменията ви да боравите с указатели към структури. Нека декларираната от вас структура(и) съдържа указател към собствения си тип.

Задача 6. Напишете програма, която да изчислява средния успех на всеки студент и целия курс, използвайки структури. Входните данни за студентите трябва да бъдат въведени от потребителя. Принтирайте резултатите за всеки студент поотделно, както и за целия курс.

.



# Задачи за структура

Задача 7. Направете функция `struct point makepoint(int x, int y)`, която създава две точки. Използвайте `malloc()`.

**ДОМАШНО** Задача 8. Направете структура `struct rect`, която съдържа в себе си две точки. Създайте обект от тип тази структура наречен `screen`. Използвайте функцията `makepoint` за да зададете начална точка на екрана (0, 0) и крайна точка (15, 15). Запълнете пространството между тях с тирета.

Задача 9. Дефинирайте структура, в която има член указател към самата структура. Използвайте тази структура, за да дефинирате две променливи, които се указват една друга.



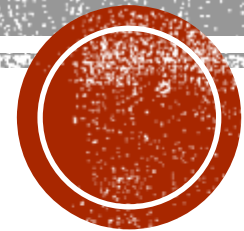
# Задачи за структура

Задача 10\*. **ДОМАШНО** -да се довърши в детайли **Представете служителите във фирма в структура с членове: пореден номер, име, презиме, фамилия, позиция, трудов стаж в години, заплата в лева, указател към структурата, описващ неговия началник. Напишете програма, която въвежда 10 служителя, които се съхраняват в масив от описаните структури. Въвеждането на служителите може да стане на два паса, първо въвеждане на всички данни без указателя към началника и на втори пас, указване на всеки служител кой е неговият началник.**



# ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Среща 12 typedef Type Casting



# Потребителски типове

- 1.представяват псевдоними на съществуващи типове
- 2.добавя се име на тип, а не се създава нов тип
- 3.използва се ключовата дума **typedef**
- 4.правят програмата по-четима
- 5.обикновено се поставя в заглавни файлове, за да е видим

псевдонимът в целия проект

```
typedef C-type t_Alias;
```

Пример:

```
typedef unsigned short int t_uint16; /* навсякъде,  
където използваме името t_uint16, ще означава unsigned  
short int */
```



# Потребителски типове

```
typedef unsigned char t_Byte;  
typedef unsigned char t_8Bits;  
typedef t_8Bits TBYTE;
```

```
typedef struct { int x, y, z; } t_Point;  
typedef union { int nVal; double dfVal; } t_value;
```

```
typedef enum { FALSE, TRUE } t_bool;
```

```
typedef int* t_ptrInt;
```

```
/* много често се използва за указател към функция */  
typedef int ( * TFnCallback ) (int, int);
```





# typedef и Структурите

typedef ще използваме главно за дефиниране на тип на структура, което е много удобно, за да се избегне писането на ключовата дума struct, когато дефинираме променливи от тази структура:

```
#include <stdio.h>
typedef struct some_struct { char c; int i; } some_struct_t;
int main(void){
    some_struct_t t0 = {.c = 'a', .i = 0};

    struct some_struct t1 = { .i = 100, .c = 'A'};
    return 0;
}
```



# typedef и указатели

Ако дефинираме тип на структура без тяло - `typedef struct node node_t;`, компилаторът не знае какво е съдържанието на структурата `node_t` и затова не може да декларира променлива от този тип, защото не са описани членовете на структурата. Но компилаторът винаги може да направи указател от тип `node_t`, който по късно да се напълни със съдържание.

```
typedef struct node node_t;  
typedef struct node* ptr_node_t;  
struct node{  
    int v;  
    ptr_node_t ptrNode;  
};
```



# typedef

Трябва да отбележим, че `typedef` декларацията не създава нов тип, тя просто добавя ново име на вече съществуващ тип. Също така не съществува нова семантика на променливите: променливите, декларирани по този начин, имат абсолютно същите свойства, както и променливите, чиито декларации описва.



# typedef

Съществуват две основни причини за употребата на `typedef`:

Първата се отнася до поставянето на параметри в програмата, които ограничават проблемите с преносимостта. Ако `typedef` се използва за типове данни, зависещи от архитектурата на машината, когато програмата се премести на друга архитектура, единственото, което трябва да се промени, са `typedef` декларациите, т.е. ако трябва да променим всички променливи от **`int`** на **`long int`** трябва да минем по целия код и да го променим навсякъде:

```
typedef int int_t;
```

```
typedef long int int_t;
```

Втората да използвате `typedef` е свързана с по-добрата документация на програмата. Например тип наречен **`treeptr_t`** ни казва, че това е указател към сложна структура от тип дърво, което за ползващия е по лесно за разбиране.



# C - Type Casting

По същия начин декларацията на :

```
typedef char* String;
```

прави `String` синоним на `char*`, като в последствие може да се използва променлива от тип `String`, за която да заделим памет по следния начин:

```
String s = (String)malloc(93);
```

Като направим кастване към дефинирания тип `String`.



# Стандартни Вход / Изход / Грешки

при стартиране на програма, ОС отваря 3 файла (вход, изход, грешки)

С програмата третира устройствата като файлове

**stdin** - стандартен вход (напр. клавиатура)

**stdout** - стандартен изход (напр. дисплей)

**stderr** - стандартен изход за грешки

```
user ---> device { keyboard } --->  
-> stdin ---> { C program } ---> stdout, stderr ->  
---> device { screen } ---> user
```



# Задачи

Задача 1. Дефинирайте структура като потребителски тип. Инициализирайте членовете на структурата, използвайки новия потребителски тип. Отпечатайте.

Задача 2. Създайте нов потребителски тип към тип `long long int`. Използвайте го във функцията `printf`, отпечатайте размера.

Задача 3. Дефинирайте потребителски тип към указател. Създайте променлива, насочете указателя към нея, използвайки новия потребителски тип.



## ДОМАШНО – да се довърши

**Задача 4. Дефинирайте тип указател към функция, която изпълнява определена операция върху две целочислени променливи. Дефинирайте функции, които изпълняват операциите +, -, \*, /. Използвайте новия тип, за да извикате всяка една от функциите.**

```
typedef int (*Ptr)(int,int);
```

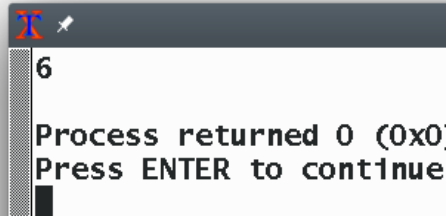
Вместо кода

```
int (*Convert(const char code)) (int, int) {  
    if (code == '+') return &Sum;  
    if (code == '-') return &Difference;  
}
```

записваме

```
Ptr Convert(const char code) {  
    if (code == '+') return &Sum;  
    if (code == '-') return &Difference;  
}  
  
int main ( ) {  
    Ptr ptr;  
    ptr = Convert('+');  
    printf( "%d \n", ptr(2,4));  
}
```

```
#include<stdio.h>  
  
typedef int (*Ptr)(int,int);  
  
int sum(int x, int y) {  
    return (x+y);  
}  
  
Ptr Convert(const char code) {  
    if (code == '+') return &sum;  
    //if (code == '-') return &Difference;  
}  
  
int main ( ) {  
    Ptr ptr;  
    ptr = Convert('+');  
    printf( "%d \n", ptr(2,4));  
}
```



```
6  
Process returned 0 (0x0)  
Press ENTER to continue
```

<http://www.cs.cmu.edu/~ab/15-123N09/lectures/Lecture%2008%20-%20Function%20Pointers.pdf>





## Задачи

Задача 5. Дефинирайте потребителски тип към масив. Инициализирайте масива, изведете на конзолата.

**Пример - дефиниране на матрица- двумерен масив с typedef**

```
typedef int red[4];
typedef red matrix[3];
int main()
{
    matrix M;
}
```

**M[2]** е от тип **red**

**M[2][1]** е от тип **int**

**M** е от тип **matrix**

Задача 6. Напишете структура с потребителски тип **key\_t**, която съдържа символен низ и число. Създайте променлива от новия тип, като инициализирате символния низ с динамично заделена памет за него. Принтирайте съдържанието на структурата.



# Задачи

**ДОМАШНО** – да се довърши **Задача 7**.

Напишете масив от структури наречен `key_tab[ ]`, като използвате тази, дефинирана в горното упражнение - потребителски тип `key_t`, която съдържа символен низ и число. Инициализирайте масива с всички ключови думи на C, като заделяте паметта за всяка ключова дума динамично. Принтирайте масива.



# C - Type Casting

**ДОМАШНО** Задача 8. Напишете собствен тип за структура `node`, съдържаща един член от тип `int` и един указател към тип самата структура. Заделете динамично памет за масив от 10 елемента от тази структура с `malloc`. За всеки елемент от масива попълнете цялото число с неговия пореден номер. Принтирайте резултатите.

**ДОМАШНО – да се довърши** Задача 9. Представете служителите във фирма в структура с членове: пореден номер, име, презиме, фамилия, позиция, трудов стаж в години, заплата в лева, указател към структурата, описващ неговия началник. Напишете програма, която въвежда 10 служителя, които се съхраняват в масив от описаните структури. След като въведете всички служители, задайте началник за всеки от тях. В решението трябва да се използва динамично заделяне на памет и `typedef`.



## ДОМАШНО Задача 10.

Създайте структура диня с два елемента - диаметър и дебелина на кората заделете динамично с `malloc()` за масив от A на брой дини от сорт мелон и попълнете данните за диаметър между 15 и 140 см с функцията `rand()`, за всяка една диня в масива и дебелина на кората между 0.5 и 3.5 см. Създайте структура диня с два елемента - диаметър и дебелина на кората заделете динамично с `malloc()` за масив от B на брой дини от сорт пъмпкин и попълнете данните за диаметър между 35 и 95 см с функцията `rand()`, за всяка една диня в масива и дебелина на кората между 0.3 и 0.9 см. Намерете средната големина на динята и средната дебелина на кората и ги съпоставете с тези от втория масив. Изведете на екрана купчината от кой сорт е по добре да се купи.

Изход: По-добре е да се купят ... дини с диаметър ... сантиметра и кора с дебелина D см вместо ... дини с диаметър ... см и дебелина на кората D1 см.



**ДОМАШНО** Задача 11. Създайте структура **fraction** (аритметична дроб) с **член данни** **int n**(nominator=числител) и **int d** (denominator=знаменател)

С помощта на **typedef struct** създайте **нов тип** **FRACTION** и **указател към него** **\*FPTR**

Дефинирайте функциите

**FRACTION create(int numerator, int denominator);**-създаване на дроб по дадени стойности на числител и знаменател

**FRACTION input(void);**-създаване на дроб по стойности въведени от клавиатурата

**void print(FPTR);** - отпечатване на дроб

**int gcd(int first\_dividend, int second\_dividend);** - НОД

**FRACTION add(FPTR, FPTR);** -събиране на две дроби

**FRACTION mult(FPTR, FPTR);**-умножение на две дроби

**FRACTION divide(FPTR, FPTR);**-деление на две дроби

**FRACTION subtract(FPTR, FPTR);**-изваждане на две дроби

