ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

# СТРУКТУРИ,
## битски операции, енумерација

**–Податочни структури и програмирање–**

# Дефинирање на структури (1)

- Структурите претставуваат **колекции (агрегации) на логички поврзани податоци** од различни (нехомогени) видови **во единствена целина**, заради поедноставна претстава и манипулација
- Во комбинација со покажувачите претставуваат **основа за** креирање на **сложени динамички структури на податоци** како листи, магацини, редови, стебла, ...
- **Општиот облик** за дефинирање на **структури во C** е следниот:

```
struct ime_na_strukturata
{
      field_type field_name; /* deklaracii na promenlivi */
      field_type field_name;
      ...
} lista_na_promenlivi;
```

# Дефинирање на структури (2)

```
struct student
{
     char ime[30];
     int indeks;
     float prosek;
};
```

```
struct produkt
{
     char ime[30];
     float cena;
};
```

- **`struct`** дефинира структура со име **`student`**
- **`student`** е името на структурата (податочниот тип) со кое се **декларираат променливи** од типот структура
- `student` содржи **три елементи** (членови, компоненти):
  - една низа од 30 знаци – `ime;`
  - еден цел број – `indeks;` и
  - еден децимален број – `prosek.`

# Правила кај структурите (1)

- Структурите **не можат** да содржат член што претставува структура од истиот тип (инстанца на самите себе)
- Структурите **може** да содржат член што претставува структура од друг тип
- Структурите **може** да содржат член покажувач кон структура од истиот тип
- Со **дефиницијата** на структура не се резервира никаков мемориски простор, туку само се **воведува нов податочен тип** од кој ќе може да се **декларираат променливи** од типот структура

# Правила кај структурите (2)

- Со `struct student najdobar;` **се креира променлива** за структурата со име `najdobar` и со форма како што е дефинирана за структурата `student`
- Променливите од тип структура **може да се декларираат и во продолжение**, веднаш по дефиницијата на структурата:

```
struct student
{
     char ime[30];
     int indeks;
     float prosek;
} najdobar, najubav;
```

- Променливите од тип структура **може да се иницијализираат** на следниот начин:

```
struct student najdobar = { "Jas", 287, 8.25 };
struct produkt x = { "mleko", 40 };
```

# Правила кај структурите (3)

- **До членовите** (компонентите) на структурата **се пристапува** со името на променливата од тип структура и името на членот на структурата раздвоени со операторот точка **(“.”)**

```
najdobar.indeks = 601;
printf("%s", najdobar.ime)
scanf("%f", &najdobar.prosek);
```

- **Областа на важење** на променливите декларирани во структурата **е во самата дефиниција** на структурата
- Имињата на членовите на структурите се независни од другите променливи со исто име дефинирани надвор од структурата:

```
struct s
{
     int i;
     int j;
} a;
```

```
int i = 10;
a.i = 20;
a.j = 50;
printf("%d   %d   %d \n", i, a.i, a.j);
```

```
10  20  50
```

# Правила кај структурите (4)

- **Полиња (низи) од структури може да се формираат** на следниот начин:

```
struct student prva_godina[500];
```

при што секој елемент на полето `prva_godina` е структура од типот `student`.

- **До елементите** (членовите) на структурата што се наоѓа **на позиција `i`** во полето, **се пристапува** на следниот начин:

```
prva_godina[i].indeks
```

- Првата буква од името на `i`-тиот студент:

```
prva_godina[i].ime[0]
```

# Правила кај структурите (5)

- Често се јавува потреба членови на една структура да бидат други структури што се нарекува и **вгнездување на структури**:

```
struct feit_student
{
      char pol;
      char adresa[20];
      struct student lice;
      float stipendija;
} apsolventi[100];
```

- Првата буква од името на `i`–тиот студент:

```
apsolventi[i].lice.ime[0]
```

- **Покажувач на структура** се декларира со:

```
struct student *pok;
```

# Операции со структури (1)

Единствени операции што се дозволени со структурите се:

- **(=) доделување** на вредноста на променлива од типот структура на друга променлива од истиот тип;

```
najdobar = najubav;
```

- **(&) земање на адресата** на променлива од типот структура;

```
pok = &najdobar;
```

- **(.) пристап до членовите** на структурата;

```
int i = najdobar.indeks;
```

- **(sizeof) одредување на должината** на структурата;

```
sizeof(struct student) или sizeof(najdobar)
```

# Операции со структури (2)

- Со структурите **често се работи преку покажувачи**. Пристап до членовите на структура кон која покажува даден покажувач може да се оствари со операторот '**->**'.
- Пример **програма 2.1**

```
#include <stdio.h>
#include <string.h>
struct st
{
      int i;
      char niza[80];
} s, *p;
void main()
{
      p = &s;
      s.i = 10;
      p->i = 20;
      strcpy(p->niza, "Mnogu sakam strukturi");
      printf("%d %d %s\n", s.i, p->i, p->niza);
}
```

```
p->i
е исто со: (*p).i
```

```
20 20 Mnogu sakam strukturi
```

# Операции со структури (3)

- Ако ја земеме предвид претходната структура

```
struct student
{
    char ime[30];
    int indeks;
    float prosek;
} najdobar, najubav;
```

и дополнително поставиме

```
struct student *pok;
pok = &najdobar;
```

`pok->indeks = 123;`
**е идентично со**
`(*pok).indeks = 123;`

`najdobar.prosek`
**е идентично со**
`(&najdobar)->prosek`

# typedef

- Креира **синоним за веќе дефиниран тип**
- Вообичаено се употребува за креирање скратени имиња на типови, на пример

```
typedef int celbroj;
typedef float realen;
celbroj I, j;
realen x, y;
```

- Може да се користи и `typedef struct student kandidat;` по што наместо `struct student najdobar;` можеме да пишуваме и само `kandidat najdobar;`

```
typedef kandidat * kan_ptr;
kan_ptr kp = &najdobar;
```

# Пример 2.2

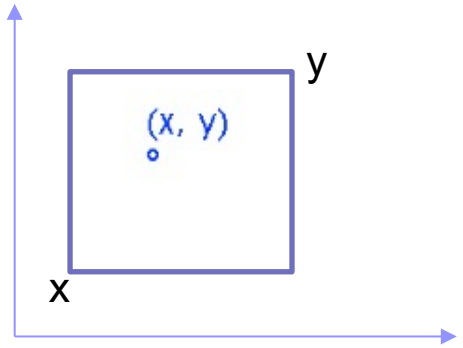
```
#include <stdio.h>

typedef struct point {
      float x;
      float y;
} tocka;

typedef struct rect {
      tocka pt1; //dolu levo
      tocka pt2; //gore desno
} pravoagolnik;

tocka makepoint(float x, float y)
{
      tocka temp;
      temp.x = x;
      temp.y = y;
      return temp;
}
```

```
pravoagolnik makerect(tocka x, tocka y)
{
      pravoagolnik temp;
      temp.pt1 = x;
      temp.pt2 = y;
      return temp;
}
```



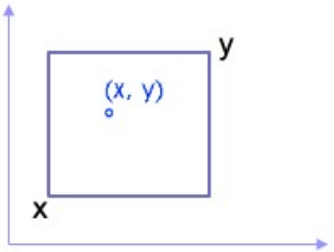
# опција (1)

```
/* ptinrect: vraka 1 ako p e vo r, 0 inaku */
int ptinrect(tocka p, pravoagolnik r)
{ return p.x >= r.pt1.x && p.x < r.pt2.x && p.y >= r.pt1.y && p.y < r.pt2.y; }

tocka centar(pravoagolnik p)
{ return(makepoint((p.pt1.x + p.pt2.x) / 2, (p.pt1.y + p.pt2.y) / 2)); }

float povrsina(pravoagolnik r)
{ return((r.pt2.x - r.pt1.x)*(r.pt2.y - r.pt1.y)); }

int main()
{
      tocka pt1, sredina; pravoagolnik prozor1, prozor2;
      pravoagolnik ekran = { { 2,3 },{ 4,5 } };
      pt1 = makepoint(0, 0);
      prozor1.pt1 = pt1;
      prozor1.pt2 = makepoint(5, 5);
      sredina = centar(ekran);
      prozor2 = makerect(sredina, makepoint(5, 5));
      printf("Tockata (%4.2f,%4.2f) se naogja %s pravoagolnikot
            (%4.2f, %4.2f) : (%4.2f, %4.2f)\n", sredina.x, sredina.y,
            (ptinrect(sredina, prozor1) ? "vo" : "nadvor od"),
            prozor1.pt1.x, prozor1.pt1.y, prozor1.pt2.x, prozor1.pt2.y);
      printf("Tockata (%4.2f,%4.2f) se naogja %s pravoagolnikot
            (%4.2f, %4.2f) : (%4.2f, %4.2f)\n", pt1.x, pt1.y,
            (ptinrect(pt1, prozor2) ? "vo" : "nadvor od"),
            prozor2.pt1.x, prozor2.pt1.y, prozor2.pt2.x, prozor2.pt2.y);
}
```




```
Tockata (3.00,4.00) se naogja vo
pravoagolnikot (0.00, 0.00) : (5.00, 5.00)
Tockata (0.00,0.00) se naogja nadvor od
pravoagolnikot (3.00, 4.00) : (5.00, 5.00)
```

# опција (2)

```
/* ptinrect: vraka 1 ako p e vo r, 0 inaku */
int ptinrect(const tocka *p, const pravoagolnik *r)
{ return p->x >= r->pt1.x && p->x<r->pt2.x && p->y >= r->pt1.y && p->y<r->pt2.y; }

tocka centar(const pravoagolnik *p)
{ return(makepoint((p->pt1.x + p->pt2.x) / 2, (p->pt1.y + p->pt2.y) / 2)); }

float povrsina(const pravoagolnik *r)
{ return((r->pt2.x - r->pt1.x)*(r->pt2.y - r->pt1.y)); }

int main()
{
      tocka pt1, sredina; pravoagolnik prozor1, prozor2;
      pravoagolnik ekran = { { 2,3 },{ 4,5 } };
      pt1 = makepoint(0, 0);
      prozor1.pt1 = pt1;
      prozor1.pt2 = makepoint(5, 5);
      sredina = centar(&ekran);
      {  tocka t = { 5,5 };   prozor2 = makerect(sredina, t); }
      printf("Tockata (%4.2f,%4.2f) se naogja %s pravoagolnikot
            (%4.2f, %4.2f) : (%4.2f, %4.2f)\n", sredina.x, sredina.y,
            (ptinrect(&sredina, &prozor1) ? "vo" : "nadvor od"),
            prozor1.pt1.x, prozor1.pt1.y, prozor1.pt2.x, prozor1.pt2.y);
      printf("Tockata (%4.2f,%4.2f) se naogja %s pravoagolnikot
            (%4.2f, %4.2f) : (%4.2f, %4.2f)\n", pt1.x, pt1.y,
            (ptinrect(&pt1, &prozor2) ? "vo" : "nadvor od"),
            prozor2.pt1.x, prozor2.pt1.y, prozor2.pt2.x, prozor2.pt2.y);
}
```

```
Tockata (3.00,4.00) se naogja vo
pravoagolnikot (0.00, 0.00) : (5.00, 5.00)
Tockata (0.00,0.00) se naogja nadvor od
pravoagolnikot (3.00, 4.00) : (5.00, 5.00)
```

# Битски операции

- Сите податоци се **интерно претставени како низа од бити**
  - Секој **бит** може да има една од двете вредности или **0 или 1**;
  - Низа од **8 бита** формира **бајт**.

| Оператор | Име | Опис |
|---|---|---|
| **&** | бит по бит И | Битот во резултатот се поставува на 1 само ако соодветните бити во обата операнда се 1. |
| **\|** | бит по бит ИЛИ | Битот во резултатот се поставува на 1 ако барем едниот од соодветните бити во операндите е 1. |
| **^** | бит по бит ексклузивно ИЛИ | Битот во резултатот се поставува на 1 ако точно еден од соодветните бити во операндите е 1. |
| **<<** | поместување во лево | Ги поместува битите во првиот операнд НАЛЕВО за онолку позиции колку што изнесува вториот операнд; ги пополнува битите оддесно со 0. |
| **>>** | поместување во десно | Ги поместува битите во првиот операнд НАДЕСНО за онолку позиции колку што изнесува вториот операнд; ги пополнува битите одлево со 0. |
| **~** | единичен комплемент | Сите бити 1 ги поставува на 0, а сите бити 0 на 1. |

# Битски операции

```c
#include <stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;
    printf("a = %d, b = %d\n", a, b);

    // The result is 00000001
    printf("a&b = %d\n", a & b);
    // The result is 00001101
    printf("a|b = %d\n", a | b);
    // The result is 00001100
    printf("a^b = %d\n", a ^ b);
    // The result is 11111010
    printf("~a = %d\n", a = ~a);
    // The result is 00010010
    printf("b<<1 = %d\n", b << 1);
    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);
    return 0;
}
```

```
5(00000101)
9(00001001)
```

**Output**

```
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

# Пример 2.3

```
#include <stdio.h>

void displayBits(unsigned);

int main()
{
	unsigned number1, number2, mask, setBits;
	number1 = 65535;

	mask = 1;
	printf("The result of combining the following\n");
	displayBits(number1);
	displayBits(mask);
	printf("using the bitwise AND operator & is\n");
	displayBits(number1 & mask);

	number1 = 15;
	setBits = 241;
	printf("\nThe result of combining the following\n");
	displayBits(number1);
	displayBits(setBits);
	printf("using the bitwise inclusive OR operator | is\n");
	displayBits(number1 | setBits);

	number1 = 139;
	number2 = 199;
	printf("\nThe result of combining the following\n");
```

# Пример 2.3 (продолжува)

```
        displayBits(number1);
        displayBits(number2);
        printf("using the bitwise exclusive OR operator ^ is\n");
        displayBits(number1 ^ number2);

        number1 = 21845;
        printf("\nThe one's complement of\n");
        displayBits(number1);
        printf("is\n");
        displayBits(~number1);

        return 0;
}

void displayBits(unsigned value)
{
        unsigned c = 0, displayMask = ~(~c >> 1);
        printf("%7u = ", value);

        for (c = 1; c <= sizeof(unsigned) << 3; c++) {
                putchar(value & displayMask ? '1' : '0');
                value <<= 1;

                if (c % 8 == 0) putchar(' ');
        }

        putchar('\n');
}
```

**MASK** со најзначајниот бит поставен на 1

i.e. (**10000000 00000000**)

**MASK** постојано се **AND**ува со **value**.

**MASK** има само еден поставен бит, што значи дека ако **AND** врати ненулова вредност тогаш во **value** мора овој бит да бил поставен.

**value** потоа се поместува за да се провери следниот бит

```
The result of combining the following
  65535 = 00000000 00000000 11111111 11111111
      1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
      1 = 00000000 00000000 00000000 00000001

The result of combining the following
     15 = 00000000 00000000 00000000 00001111
    241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
    255 = 00000000 00000000 00000000 11111111

The result of combining the following
    139 = 00000000 00000000 00000000 10001011
    199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
     76 = 00000000 00000000 00000000 01001100

The one's complement of
  21845 = 00000000 00000000 01010101 01010101
is
4294945450 = 11111111 11111111 10101010 10101010
```

# Битски полиња (1)

- **Составени** се само **од целобројни податоци**
- На членовите од структурата им **се специфицира должината во бити**
- Овозможува подобро искористување на меморија
- **Не е можно** да се пристапи до **адреса на член** од поле од бити

```
struct bitovi
{
     unsigned ni : 4;
     unsigned sr : 2;
     unsigned vi : 6;
} test;
```

# Битски полиња (2)

- Овие структури може да се искористат за **пакување на податоци помали од еден бајт** со што се добива покомпактна форма на податоците и **се штеди на меморија**

```
struct vreme
{
    unsigned cas;
    unsigned minuta;
    unsigned sekunda;
};

sizeof(vreme) = 12
```

bytes

```
struct pakuvanovreme
{
    unsigned cas : 4;
    unsigned minuta : 6;
    unsigned sekunda : 6;
};

sizeof(pakuvanovreme) = 4
```

bytes

# Енумерација

- **Множество од цели броеви** претставени **со симболички вредности**

- **Енумерирани константи** – слични на симболичките константи, чии вредности автоматски се поставуваат:
  - вредностите **почнуваат од 0 и се зголемуваат за 1** за секој нареден симбол;
  - вредностите може експлицитно да се доделат со операторот **=**;
  - **имињата** на симболите **мора да бидат уникатни**.

- Со дефиницијата на енумерација се воведува **нов податочен тип**. Променливите од овој тип може да ги примаат **вредностите само на дефинираните симболички константи** за тој тип.

# Пример 2.4

```c
/*Using an enumeration type */
#include <stdio.h>

enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
      JUL, AUG, SEP, OCT, NOV, DEC };

int main()
{
      int month;
      const char *monthName[] = { "", "Januari", "Fevruari",
                                  "Mart", "April", "Maj",
                                  "Juni", "Juli", "Avgust",
                                  "Septemvri", "Oktomvri",
                                  "Noemvri", "Dekemvri" };
      for (month = JAN; month <= DEC; month++)
            printf("%2d%11s\n", month, monthName[month]);

      return 0;
}
```

```
 1    Januari
 2   Fevruari
 3      Mart
 4      April
 5        Maj
 6       Juni
 7       Juli
 8     Avgust
 9  Septemvri
10   Oktomvri
11    Noemvri
12   Dekemvri
```

```c
                0       2       3       5       6       7
enum boja { bela, zolta = 2, crvena, sina = 5, zelena, crna };
                0       3       4       5       4       5
enum boja { bela, zolta = 3, crvena, sina = 5, zelena = 4, crna };
```

?