# X - Korisnički definisane strukture

- Kod rešavanja mnogih složenih problema programerima je potrebno da se omogućiti samostalno kreiranje sopstvenih struktura podataka
- Takve strukture pogodne su za grupisanje atributa nekog entiteta na jednom mestu (stvarnog ili nestvarnog)
- U C-jeziku implementirani su neki mehanizmi koji omogućavaju programerima definisanje sopstvenih, korisničkih tipova podataka i to:

1. **Struktura** - tip koji označava uređen skup promenljivih, koje mogu biti različitog tipa.
2. **Unija** - tip kojim se jednom memorijskom objektu dodeljuje skup tipova
3. **Bit polja** - specificira se veličina celobrojnih članova u bitovima
4. **Pobrojani tip** (ili enumeracija) - tip definisan skupom imenovanih celobrojnih konstanti
5. **Strukture za očitavanje vremena** – strukture i funkcije koje omogućavaju manipulaciju sa podacima o vremenu i datumu

# X - Strukture

- Struktura predstavlja skup jedne ili više promenljivih, koje mogu biti različitih tipova, a koje su radi lakše manipulacije **grupisane zajedno**.
- Strukture pomažu pri organizaciji kompleksnih podataka, posebno u velikim programima, jer one omogućavaju obradu grupe međusobno povezanih promenljivih kao jedne celine.
- Strukture su **složeni tipovi podataka** koje se, za razliku od nizova, sastoje od komponenti (segmenata) različitog tipa.
- Komponente strukture se obično nazivaju polja.
- Svako polje poseduje sopstveno jedinstveno ime i tip.
- Imena polja se grade kao i drugi identifikatori.
- Strukture omogućavaju grupisanje podataka u jednu celinu.
- U **poljima** svi elementi moraju biti istog tipa, a u **strukturama** svaki elemenat tj. polja može biti različitog tipa.
- Elementi strukture nazivaju se i članovi strukture.
- Strukture predstavljaju pogodno sredstvo za rad sa podacima koji su u međusobnoj vezi, jer se mogu grupisati pod istim imenom.

# X - Strukture

- Strukture se definišu pomoću ključne reči **struct** i u najopštijem obliku one se opisuju na sledeći način:

```
struct [oznaka]
{ tip ime_elementa1[, ime_elementa2...];
…
}[<ime_promenljive 1>[, <ime_ promenljive 2>...]];
```

- Rezervisana reč **struct** služi kao informacija kompajleru da neposredno iza nje sledi opis neke nove korisničke strukture.
- Zatim sledi neobavezni identifikator, označen sa oznaka, koji predstavlja ime strukture koje mora biti jedinstveno u program.modulu
- Ime dodeljeno strukturi se može kasnije koristiti pri deklaraciji promenljivih strukturnog tipa.
- Između velikih zagrada, deklarišu se pojedini delovi strukture.
- Elementi strukture mogu biti bilo koji tip podataka,pa i druge strukture
- Iza poslednje zagrade piše se obavezno znak '**;**'.
- Između zatvorene zagrade **}** i znaka **;** opciono se mogu navesti imena promenljivih strukturnog tipa.

# X - Strukture

**Primer:** *Sledećom definicijom strukture opisani su osnovni podaci o strukturi* ***student****: ime, broj indeksa i upisana godina studija.*

```
struct student { char *ime;
                 int indeks;
                 int godina;
               } s1,s2,s3;
```

- Svaki slog o studentu sadrži tri komponente: **ime, indeks i godina**.
- Komponenta **ime** je string, dok su komponente **indeks** i **godina** int.
- Promenljive **s1, s2, s3** se deklarišu kao promenljive koje mogu da sadrže konkretne vrednosti saglasno tipu **struct student**.
- Bez toga struktura predstavlja običnu definiciju kojom se ne rezerviše prostor u memoriji.
- Navođenjem promenljivih **s1, s2, s3**, C rezerviše prostor u memoriji za smeštanje podataka članova strukture.
- Broj memorijskih lokacija koje struktura zauzima može se dobiti pomoću **sizeof** operatora, isto kao i za jednostavne tipove podataka.
- Taj broj može biti veći od sume broja memorijskih lokacija za pojedine članove strukture.

# X - Strukture

- U bazama podataka svaki navedeni entitet odgovara imenu polja, a zajedno čine slog baze pa je zato struktura analogna pojmu sloga.

**Primer:** *Pretpostavimo da želimo koristiti strukture podataka o kupcima. Za svakog kupca potrebno je evidentirati sledeće podatke:*

***Ime -*** karakter niz od 20 znakova
***Adresa -*** karakter niz od 30 znakova
***Id -*** Identifikacijski broj, ceo broj (int)
***Dug*** - realan broj (float)
***Vrednost -*** kupljena robe u tekućem periodu, realan broj (float)

```
struct kupac {
        char ime [20];
        char adresa [30];
        int id;
        float dug;
        float vrednost;
        } kup1, kup2;
```

```
struct kupac {
char ime [20];
char adresa [30];
int id;
float dug;
float vrednost;
};
struct kupac kup1, kup2;
```

- Napomenimo da nije nužno navesti ime strukture (***kupac***)

# X - Strukture

- Strukturne promenljive mogu se deklarisati i odvojeno od strukture tj. mogu se inicijalizirati prilikom deklaracije ili kasnije u programu, kao i svaka druga promenljiva

**Primer:** inicijalizacija promenljivih pri deklarisanju**:**

```
struct kupac {
        char ime [20];
        char adresa [30];
        int id;
        float dug;
        float vrednost;
        } kup1 = {
                "Vemi-soft",
                "7 juli 15, Nis",
                "3125",
                "230.00",
                "8936.75"
               } ;
```

# X - Strukture

- Podaci za inicijalizaciju češće se prikupljaju unošenjem od strane korisnika ili učitavanjem podataka sa diska.
- Za pridruživanje vrednosti članovima strukture koristi se poseban operator tačke (.) koji ima najveću prioritetnu grupu i ima asocijativnost sleva na desno
- Tako sintaksa naredbe pridruživanja izgleda:

**ime_strukturne_varijable.ime_clana = izraz;**

**Primer:** inicijalizacija strukture u programu

```
#include <stdio.h>
#include <string.h>
main()
{  struct kupac {
   char ime[20];
   char adresa[30];
   int id;
   float dug;
   float vrednost;
} netcom;
strcpy(netcom.ime,“Vemi-soft");
strcpy(netcom.adresa,“7 juli 15, Nis");
netcom.id=3125;
netcom.dug=230.00;
netcom.vrednost=8936.75;
printf("Ime: %s\n", netcom.ime);
printf("Adresa: %s\n", netcom.adresa);
printf("Identif.broj: %d\n", netcom.id);
printf("Dug: %f\n", netcom.dug);
printf("Vrednost narucene robe: %f\n",
netcom.vrednost);
return 0;
}
```

# X - Strukture i nizovi

- Elementi strukture mogu biti nizovi.
- Možemo formirati i nizove struktura gde su elementi nizova strukture.

**Primer:** *struktura za vođenje evidencije o ocenama koje ima pojedini student može imati sledeći oblik:*

```
typedef struct  _student
{
char ime[25];
char predmet[25];
int ocena;
} studentinfo_t;
```

- Uz pretpostavku da nastavu pohađa 60 studenata, za vođenje evidencije o studentima može se definisati sledeći niz:

```
studentinfo_t student[60];
```

- Pojedinom članu strukture, koji je element niza, pristupa se na način da se tačka operator piše iza srednjih zagrada:

```
student[0].predmet = ARM;        student[0].ocena=6;
student[1].predmet = PJ1;        student[1].ocena = 8;
```

# X - Strukture kao argumenti funkcija

- Strukture se, za razliku od nizova, tretiraju kao "tipovi prve klase".
- To znači da se može dodeliti vrednost jedne strukturne promenljive drugoj, da se vrednost strukturne promenljive može prenositi kao argument f-je i da f-ja može vratiti vrednost strukturne promenljive.

**Primer:** upotreba strukturnih promenljivih u argumenatima funkcije.

```
#include <stdio.h>
#include <string.h>
typedef struct _student
{
char ime[30];
int ocena;
}studentinfo_t;
void display( studentinfo_t st );
int main()
{
studentinfo_t student[30];
int i=0;
strcpy( student[0].ime, "Marko Ilić”);
student[0].ocena = 8;
strcpy( student[1].ime,"Miloš Korać”);
student[1].ocena = 9;
strcpy( student[2].ime, “Ana Perić" );
student[2].ocena = 5;
strcpy( student[3].ime, "" );
student[3].ocena = 0;
while (student[i].ocena != 0 )
display( student[i++]);
return 0;
}
void display(studentinfo_t st)
{ printf( "Ime: %s ", st.ime );
  printf( "\tocena: %d\n", st.ocena );
}
```

Ime: Marko Ilić ocena: 8
Ime: Miloš Korać ocena: 9
Ime: Ana Perić ocena: 5

# X - Strukture kao argumenti funkcija

- U programu se prvo inicijaliziraju prva tri elementa niza koji su tipa studentinfo_t.
- Četvrtom elementu se vrednost člana ocena postavlja na vrednost nula.
- Ova nulta ocena će kasnije služiti kao oznaka elementa niza, do koga su uneseni potpuni podaci.
- Ispis se vrši pomoću funkcije **display(studentinfo_t)**.
- U prethodnom programu strukturna promenljiva se prenosi u funkciju po vrednosti.
- Ovaj način prenosa strukture u funkciju nije preporučljiv jer se za prenos po vrednosti u operativnu memoriju mora kopirati kompletan sadržaj definisane strukture.
- To zahteva veliku količinu memorije i procesorskog vremena (u ovom slučaju veličina strukture je 34 bajta) pa program sporije radi.
- Mnogo bolji način prenosa strukture u funkciju je da se koristi pokazivač na strukturu, a da se zatim u funkciji elementima strukture pristupa indirekcijom.

# X – Pokazivači na strukturne tipove

- Neka su deklarisani promenljiva i pokazivač tipa **studentinfo_t**:

  **studentinfo_t St, *pSt;**

  i neka je promenljiva **St** inicijalizovana sledećim iskazom:

  **St.ime = strcpy(“Marko Ilić");**
  **St.ocena = 8;**

- Ako se pokazivač **pSt** inicijalizuje na adresu promenljive **St**, tj.

  **pSt = &St;**

  tada se može pristupiti promenljivoj **St** i pomoću **dereferenciranog pokazivača.**

**Primer:** *ako se želi promeniti ocena na vrednost 9 i ispisati novo stanje, možemo koristiti sledeće naredbe*:

```
(*pSt).ocena = 9;
printf("Ime: %s, ocena: %d", (*pSt).ime, (*pSt).ocena)
```

- Uočite da se zbog primene tačka operatora dereferencirani pokazivač mora napisati u zagradama, jer bi inače tačka operator imao prioritet.
- Ovakav način označavanja indirekcije je dosta komplikovan pa je zato u C jeziku definisan operator →

# X - Pokazivači na strukturne tipove

```
pSt →ocena = 9;
printf("Ime: %s, ocena: %d", pSt→ime, pSt→ocena)
```

- Važi ekvivalentni zapis

**(*pStudent).ocena ⇔ pStudent →ocena**

- Operator → označava pristup elementu strukture pomoću pokazivača i može se pozvati operatorom indirekcije pokazivača strukture.

**Primer:** Prethodni program napisan korišćenjem pokazivača.

```
#include <stdio.h>
#include <stdlib.h> /* def. NULL */
typedef struct _student
{
char *ime;
int ocena;
}studentinfo_t;
void display( studentinfo_t *pSt );
int main()
{
int i=0;
studentinfo_t *p;
studentinfo_t student[30] = {
{ "Marko Ilić", 8 },
{ "Miloš Korać", 9 },
{ “Ana Perić", 5 },
{ NULL, 0}
};
p=&student[0];
while (p->ime != NULL )
display( p++);
return 0;
}
void display(studentinfo_t *pS)
{
printf( "Ime: %s ", pS->ime );
printf( "\tocena: %d\n", pS->ocena );
}
```

# X – Pokazivači na strukturne tipove

- Prvo što se treba uočiti u ovom programu je način kako je deklarisana struktura **_student** .
- U njoj sada nije rezervisan fiksni broj mesta za član **ime**, već **ime** sada predstavlja pokazivač na **char (string)**.
- Ovime se postiže ušteda memorije jer se za string **ime** rezerviše tačno onoliko mesta koliko je upisano inicijalizacijom (plus nula!).
- Pomoćnim pokazivačem **p**, koji se početno inicira na adresu nultog elementa niza **student**, pretražuje se niz sve dok se ne dođe do elementa kojem član **ime** ima vrednost NULL pokazivača.
- Dok to nije ispunjeno vrši se ispis sadržaja niza pomoću f-je **display()**, kojoj se kao argument prenosi pokazivač na tip **studentinfo_t**.
- Unutar funkcije **display()** elementima strukture se pristupa indirekcijom pokazivača strukture (→).
- Pokazivače na strukturu treba koristiti i kada funkcija vraća vrednost strukture, jer se štedi memorija i vreme izvršavanja programa.
- Prenošenje strukture po vrednosti se može tolerisati samo u slučajevima kada struktura ima malo zauzeće memorije.

# X – Polje strukture

- Ako je potrebno da operišemo sa nizom slogova u jednoj bazi podataka u C-u se koristi **polje strukture**
- Polje strukture predstavlja sredstvo za čuvanje podataka koji se zapisuju ili učitavaju sa diska

**Primer:**

```
struct kupac {
        char ime [20];
        char adresa [45];
        int id;
        float dug
        float vrednost;
        } kup [500]          /* polje od 500 struktura*/
```

- Potrebno je voditi računa prilikom definisanja ovog polja jer velika polja struktura zauzimaju i dosta memorije,
- Pojedinom članu pristupa se pomoću operatora tačke ili pokazivačem:

```
kup [34].id = 2341;
(*(kup+34)).id = 2341
```

# X-Memorijska slika strukturnih tipova

- U tabeli su date uporedne karakteristike nizova i struktura.
- Važno je uočiti da članovi struktura u memoriji nisu nužno poređani neposredno jedan iza drugog.
- Razlog tome je činjenica da je dozvoljeno da se članovi strukture smeštaju u memoriju na način koji će omogućiti najbrže izvršavanje programa (većina CPU brže pristupa parnim nego neparnim adresama)

| **Karakteristika** | **Niz** | **Struktura** |
|---|---|---|
| ***strukturalni sadržaj*** | kolekcija elemenata istog tipa | kolekcija imenovanih članova različitog tipa |
| ***pristup elementima složenog tipa*** | elementima se pristupa pomoću indeksa niza | članovima se pristupa pomoću imena |
| ***raspored elemenata u memoriji računara*** | u nizu jedan do drugog | u nizu, ali ne moraju da budu jedan do drugog |

# X - Operacije sa strukturama

- Sadržaj jedne strukturne promenljive može se pridružiti drugoj, kao i kod promenljivih osnovnih tipova:

```
struct kupac kup1,kup2;
…
kup2=kup1;
```

- Sve operacije koje se izvode nad običnim promenljivama osnovnih tipova, mogu se vršiti i sa promenljivama članovima struktura.

**Primeri:**

```
kup1.dug+=117.34;
kup1.dug=kup2.dug;
++kup1.dug;
```

- Zbog najvišeg prioriteta operatora **tačka** izraz ++**kup1.dug;** je ekvivalentan sa ++(**kup1.dug**);
- Isto tako **&kup1.dug** je isto kao **&**(**kup1.dug**).
- Adresni operator može se primeniti na čitavu strukturu ili na pojedine njene članove.

# X - Operacije sa strukturama

- Kada struktura sadrži polje kao član strukture, onda se članovima polja može pristupiti izrazom

**promenljiva.clan[izraz]**

**Primer:** *Ako je kupac promenljiva tipa* ***struct racun****, onda izrazom* ***kupac.ime[7]*** *dobijamo osmi karakter u imenu kupca.*

- Ako imamo polje struktura, onda pojedinom članu elementa polja pristupamo izrazom

**polje[izraz].clan**

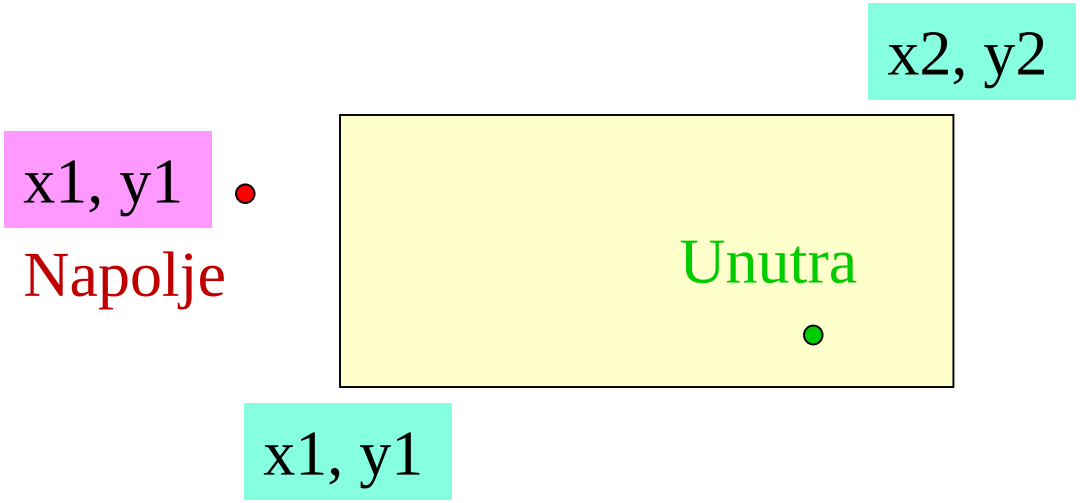
**Primer:** *Ako je promenljiva* ***kupci*** *polje tipa* ***struct racun****, onda broju računa osmog kupca pristupamo izrazom* ***kupci[7].broj_racuna***

- U zadnja dva primera dolazi do izražaja asocijativnost, jer su srednje zagrade i operator tačka istog prioriteta.
- Njihova asocijativnost je sleva na desno, tako da se operandi grupišu prvo oko levog operatora.
- Sa članovima strukture postupa se potpuno isto kao i sa običnim promenljivama istog tipa.
- Isto važi i za složene članove strukture, polja, strukture itd.

# X - Primer korišćenja strukture

**Primer:**

*Napisati glavni program i funkciju u programskom jeziku C koja proverava da li se zadata tačka nalazi unutar zadatog pravougaonika. Funkcija treba da vrati vrednost različitu od nule ukoliko se tačka nalazi unutar zadatog pravougaonika.*



# X - Primer korišćenja strukture

```
#include <stdio.h>
struct tacka{ int x;
              int y;
            };
struct Pravougaonik {
        struct tacka dole_levo;
        struct tacka gore_desno;
                    };
int testPtIn(struct Pravougaonik,struct
tacka*);
void main ()
{
  struct tacka tt;
  int x1, y1, x2, y2;
  struct Pravougaonik testRect;
  printf("Unesi dve tacke za
pravougaonik\n");
scanf("%d,%d,%d,%d", &x1, &y1, &x2,
&y2);
testRect.dole_levo.x = x1;
testRect.dole_levo.y = y1;
testRect.gore_desno.x = x2;
testRect.gore_desno.y = y2;
printf("Unesi tacku koja se testira\n");
scanf("%d, %d", &x1, &y1);
tt.x = x1;
tt.y = y1;
if(testPtIn(testRect,&tt))
 printf("Tacka je unutar
pravougaonika");
else
  printf("Tacka je van pravougaonika");
}

/* funkcija za testiranje */
int testPtIn(struct Pravougaonik Pr,
struct tacka *t)
{
return t->x > Pr.dole_levo.x && t->x <
Pr.gore_desno.x && t->y >
Pr.dole_levo.y && t->y <
Pr.gore_desno.y;
}
```

# X – Unija

- Sa ključnom reči **union** definiše se korisnički tip podataka pomoću koga se nekom memorijskom objektu može pristupiti sa različitim tipskim pristupom.
- Deklaracija unije slična je deklaraciji strukture, ali je njeno značenje sasvim drugačije.

**Primer:**

```
union skalar
{ char c;
  int i;
  double d;
};
```

- Ako se unijom **skalar** deklariše objekat **obj: union skalar obj;** tom se objektu može pristupiti na više načina, jer **obj** može sadržati različite vrednosti i to tipa char, int ili double.
- Dozvoljeno je pisati: **obj.c = 'A'; obj.i = 1237;** ili **obj.d = 457.87**
- Sve ove vrednosti se upisuju u **istu memorijsku lokaciju**.
- Veličina zauzeća memorije je određena elementom unije koji zauzima **najveći prostor memorije**.

# X – Unija

- Unija se često koristi unutar neke strukture kojoj posebni element služi za označavanje tipa vrednosti koji trenutno sadrži unija.

**Primer:**

```
#define char_type 0
#define int_type 1
#define double_type 2
struct variant
{  int var_type;                /* oznaka tipa vrednosti unije skalar */
   union skalar var;
};
struct variant obj;
/* pored vrednosti, treba upisati i tip vrednosti*/
obj.var.c = 'A'; obj.var_type = char_type;
obj.var.i =1237; obj.var_type = int_type;
obj.var.d =457.87; obj.var_type = double_type;
.....
/* unija se koristi tako da je prethodno potrebno proveri tip vrednosti */
switch(obj.val_type)
{
case char_type: printf("%c", obj.var.c); break;
case int_type: printf("%d", obj.var.i); break;
case double_type: printf("%f", obj.var.d); break;
```

# X – Bit polja

- Unutar strukture ili unije može se specificirati veličina celobrojnih članova u bitovima.
- To se izvodi tako da se iza člana strukture navede **znak dvotačke i broj bitova** koliko taj član zauzima.

**Primer:**

```
struct bitfield1
{
int i3b : 3;
unsigned int i1b : 1;
signed int i7b : 7;
};
```

- Ovom deklaracijom definisano je da u strukturi **bitfield1** član **i3b** je ceo broj od 3-bita, član **i1b** je 1-bitni a član **i7b** je 7-bitni ceo broj.
- Ovakve strukture se koriste u cilju uštede memorijskog prostora, posebno u slučaju kad se većina članova tretira kao logička vrednost.
- Članovima strukture se pristupa kao da su normalni celi brojevi, a kompajler vodi računa o tome da se maskiraju nepostojeći bitovi:

```
struct bitfield1  a, b;    a.i3b=3;    b.i7b=65; .
```

# X – Bit polja

- Upravljanje ovim strukturama je potpuno pod kontrolom kompajlera.
- Ovde se ne mogu koristiti pokazivači na članove strukture niti nizovi bitnih polja jer oni nisu direktno adresibilni.
- Kada se želi kontrolisati kako se slažu bit-polja unutar jedne reči, na raspolaganju su dva mehanizma:
  1. da se umetnu bezimeni članovi koji će predstavljati "prazne bitove".
  2. da veličina bezimenog polja može biti 0.
- To je poruka kompajleru da se tu završi smeštanje članova strukture u jednu reč, i da se od tog mesta članovi strukture smeštaju u novu reč.

**Primer:**

```
struct bitfield2
{ int i3b : 3;
  unsigned int i1b : 1;
  signed int i7b : 7;
  int : 2;
  int i2b: 2;
  int : 0;
  int i4b : 4, i5b : 5;
};
```

Opisuje strukturu koja se pakuje u dve reči:
1.prva reč sadrži redom 3bit., 1bit., i 7 bitna polja, (**i3b, i1b, i7b**) zatim 2 bitnu prazninu, te 2 bitno polje (**i2b**).
2. druga reč sadrži 4 bitna i 5 bitna polja (**i4b** i **i5b**)

# X - Nabrojivi tipovi (enum)

- Promenljive ovog tipa mogu da imaju samo konačan skup vrednosti
- Deklaracija promenljivih nabrojivog tipa se sastoji od ključne reči **enum**, imena enumerisanog tipa, eventualno liste dozvoljenih vrednosti i liste enumerisanih promenljivih koje se deklarišu.
- Služi definisanju integralnog celobrojnog tipa kome se skup vrednosti označava simboličkim (imenovanim) konstantama u sledećoj notaciji:

**enum ime_tipa { lista_definicije_ konstanti };**

**Primer:**

```
enum dani_t {Nedelja,Ponedeljak,Utorak,Sreda,Cetvrtak,Petak,Subota};
```

- Ovde se definiše korisnički nabrojivi tip **dani_t** kod koga se vrednost označava simboličkim konstantama: **Nedelja, Ponedeljak, Utorak, Sreda, Cetvrtak, Petak, Subota.**
- Pri kompajliranju programa ovim se konstantama dodeljuju numeričke vrednosti prema pravilu da prvo ime u listi ima numeričku vrednost 0, drugo ime ima vrednost 1, treće ime ima vrednost 2 itd.

# X – Nabrojivi tipovi (enum)

- Pomoću nabrojivog tipa definišu se promenljive, prema pravilu:

**deklaracija_promenljive: enum ime_tipa lista_promenljivih;**

**Primer:**

```
enum dani_t danas, sutra;
.......
danas = Sreda;
sutra = Cetvrtak; ........
```

- Vrednost neke ili svih simboličkih konstanti mogu se inicijalizirati već pri samoj deklaraciji nabrojivog tipa.

**Primer:**

```
enum karte_t {AS = 1, JACK = 11, DAMA, KRALJ};
```

- Neinicijalizirane konstante imaju vrednost za jedan veću od vrednosti prethodne konstante, tako DAMA ima vrednost **12**, a KRALJ ima vrednost **13**.
- Nabrojive tipove ne treba smatrati posebno "čvrstim" tipovima jer ih se može tretirati ravnopravno sa celobrojnim tipom pa je dozvoljeno:

```
enum karte_t  c=10; c++;  /*c postaje JACK(11)*/
```

# X - Strukture za očitavanje vremena

- U datoteci <**time.h**> definisano je nekoliko funkcija i struktura imena **tm**, za očitanje i manipulisanje podacima o vremenu i datumima.
- Do sada je za očitavanje vremena korišćena funkcija

  **time_t time(time_t *tp);**

  koja je vraćala vrednost **time_t** tipa, tj. broj koji predstavlja trenutno vreme (obično je to broj sekundi od 1.1.1970.).
- Parametar **tp**, ako nije NULL, takođe prihvata trenutno vreme u ***tp**.
- Da bi se olakšalo pretvartanje ovog vremena u stvarno vreme i datum, definisana je struktura **tm** sa sledećim članovima:

```
struct tm          /* opisuje vreme i datum */
{ int tm_sec;      /* sekunde 0..61 */
int tm_min;        /* minute 0..59 */
int tm_hour;       /* sat 0..23 */
int tm_mday;       /* dan 1..31 */
int tm_mon;        /* mesec 0..11 */
int tm_year;       /* broj godina nakon 1900 */
int tm_wday;       /* dan u sedmici 0..6 */
int tm_yday;       /* dan u godini 0..365 */
int tm_isdst;      /* da li je dan promene sata 0..1 */
};
```

# X - Strukture za očitavanje vremena

- Ako dan promene sata nije implementiran tada **tm_isdst** ima negativnu vrednost.
- Broj sekundi može biti veći od 59 u slučaju prestupnog vremena.
- **Meseci** su kodirani tako da 0 označava januar, 1 februar itd.
- **Dani u sedmici** su kodirani tako da 0 označava nedelju, 1 ponedeljak...
- Stvarna godina se dobija tako da se članu **tm_year** doda vrednost 1900

**Funkcije localtime, gmtime**

- Pretvaranje vremena iz formata **time_t** u strukturu tipa **tm** vrši se funkcijom **localtime()**, kada se želi dobiti lokalno vreme, ili f-jom **gmtime()** za dobijanje univerzalnog vremena u nultom meridijanu.

```
struct tm *localtime(const time_t *t);
struct tm *gmtime(const time_t *t);
```

- Obe funkcije primaju adresu promenljive koja sadrži vreme u formatu **time_t**, a vraćaju pokazivač na statičku strukturu tipa **tm**

**Funkcije ctime, asctime**

- Ako se želi dobiti zapis vremena u obliku stringa, mogu se koristiti f-je

```
char *ctime(const time_t *t);   char *asctime(const struct tm  *tp);
```

# X - Strukture za očitavanje vremena

- Funkcija **ctime()** za argument koristi adresu promenljive koja sadrži vreme u formatu **time_t**, a funkcija **asctime()** za argument koristi pokazivač na strukturu **tm**.
- Obe funkcije vraćaju pokazivač statičkog stringa koji sadrži zapis vremena u standardnom formatu.

**Primer:**

```
time_t t = time(NULL);
char *s = ctime(&t);
puts(s);
```

ova sekvenca naredbi generiše ispis: **Sat May 11 14:21:20 2002**

- Ovde je poziv **ctime(&t)** ekvivalentan pozivu **asctime(localtime(&t))**
- Standardna verzija je prilagođena američkim standardima a ako se želi napisati vreme u formatu **11.05.2002 14:21** tada se može koristiti:

```
time_t t = time(NULL);
struct tm *p = localtime(&t);
printf("%.2d.%.2d.%.2d %2d:%.2d\n",
p->tm_mday, p->tm_mon + 1,
p->tm_year +1900,
p->tm_hour, p->tm_min);
```

# X - Strukture za očitavanje vremena

## Funkcija strftime

- Funkcija ***strftime()*** se koristi za formatirani ispis vremena.
- Format se zadaje kao kod ***printf()*** funkcije.
- Prototip funkcije ***strftime()*** glasi:

  **size_t strftime(char *buf, size_t bufsize, const char *fmt, const struct tm *tp);**

- Prvi argument je string **str** u koji se vrši formatirani zapis.
- Drugi argument (**bufsize**) ograničava broj znakova stringa.
- Treći parametar je string u kojem se zapisuje format ispisa nizom specifikatora oblika **%x** (kao kod printf() funkcije).
- Poslednji argument je pokazivač strukture **tm**.
- Funkcija vraća broj znakova u stringu ili 0 ako nije moguće generisati formatirani string.

# X - Strukture za očitavanje vremena

Specifikatori formata su:

**%a** skrač.od tri slova za ime dana u sedmici (eng. Sun, Mon, Tue,..)
**%A** puno ime dana u sedmici (eng...)
**%b** skrač. od tri slova za ime meseca (eng. Jan, Feb, Mar,...)
**%B** puno ime meseca (eng....)
**%c** kompletni zapis vremena i datuma
**%d** dan u mesecu (1..31)
**%H** sat u formatu (1..24)
**%I** sat u formatu (1..12)
**%j** dan u godini (1..365)
**%m** mesec u godini (1..12)
**%M** minute
**%p AM/PM** (eng.) string koji označava jutro ili popodne
**%S** sekunde
**%U** broj za sedmicu u godini (1..52),1 određen prvom nedeljom
**%w** broj za dan u sedmici,0-nedelja
**%W** broj za sedmicu u godini (1..52), 1 određen prvim ponedeljkom
**%x** kompletni zapis datuma
**%X** kompletni zapis vremena
**%y** zadnja dva broja godine
**%Y** godina u formatu sa 4 broja
**%Z** ime vremenske zone (ako postoji )
**%%** znak %

# X - Strukture za očitavanje vremena

**Primer:**

```
#include <stdio.h>
#include <time.h>
int main()
{
time_t vreme = time(NULL);
struct tm *ptr;
char datum_str[20];
/* ispisuje datum i vreme u standardnom
   formatu */
puts(ctime(&vreme));
/* ispisuje datum i vreme pomoću
   strftime funkcije */
strftime(datum_str,
   sizeof(datum_str),
"%d.%m.%y %H:%M\n",
   localtime(&vreme));
puts(datum_str);
/* ispisuje datum i vreme u proizvoljnom
   formatu */
ptr = localtime(&vreme);
printf("%.2d.%.2d.%.2d %2d:%.2d\n",
ptr->tm_mday, ptr->tm_mon+1, ptr-
   >tm_year +1900,
ptr->tm_hour, ptr->tm_min);
return 0;
}
```

Dobija se ispis:
Mon May 13 20:13:06 2002
13.05.02 20:13
13.05.2002 20:13

# X - Strukture za očitavanje vremena

**Funkcija mktime** **time_t mktime(struct tm *tp)**

- Funkcija **mktime()** pretvara zapis iz strukture **tm** u **time_t** format.
- Korisna je u tzv. kalendarskim proračunima kada je potrebno dodati nekom datumu **n** dana, tada se može upisati datum u **tm** strukturu, povećati član **tm_mday** za **n**, zatim pozivom mktime() se dobije time_t vrednost koja odgovara novom datumu.

**Funkcija difftime** **double difftime(time_t t1, time_t t2)**

- Funkcija **difftime()** vraća realnu vrednost koja je jednaka razlici vremena t1 i t2 u sekundama.

**Funkcija clock** **clock_t clock(void);**

- Funkcija **clock()** služi za preciznije merenje vremena
- Ona vraća vrednost CPU merača vremena, koji startuje na početku programa, u jedinicama koje su znatno manje od sekunde: nekoliko ms
- Koliko je tih jedinica u jednoj sekundi određeno je konstantom **CLOCKS_PER_SEC.**
- Izraz **(double)clock()/CLOCKS_PER_SEC** daje vrednost koja je jednaka vremenu (u sekundama) od startovanja programa.

# X - Strukture za očitavanje vremena

**Primer:** *Program treba da ispita vremensku rezoluciju funkcije clock(), tj. minimalno vreme koje se njome može meriti. Pomoću funkcije clock() meri se koliko je potrebno vremena za izvršenje sinusne f-je.*

```
#include <stdio.h>
#include <math.h>
#include <time.h>
int main()
{
double start, stop;
double n ;
double rezolucija;
start=(double)clock()/CLOCKS_PER_SEC;
do {
stop=(double)clock()/CLOCKS_PER_SEC;
     }
while (stop == start);
rezolucija = stop-start;
printf("Rezolucija CLOCK-a je %g
    sekundi\n" , rezolucija);
start
    =(double)clock()/CLOCKS_PER_SEC;
stop = start + 10*rezolucija;
do {
n += 1.0;
sin(n);
}
while (stop >
    (double)clock()/CLOCKS_PER_SEC);
printf("Funkcija sin se izvršava %g
    sekundi\n" , 10*rezolucija/n);
return 0;
}
```

Rezolucija CLOCK-a je 0.015 sekundi
Funkcija sin se izvrsava u 2.3543e-007 sekundi

# Hvala na pažnji !!!

Pitanja

? ? ?