Strukture

Slide 1

Deklaracija strukture:

- Polja grupiraju veći broj podataka istog tipa.
- Strukture služe grupiranju više podataka različitih tipova.

```
struct ime {
      tip_1 ime_1;
      tip_2 ime_2;
      ....   ....
      tip_n ime_n;
};
```

Primjer:

```
struct tocka {
      int x;
      int y;
};
```

Slide 2

Programiranje (C) Ak. god. 2003/04

Definicija varijabli tipa strukture:

```
mem_klasa struct ime  varijabla1, varijabla2,...;
```

Primjer:

```
struct tocka {
      int x;
      int y;
};
struct tocka  p1,p2;
```

ili

```
struct tocka {
      int x;
      int y;
} p1,p2;
```

Slide 3

Inicijalizacija strukture:

```
mem_klasa struct ime  varijabla={v_1, v_2,..., v_n};
```

Konstante `v_i` pridružuju se odgovarajućim članovima strukture.

Primjer:

```
struct racun {
      int broj_racuna;
      char ime[80];
      float stanje;
};

struct racun kupac={1234,"Dalibor M.", -23456.00};
```

Slide 4

M. Jurak pred9.tex

Polje struktura:

```
struct racun kupci[]={34, "Ivo R.",   456.00,
                      35, "Goran S.", 234.00,
                      112,"Dalibor M.",00.00};
```

Struktura definirana pomoću strukture:

```
    struct pravokutnik {
          struct tocka pt1;
          struct tocka pt2;
   };
```

Slide 5

Rad sa strukturama:

Primjer:

```
    struct tocka {
          int x;      // prvi clan strukture
          int y;      // drugi clan strukture
   };

   struct tocka  ishodiste;
```

- `ishodiste` je varijabla tipa `struct tocka`.
- `ishodiste.x` je prva komponenta (x-komponenta) varijable `ishodiste`;
- `ishodiste.y` je druga komponenta (y-komponenta) varijable `ishodiste`.

Slide 6

Primjer:

```
      struct racun {
            int broj_racuna;
            char ime[80];
            float stanje;
     } kupac={1234,"Dalibor M.", 23456.00};
```

Tada je

```
kupac.broj_racuna=1234,
kupac.ime="Dalibor M.",
kupac.stanje=23456.00.
```

Slide 7

Općenito:

- Ako je `var` varijabla tipa strukture koja sadrži član `memb`, onda je

  `var.memb`

  član `memb` u strukturi `var`.
- Operator točka (.) separira ime varijable i ime člana strukture. Spada u najvišu prioritetnu grupu i ima asocijativnost slijeva na desno.

`++varijabla.clan` je ekvivalentno s `++(varijabla.clan)`;

`&varijabla.clan` je ekvivalentno s `&(varijabla.clan)`.

Slide 8

- Kada struktura sadrži polje kao član strukture, onda se elementi polja dosežu izrazom

```
varijabla.clan[izraz]
```

- Ako imamo polje struktura onda za pojedini element polja član srukture dosežemo izrazom

```
polje[izraz].clan
```

(Ovdje je bitna asocijativnost jer su svi operatori istog prioriteta.)

Slide 9

Primjer 1:

```
 struct racun {
       int broj_racuna;
       char ime[80];
       float stanje;
} kupac={1234,"Dalibor M.", -23456.00};
......
if(kupac.ime[0] == 'D') ...
```

Primjer 2:

```
 struct tocka {
       int x;
       int y;
} vrhovi[1024];
......
if(vrhovi[17].x == vrhovi[17].y) ...
```

Slide 10

Operacije nad strukturom kao cjelinom:

- Pridruživanje;
- Uzimanje adrese, primjena `sizeof` operatora;
- Struktura može biti argument funkcije;
- Funkcija može vratiti strukturu.

Primjer:

Slide 11

```
 struct tocka {
         int x;
         int y;
 } t,ishodiste={0,0};
 struct tocka suma(struct tocka p1, struct tocka p2) {
      p1.x += p2.x;
      p1.y += p2.y;
      return p1;
}
  ......
t=ishodiste;   // t i ishodiste moraju biti istog tipa
printf("%d\n",sizeof(t));
```

Slide 12

Programiranje (C) Ak. god. 2003/04

Strukture i pokazivači

- Pokazivač na strukturu definira se kao i pokazivač na osnovne tipove varijabli.

```
 struct tocka {
       int x;
       int y;
} p1, *pp1;
 ......
pp1= &p1;
(*pp1).x=13;
(*pp1).y=27;
*pp1.x = 13; // GRESKA:
             //  *pp1.x isto sto i *(pp1.x)
```

Slide 13

Operator ->

Ako je `ptvar` pokazivač na strukturu, a `clan` jedan član strukture, onda je izraz

```
ptvar->clan
```

ekvivalentan s

```
(*ptvar).clan
```

Primjer:

```
struct tocka  p1, *pp1=&p1;
pp1->x=13;
pp1->y=27;
```

Slide 14

Složeni izrazi:

```
struct pravokutnik {
      struct tocka pt1;
      struct tocka pt2;
}  r, *pr=&r;
```

Sljedeći su izrazi ekvivalentni:

```
 r.pt1.x        /* operatori . i ->     */
 pr->pt1.x      /* imaju isti prioritet */
(r.pt1).x       /* i asocijativnost     */
(pr->pt1).x     /*  L->D                */
```

Slide 15

Složeni izrazi:

Kakav će ispis dati sljedeći program?

```
struct  {
   int n;
   char *ch;
} t[10], *pt=&t[0];   // Globalne varijable

int main(void) {
  int i;
  char tmp[10];

  for(i=0;i<10;i++){ /* inicijalizacija polja */
     t[i].n=i;
     t[i].ch=(char *)malloc(10);
     sprintf(tmp,"%c",'a'+i);
     strcat(tmp,"++");
```

Slide 16

M. Jurak pred9.tex

```
    strcpy(t[i].ch,tmp);
  }
  printf("%s \n",(++pt)->ch);
  printf("%s \n",pt++->ch);
  /* pt++->ch je ekvivalentno s (pt++)->ch   */
  /* zbog razlicite asocijatvnosti operatora */
  printf("%c \n",*pt->ch++);
  printf("%c \n",*pt++->ch);
  printf("%s \n", pt->ch);
  return 0;
}
```

Slide 17

Ispis programa:

```
b++
b++
c
+
d++
```

Slide 18

Programiranje (C) Ak. god. 2003/04

Samoreferentne strukture

- Za implementaciju tipova podataka kao što su vezane liste i stabla koristimo samoreferentne strukture.
- Samoreferentna strukture ima jedan ili više članova pokazivača na strukturu istog tipa.
- Pokazivači na strukturu istog tipa služe za povezivanje elemenata u listu, stablo itd.

Primjer:

```
struct element {
   char ime[64];
   struct element *next;
};
```

Slide 19

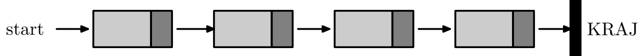
Primjer: vezana lista.

```
#include <stdio.h>     /* Poceta programa */
#include <string.h>
#include <stdlib.h>
                  /* oznaka kraja liste */
#define KRAJ (struct ime *) 0
/* deklaracija samoreferentne strukture */
struct ime {
    char *p_ime;
    struct ime *next;
};

/* pokazivac na prvi element liste
                   (globalna varijabla) */
struct ime *start=KRAJ;
```

Slide 20

M. Jurak pred9.tex

Operacije nad vezanom listom;



Implementiramo sljedeće operacije:

- Dodavanje novog elementa na kraj liste: `void unos(void);`
- Ispis liste: `void ispis(void);`
- Pretraživanje liste: `void trazi(void);`
- Brisanje elementa iz liste: `void brisi(void);`

Slide 21

```
void unos(void) {
  struct ime  *zadnji, *novi;
  char line[128];

  novi = (struct ime *) malloc(sizeof(struct ime));
  novi->next=KRAJ;
  printf("Unesite ime > "); scanf(" %[^\n]",line);
  novi->p_ime=(char *) malloc(strlen(line)+1);
  strcpy(novi->p_ime,line);
  if(start==KRAJ) start=novi;
  else {    /* pronadji kraj liste */
    for(zadnji=start; zadnji->next != KRAJ;
                             zadnji=zadnji->next) ;
    zadnji->next=novi;
  }
}
```

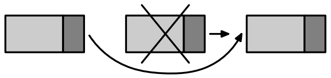
Slide 22


```
void ispis(void) {
    struct ime *element;
    int i=1;

    if(start == KRAJ ) {
       printf("Lista je prazna.\n");
       return;
    }
    printf("Ispis liste \n");
    for(element=start; element != KRAJ;
                   element=element->next) {
       printf("%d. %s\n",i,element->p_ime); i++;
    }
}
```

Slide 23

```
void trazi(void) {
  struct ime *test;
  char line[128];
  int i;

  printf("Nalazenje imena u listi.\n");
  printf("Unesite ime "); scanf(" %[^\n]",line);
  i=1;
  for(test=start; test != KRAJ; test=test->next){
     if( !strcmp(test->p_ime,line) ) {
        printf("Podatak je %d. u listi\n",i);
        return; }
     i++;
    }
    printf("Podatak nije u listi.\n");
}
```

Slide 24



```
void brisi(void) {
    struct ime *test, *tmp;
    char line[128];
    int i;

    printf("Brisanje imena iz liste.\n");
    /*  Da li je lista prazna ?  */
    if(start == KRAJ){
      printf("Lista je prazna.\n");
      return;
    }
```

Slide 25

```
    /* Unos imena koje treba obrisati.  */
    printf("Unesite ime ");
    scanf(" %[^\n]",line);

   /* Testiranje prvog elementa liste.  */

    if( !strcmp(start->p_ime,line) ) {
       printf("Podatak je 1. u listi\n");
       tmp=start;
       start=start->next;
       free(tmp->p_ime);
       free(tmp);
       return;
     }

  /* Testiranje preostalih elemenata liste  */
```

Slide 26

Programiranje (C) Ak. god. 2003/04

```
  i=1;
  for(test=start; test != KRAJ; test=test->next) {
    i++;
    if(!(tmp=test->next)) break;
    if( !strcmp(tmp->p_ime,line) ) {
      printf("Brisemo podatak br. %d. u listi\n",i);
      test->next=test->next->next;
      free(tmp->p_ime);
      free(tmp);
      return;
    }
  }
  printf("Podatak nije u listi.\n");
  return;
}
```

Slide 27

Izbor operacija putem menu-a:

```
int menu(void) {
    int izbor;
    printf("\n\n\n");
    printf("\n    OPERACIJE : ");
    printf("\n    =========");
    printf("\n 1. Unos novog elemeta ");
    printf("\n 2. Ispis liste ");
    printf("\n 3. Pretrazivanje liste ");
    printf("\n 4. Brisanje iz liste ");
    printf("\n 5. Izlaz ");
    printf("\n          izbor = ");

    scanf("%d",&izbor);
    return izbor;
}
```

Slide 28

M. Jurak pred9.tex

```
int main(void) {  /*  GLAVNI PROGRAM   */
    int izbor;
    do {
       switch(izbor=menu()) {
         case 1: unos(); break;
         case 2: ispis(); break;
         case 3: trazi(); break;
         case 4: brisi(); break;
         case 5: break;
         default:
             printf("\n Pogresan izbor.");
       }
    } while(izbor != 5);
    return 0;
}
```

Slide 29

typedef

Pomoću ključne riječi `typedef` postojećim tipovima podataka dajemo nova imena (ne kreiramo nove tipove podataka!).

Primjer:

```
typedef double Masa;
```

Sada `Masa` postaje sinonim za `double`.

```
Masa m1,m2,*pm1;
Masa elementi[10];
.........
```

Općenito `typedef` ima oblik:

```
typedef tip_podatka  novo_ime_za_tip_podatka;
```

Slide 30

typedef, složenije supstitucije:

`typedef` i pokazivači:

```
typedef  double *Pdouble;
```

`Pdouble` postaje tip pokazivač na `double`:

```
Pdouble px;    /* =  double *px;  */

void f(Pdouble,Pdouble);
               /* = void f(double *,double *); */

px= (Pdouble) malloc(100*sizeof(Pdouble));
```

Slide 31

`typedef` i strukture:

Umjesto

```
struct tocka {
      int x;
      int y;
};
........
struct tocka p1, *pp1;
```

možemo pisati

```
typedef struct  {
      int x;
      int y;
} tocka;
...........
tocka p1, *pp1;
```

Slide 32

`typedef` i samoreferentne strukture:

- Kad se `typedef` primijenjuje na samoreferentnu strukturu potrebno je uvesti jednu prethodnu deklaraciju tipa kako bi se izbjegla cirkularna deklaracija.

```
typedef struct element *Pelement;

typedef struct element {
   char ime[64];
   Pelement next;
} Element;
.........
Element root;
```

Slide 33

`typedef` i deklaracije funkcija:

Primjer:

```
typedef int (*PF)(char *, char *);
```

`PF` postaje ime za pokazivač na funkciju koja uzima dva pokazivača na `char` i vraća `int`.

Umjesto

```
void f(double x, int (*g)(char *, char *))
{   ......
```

možemo pisati

```
void f(double x, PF g)
{ ......
```

Slide 34

Unije

Sintaksa:

```
union  ime {
      tip_1 ime_1;
      tip_2 ime_2;
      ....   ....
      tip_n ime_n;
 };
```

- Unija kao i struktura sadrži članove različitih tipova.
- Kod unije svi članovi dijele istu memorijsku lokaciju.
- Memorijska lokacija bit će dovoljno velika da u nju stane najširi tip podatka iz unije.

Deklaracija varijabla tipa `ime`:

```
union  ime  x,y;
```

Slide 35

Primjer:

```
union  pod {
      int  i;
      float x;
 } u, *pu;
```

- `u.i` i `pu->i` su varijable tipa `int`;
- `u.x` i `pu->x` su varijable tipa `float`.
- Osnovna svrha unije je ušteda memorijskog prostora. Na istoj memorijskoj lokaciji možemo čuvati varijable različitih tipova. Pri tome moramo paziti da uniji pristupamo konsistentno.

```
/* ispisivanje decimalnog zapisa broja tipa float */
   u.x=0.234375;
   printf("0.234375 binarno = %x\n",u.i);
```

Slide 36