Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésautomatikai Tanszék

Járműfedélzeti rendszerek II.

4. előadás

Dr. Bécsi Tamás

6. Struktúrák

- A struktúra egy vagy több, esetleg különböző típusú változó együttese, amelyet a kényelmes kezelhetőség céljából önálló névvel látunk el. Néhány nyelvben az így értelmezett struktúrát rekordnak nevezik.
- A struktúra bevezetése segíti az összetett adathalmazok szervezését, ami különösen nagy programok esetén előnyös, mivel lehetővé teszi, hogy az egymással kapcsolatban lévő változók egy csoportját egyetlen egységként kezeljük, szemben az egyedi adatkezeléssel.

6.1. Alapfogalmak

- A struktúra deklarációját a `struct` kulcsszó vezeti be, amelyet kapcsos zárójelek között a deklarációk listája követ. A struct kulcsszót opcionálisan egy név, az ún. *struktúracímke* követheti (mint a példánkban a pont). Ez a címke vagy név azonosítja a struktúrát és a későbbiekben egy rövidítésként használható a kapcsos zárójelek közötti deklarációs lista helyett.

6.1. Alapfogalmak

- A struktúrában felsorolt változóneveket a struktúra *tagjainak* nevezzük. Egy struktúra címkéje (neve), ill. egy tagjának a neve és egy közönséges (tehát nem struktúratag) változó neve lehet azonos, mivel a programkörnyezet alapján egyértelműen megkülönböztethetők. Továbbá ugyanaz a tagnév előfordulhat különböző struktúrákban, bár célszerű azonos neveket csak egymással szoros kapcsolatban lévő adatokhoz használni.

6.1. Alapfogalmak
Példa: címkézett struktúra


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

- A kifejezés egy típust definiál, ahol a pont a struktúracímke, x és y a tagok. Ha a struktúra címkézett volt, akkor a címke a későbbi definíciókban a struktúra konkrét előfordulása helyett használható:

```
struct pont pt;
```

- Amely egy pont típusú változót definiál, amely akár közvetlenül inicializálható a tagok értékeinek felsorolásával:

```
struct pont pt={ 320 , 200 };
```

6.1. Alapfogalmak
Példa: nem címkézett struktúra

- Egy struct deklaráció egy típust is definiál. A jobb oldali, záró kapcsos zárójel után következhet a változók listája, hasonlóan az alapadattípusok megadásához. Így pl. a

```
struct {...} x, y, z;
```

- Közvetlenül hoz létre x,y,z struktúra változókat, és foglal le helyet számukra a tárterületen.

6.1. Alapfogalmak

- Egy kifejezésben az adott struktúra egy tagjára úgy hivatkozhatunk, hogy

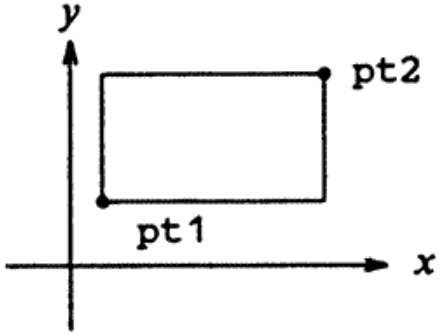
```
struktúra-név.tag
```

- A pont struktúratag operátor összekapcsolja a struktúra és a tag nevét. A pt pont koordinátáit pl. úgy írathatjuk ki, hogy

```
printf("%d, %d", pt.x, pt.y);
```

6.1. Alapfogalmak

- A struktúrák egymásba ágyazhatók. Például egy téglalap az átlója két végén lévő pontpárral írható le:




```
struct tegla {
        struct pont pt1;
        struct pont pt2; };
```

- Ennek alapján a tegla struktúra két pont struktúrából áll. Ha az abra struktúrát úgy deklaráljuk, hogy

```
struct tegla abra;
```

- akkor az

```
abra.pt1.x
```

- hivatkozás az abra pt1 tagjának x koordinátáját jelenti.

6.2. Struktúrák és függvények

- A struktúrák esetén **megengedett művelet** a struktúra **másolása** vagy **értékadása**, ill. a struktúra **címéhez való hozzáférés az & operátorral** és a struktúra **tagjaihoz való hozzáférés**. Ezek a műveletek a struktúrát egy egységként kezelik, és a másolás vagy értékadás magában **foglalja a struktúrák függvény-argumentumkénti átadását**, ill. a **struktúra típusú függvényvisszatérés** lehetőségét is. **Struktúrák egy egységként nem hasonlíthatók össze**.

6.2. Struktúrák és függvények
Példa 1


```
/* makepoint: egy pont struktúrát csinál az x és y
  komponensekből*/
  struct pont makepoint (int x, int y)
  {
   struct pont temp;
   temp.x = x;
   temp.y = y;
   return temp;
  }
…
struct tegla abra;
abra.pt1 = makepoint(0, 0);
```

6.2. Struktúrák és függvények
Példa 2


```
/* addpoint: két pont összeadása */
struct pont addpoint(struct pont p1, struct pont p2)
{
   p1.x += p2.x;
   p1.y += p2.y;
   return p1;
}
```

- Ennek a függvénynek a két argumentuma és a visszatérési értéke egyaránt struktúra. A függvényben átmeneti változó bevezetése nélkül, közvetlenül a p1 komponenst növeltük, hogy kihangsúlyozzuk, a struktúra típusú paraméterek éppen úgy érték szerint adódnak át, mint bármely más változó.

6.2. Struktúrák és függvények Struktúra mutatók

- Ha nagy struktúrát kell átadnunk egy függvénynek, akkor sokkal hatékonyabb, ha a struktúra mutatóját adjuk át és nem pedig a teljes struktúrát másoljuk át. A struktúra mutatójának deklarációja:

  ```
  struct pont *pp;
  ```

  Ez egy struct pont típusú struktúrát kijelölő mutatót hoz létre. Ha pp egy pont struktúrát címez, akkor `*pp` maga a struktúra, és `(*pp).x`, ill. `(*pp).y` pedig a struktúra tagjai. A pp értékét felhasználva pl. azt írhatjuk, hogy

```
struct pont kezdet, *pp;
pp = &kezdet;
printf("kezdet: (%d, %d)\n", (*pp).x, (*pp).y);
```

6.2. Struktúrák és függvények Struktúra mutatók

- A zárójelre a `(*pp).x` kifejezésben szükség van, mert a . struktúratag operátor precedenciája nagyobb, mint a * operátoré. A `*pp.x` kifejezés azt jelentené, mint a `*(pp.x)`, ami viszont szintaktikailag hibás, mivel jelen esetben x nem mutató. A struktúrák mutatóit gyakran használjuk rövidített jelölési formában. Ha p egy struktúra mutatója, akkor a `p-> struktúratag` kifejezés közvetlenül a struktúra megadott tagját címzi.

6.2. Struktúrák és függvények Struktúra mutatók

A `.` és `->` struktúraoperátorok a függvény argumentumát tartalmazó `()` kerek és az indexet tartalmazó `[]` szögletes zárójelekkel együtt a legmagasabb precedenciájú operátorok, így rendkívül szorosan kötnek. Például, ha adott a

```
struct { int hossz; char *str; } *p;
```

deklaráció, akkor a `++p->hossz` kifejezés a hossz változót inkrementálja és nem a p-t, mivel a precedencia-szabályoknak és a végrehajtási sorrendnek megfelelő alapértelmezés `++(p->hossz)`. A kötés zárójelezéssel változtatható meg, pl. a

`(++p)->hossz` a hossz változóhoz való hozzáférés előtt inkrementálja a pértékét, a `(p++)->hossz` pedig a hozzáférés után inkrementál.

6.2. Struktúrák és függvények Struktúra mutatók

- A fordítás idején hatásos `sizeof` unáris operátor bármilyen C nyelvű objektum méretének meghatározására használható. A

```
sizeof objektum
```

és

```
sizeof (típusnév)
```

kifejezések egy egész számot adnak eredményül, ami a megadott objektum vagy adattípus bájtokban mért mérete.

6.2. Struktúra mutatók
Példa


```
int main(int argc, char *argv[])
{
  struct tpont p1={1,2}, *p, *pp1;
  p=(struct tpont*)malloc(sizeof(struct tpont));
  *p=p1;
  pp1=&p1;
  p->x--;
  p->y--;
  p1.x++;
  printf("p1 : %d %d\n",p1.x,p1.y);
  printf("pp1: %d %d\n",pp1->x,pp1->y);
  printf("p : %d %d\n",(*p).x,p->y);
  system("PAUSE");   return 0;
}
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa definíció


```
struct node {
    int x;
    struct node *next;
};

struct node *root,*actual;
root=(struct node*)
            malloc(sizeof (struct node));
    root->x=0;
    root->next=NULL;
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa add


```
struct node* addnode(struct node *root,
int value )
{
        root->next=(struct
node*)malloc(sizeof (struct node));
        root->next->x=value;
        root->next->next=NULL;
        return root->next;
}
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa list


```
void listnodes(struct node *root)
  {
        while (root!=NULL)
        {
            printf("%d ",root->x);
            root=root->next;
        }
        printf("\n");
  }
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa search


```
struct node* searchnode(struct node
  *root,int value)
  {
      while (root!=NULL)
      {
        if (root->x==value) return root;
           else root=root->next;
      }
      return NULL;
  }
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa insert


```
int insertnode(struct node *root,int position, int value)
   {
   struct node *savenext;
   while (--position)
      {
        root=root->next;
        if (!root) return 1;
      }
   savenext=root->next;
   root=addnode(root,value);
   root->next=savenext;
   return 0;
   }
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa delete


```
int deletenode(struct node *root,int position)
   {
      struct node *temp;
      while (--position)
      {
        root=root->next;
        if (!root->next) return 1;
      }
      temp=root->next;
      root->next=root->next->next;
      free(temp);
      return 0;
   }
```

6.5. Önhivatkozó struktúrák
Láncolt lista példa main


```
struct node *root,*actual;
int i;
root=(struct node*)malloc(sizeof (struct node));
root->x=0;
root->next=NULL;
actual=root;
for (i=1;i<10;i++)
   actual=addnode(actual,i);
listnodes(root); //0 1 2 3 4 5 6 7 8 9
if (actual=searchnode(root,5)) actual->x=0;
listnodes(root); //0 1 2 3 4 0 6 7 8 9
printf("%d\n",insertnode(root,3,0)); //0
printf("%d\n",insertnode(root,12,0)); //1
listnodes(root); //0 1 2 0 3 4 0 6 7 8 9
deletenode(root,1);
listnodes(root); //0 2 0 3 4 0 6 7 8 9
```

6.8. Unionok

- Az *union* egy olyan változó, amely különböző időpontokban különböző típusú és méretű objektumokat tartalmazhat úgy, hogy a fordítóprogram ügyel az objektumok méretére és tárbeli elhelyezésére vonatkozó előírások betartására. Az unionok alkalmazásával lehetővé válik, hogy azonos tárterületen különböző fajta adatokkal dolgozzunk, anélkül, hogy a programba géptől függő információkat kellene beépíteni.

6.8. Union példa
Egy CAN üzenet


```
typedef union{
  unsigned char BYTES[8];
  struct {
    unsigned short EAD;
    unsigned char ND1 : 2;
    unsigned char CONTROL_MODE : 2;
    unsigned char PRIORITY : 2;
    unsigned char EBI_MODE : 2;
    unsigned char URGENCY;
    unsigned char ND2;
    unsigned char ND3;
    unsigned char ND4;
    unsigned char CHECKSUM :4;
    unsigned char MESSCOUNT : 4;
  } message;
} xbr_message_t;
```

6.8. Union példa
Egy CAN üzenet


```
xbr_message_t xbrMessage;
xbrMessage.BYTES[0]=0;
xbrMessage.BYTES[1]=0;
printf("\n%x %x\n",xbrMessage.BYTES[0],
                   xbrMessage.BYTES[1]);
//0 0
xbrMessage.message.EAD=0x1234;
printf("\n%x %x\n",xbrMessage.BYTES[0],
                   xbrMessage.BYTES[1]);
//34 12
```

Vége

Köszönöm a figyelmet!