Tartalomjegyzék

Operátorok, új típusok definiálása, stringek

Kiegészítések az eddigi anyagrészekhez

- C-ben a függvényeknek egyedi névvel kell rendelkezniük. Általában a többi C-re épül? nyelv (C++, C#, Java) már megengedi hogy ugyanolyan névvel de különböz? *szignatúrával* létezzenek függvények (vagyis elég ha a visszatérési érték típusa vagy a paraméterlista típusai különböznek).

- Egy tömb neve használható közvetlenül mutatóként, pl itt:

```
int egyik[20];
    egyik[0] = 11;
    printf("e[0]: %d\n", egyik[0]);
    *egyik = 12;
    printf("e[0]: %d\n", egyik[0]);
    *(egyik+3) = 24;
    printf("e[3]: %d\n", egyik[3]);
```

Ezt írja ki:

```
 e[0]: 11
 e[0]: 12
 e[3]: 24
```

Operátorok precedenciája

A precedencia határozza meg azt a sorrendet, ahogy egy kifejezésben használt operátorok kiértékel?dnek. Zárójelezéssel persze megváltoztathatjuk a kiértékelési sorrendet.

(Itt szokták megemlíteni az operátoroknak az *asszociativitás*-át is, vagyis hogy jobbról balra vagy balról jobbra értékel?dnek-e ki az operandusai. A lent linkelt táblázatokban meg lehet ezt nézni, és jó tudni err?l, de nem fogjuk visszakérdezni.)

Amiket tudni kell (persze azt is hogy az operátorok mit csinálnak):

- Egyszer? precedencia táblázat

Asszociativitást is tartalmazó táblázatok:

- Precedencia és asszociativitás táblázat, példák
- Teljes operátor táblázat(wikipedia)

Egy egyszer? példa:

```
int a = (1+4+2)*3;  /* 21 */
    int b = 1+4+2*3;    /* 11 */
```

Két tömbös-mutatós példa:

```
int * alma[20];      // alma egy 20 elem? tömb ami mutatókat (mégpedig int típusra mutatókat) tart
  int (* korte)[20];   // korte egy mutató ami egy egészeket tartalmazó tömbre mutat
```

A C nyelv megengedi azt is, hogy egy mutató függvényre mutasson. (Nem fogjuk ezt sem visszakérdezni, sem használni ebben a félévben, de jó tudni hogy van ilyen, és legalább felismerés szintjén érdemes ismerni.)

```
int (*func_p)();    // func_p egy mutató egy olyan függvényre ami int-et ad vissza
```

További példák

```
   int i;          // integer variable 'i'
   int *p;         // pointer 'p' to an integer
   int a[];        // array 'a' of integers
   int f();        // function 'f' with return value of type integer
   int **pp;       // pointer 'pp' to a pointer to an integer
   int (*pa)[];    // pointer 'pa' to an array of integer
   int (*pf)();    // pointer 'pf' to a function with returnvalue integer
   int *ap[];      // array 'ap' of pointers to an integer
   int *fp();      // function 'fp' which returns a pointer to an integer
   int (**ppa)[];  // pointer 'ppa' to a pointer to an array of integers
   int (**ppf)();  // pointer 'ppf' to a pointer to a function with return value of type integer
   int *(*pap)[];  // pointer 'pap' to an array of pointers to an integer
   int *(*pfp)();  // pointer 'pfp' to function with return value of type pointer to an integer
   int **app[];    // array of pointers 'app' that point to pointers to integer values
   int (*apa[])[]; // array of pointers 'apa' to arrays of integers
   int (*apf[])(); // array of pointers 'apf' to functions with return values of type integer
```

(forrás: http://en.wikibooks.org/wiki/C_Programming/Pointers_and_arrays)

Típusnév hozzárendelés

A *typedef* kulcsszóval valójában nem definiálhatunk új típust, csak egy új nevet egy már létez? típushoz. Így használható:

typedef <*létez?_típus*> <*új_azonosító*>

Példa:

```
typedef long* long_mutato;  /* ezzel elneveztük "long_mutato"-nak a "long*" típust*/
    long szam1 = 42;
    long* szam1_ptr = &szam1;
    long_mutato szam2_ptr = szam1_ptr;
    *szam2_ptr = 23;
```

A felhasználó által definiálható típusok

- Felsorolás (*enum*): olyan új típus, aminek véges és meghatározott értékkészlete van.
- Struktúra (*struct*): több változó (már létez? típusokkal) összefogása egy típussá.
- Unió (*union*): hasonló a stuktúrához, de itt egyszerre csak egy elemnek lehet beállított értéke, ugyanis az unió bels? változói egy közös memóriaterületet használnak.

Felsorolás típus

Az *enum* definiálásakor megadjuk az értékkészletet, és az ilyen típusú változó csak ezek közül vehet fel értéket. (A háttérben 0-tól kezdve egészek rendel?dnek az értékekhez, így m?ködik az egyenl?ség-vizsgálat és a kisebb-nagyobb összehasonlítás is.)

```
enum evszak {
   TAVASZ,
   NYAR,
   OSZ,
   TEL
} e1, e2;

enum evszak elsoevszak = TAVASZ;
enum evszak masikevszak = TEL;
if (elsoevszak > masikevszak) {
    printf("elsoevszak(%d) kesobb van mint a masik(%d)\n", elsoevszak, masikevszak);
} else {
    printf("elsoevszak(%d) hamarabb van mint a masik(%d)\n", elsoevszak, masikevszak);
}
```

Amit a fenti kódrészlet kiír:

```
  elsoevszak(0) hamarabb van mint a masik(3)
```

Struktúrák

Logikailag egy egységet alkotó, akár különböz? típusú adatokból álló összetett adattípus.

Struktúra deklaráció általános alakja (a szögletes zárójel azt jelenti hogy az a rész elhagyható): struct [<struktúra címke>] {

```
   <struktúra tag deklarációk>
```

} [<struktúra változó azonosítók>];

Példa 1:

```
struct vektor_int_2d {
    char id[20]; // a vektor neve egykaraktertömbben
    int xpos;    // x koordinata
    int ypos;    // y koordinata
};
```

Példa 2:

```
typedef int honap;
struct datum {
    int ev;
    honap ho;
```

```
    short nap;
} d1, d2;             // itt már fel is vettünk két dátum típusú változót
struct datum d3, d4;
```

Példa név(címke) nélküli struktúra definícióra (így kés?bb nem tudunk ilyen változókat létrehozni, csak itt a pontosvessz? el?tt):

```
struct {
    int ev;
    short ho;
    short nap;
} datum1, datum2;
```

Példa 4: maga a struktúradefiniálás címke(név) megadása nélkül történik, de rögtön új nevet adunk *typedef*-fel a létrejöv? új típusnak. Így nem kell mindig a "http://wiki.math.bme.hustruct"http://wiki.math.bme.hu szót kiírni az ilyen típusú változók deklarálásakor. Ebben a példában már használjuk is az adattagokat (a pont operátorral kérhetjük el a struktúra egy adattagját, ill a "http://wiki.math.bme.hu->"http://wiki.math.bme.hu operátorral ha mutatónk van a struktúrára).

```
typedef struct {
    int ev;
    honap ho;
    short nap;
} datum_t;
datum_t d1, *d2_p;  /* d1 egy dátum (datum_t típusú változó), d2_p pedig mutató egy dátumra */

d1.ev = 1992;
d1.ho = 10;
d1.nap = 2;
d2_p = &d1;      /* d2_p most már d1-re mutat */
d2_p->ev = 1990;   /* használhatjuk az adattagok átírására */
```

Struktúrákat megéri használni, mert:

- Struktúra típusú változó értéke frissíthet? egyetlen értékadással.
- Függvény paramétere és visszatérési értéke is lehet struktúra típusú.

Egy hosszabb vektoros példa:

```
#include <stdio.h>

typedef struct{
    double x,y;
} vektor;

void v_kiir(vektor a) {
    printf("(%5.2lf; %5.2lf)", a.x, a.y);
}

vektor eredo(vektor a, vektor b){
    vektor c;
    c.x = a.x+b.x;
    c.y = a.y+b.y;
    return c;
}

vektor nyujt(vektor a, double k){
    a.x = k*a.x;
    a.y = k*a.y;
```

```
    return a;
}

int main() {
    vektor v1 = {2.1, 4.3};
    vektor v2 = {1.8,-1.5};
    vektor v3;
    printf("v1="); v_kiir(v1); printf("\n");
    printf("v2="); v_kiir(v2); printf("\n");
    printf("v1+v2="); v_kiir(eredo(v1,v2)); printf("\n");
    v3 = nyujt(v1, 2.2);
    printf("2.2*v1="); v_kiir(v3); printf("\n");
    return 0;
}
```

Szövegek ábrázolása a C nyelvben

A string típus szöveges tartalmat tárolhat, valójában egy karaktertömb (char[]) aminek a végét egy speciális karakter ('\0') jelzi.

Különbség egy karakter és egy egybet?s string között:

- Karakter konstans: 'a' (1 karakteren tárolódik)
- String: "http://wiki.math.bme.hua"http://wiki.math.bme.hu (2 karakteren tárolódik, van egy lezáró '\0' karakter)

Néhány különleges karakter:

Különleges karakterek

Alakja	Jelentése
'\' '	aposztróf
'\"http://wiki.math.bme.hu'	idéz?jel
'\?'	kérd?jel
'\\'	rep
'\a'	hangjelzés
'\n'	újsor
'\t'	tabulátor
'\b'	visszatörlés
'\0'	a 0-ás kódú karakter, a stringek lezárására használjuk

- Egy szöveg ("http://wiki.math.bme.hualma"http://wiki.math.bme.hu) karakter tömbként:

```
char ct [5] = {'a','l','m','a', '\0'};
```

- Helyette egyszer?bb, és ugyanazt jelenti:

```
char ct [5] ="alma";
```

- S?t, még ezek is (nem kell a méretet megadni, úgyis kiderül):

```
char ct [] ="alma";
char ct [] ="al" "ma";  // az el?feldolgozó egyesíti
```

A string lényegében egy karaktertömb, vagy egy mutató egy olyan karaktertömbre ami '\0'-val zárul.

Ha stringekkel dolgozunk, szükségünk lehet erre a sorra:

```
#include <string.h>
```

Nem része a C szabványnak, de tartalmazza pl. a stringeket összehasonlító függvényt:

```
int strcmp (char *op1, char *op2) {
    while (*op1 && *op1 == *op2) {
        op1++; op2++;
    }
    return *op1-*op2;
}
```

Használata:

```
char * str1 = "alma";
    char * str2 = "aafdsg";
    int melyik = strcmp(str1, str2);
    printf("strcmp: %d\n", melyik);  // 1, 0 vagy -1 lesz (itt 1 mert az els? a nagyobb)
```

További függvények:

- *size_t strlen (char *str)*: string hossza (a lezáró \0 nélkül)
- *char *strchr (char *str, char ch)* : megtalált karakter címe a stringben
- *char *strcpy (char *dest, char *source)* : string átmásolása
- *char *strmove (char *dest, char *source)* : string átmásolása (kicsit okosabban)

Ellen?rz? kérdések

- Állítsd sorrendbe a precedenciájuk szerint az operátorokat, azzal kezdve ami a leger?sebben köt! + , % = >
- Az alábbi négy sor közül melyik deklarál mutatókat tartalmazó tömböt?

```
int **p1;
    int (*p2)[];
    int (*p3)();
    int *p4[];
```

- Defniálj egy struktúrát "http://wiki.math.bme.hu3d_vektor"http://wiki.math.bme.hu címkével, ami három darab double típusú értékb?l áll!
- Mire valók a "http://wiki.math.bme.hu."http://wiki.math.bme.hu és a "http://wiki.math.bme.hu->"http://wiki.math.bme.hu operátorok?
- Milyen karakter zárja le a stringeket?

Források és további olvasnivalók

- http://www.hit.bme.hu/~vitez/Progalap1/2011osz/Ea/ea05.pdf
- http://www.hit.bme.hu/~vitez/Progalap1/2011osz/Ea/ea06.pdf
- http://en.wikipedia.org/wiki/C_string_handling
- http://torusz.hu/prog/kr-c/files/06.html
- http://www.exforsys.com/tutorials/c-language/handling-of-character-strings-in-c.html