Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedés- és Járműirányítási Tanszék

Programozás C- és Matlab nyelven
C programozás kurzus
BMEKOKAM603
Előfeldolgozó rendszer
Tömbök

Dr. Bécsi Tamás

4. Előadás

A ?: operátor

- *Nézzük meg a következő kifejezést:*

```
if (a>b)
        z=a;
            else
        z=b;
```

Ez felírható egyszerűbb alakban a ?: operátor segítségével:

```
z=(a>b)?a:b;
```

formában

4.11. A C előfeldolgozó rendszer

- A fordítás első lépése a C esetében a különböző nyelvi kiterjesztések feldolgozása:
  - másik állomány tartalmának beépítése a forrásprogramba (`#include`)
  - szimbólum-helyettesítés (`#define`)
  - argumentumot tartalmazó makrók (`#define`)
  - feltételes fordítási direktívák (ezzel nem foglalkozunk)

4.11.1. Állományok beépítése

- *Formája:*

```
#include <állománynév>
```
, vagy
```
#include "állománynév"
```

- Idézőjelek esetén a keresés a forráskönyvtárban kezdődik; az állomány hiánya, vagy <> esetén a rendszer által definiált helyről történik az állomány bemásolása.

B4. Matematikai függvények: a <math.h> header

sin(x)	az x argumentum szinusza;
cos(x)	az x argumentum koszinusza;
tan (x)	az x argumentum tangense;
asin(x)	az x argumentum arkuszszinusza, az értékkészlet a [-π/2, π/2] tartomány, x E [-1, 1];
acos(x)	az x argumentum arkuszkoszinusza, az értékkészlet a [0, π] tartomány, x E [-1, 1];
atan(x)	az x argumentum arkusztangense, az értékkészlet a [-π/2, π/2] tartomány;
atan2(y, x)	az y/x érték arkusztangense, az értékkészlet a [-π, π] tartomány;
sinh(x)	az x argumentum szinusz hiperbolikusa;
cosh(x)	az x argumentum koszinusz hiperbolikusa;
tanh(x)	az x argumentum tangens hiperbolikusa;
exp(x)	az e^x exponenciális függvény;
log(x)	az x argumentum természetes alapú logaritmusa (ln(x))), x>0;
log10(x)	az x argumentum tízes alapú logaritmusa (lg(x))), x>0;

B4. Matematikai függvények: a <math.h> header

`fabs(x)`	az x argumentum abszolút értéke (|x|);
`ldexp(x, n)`	az $x*2^n$ függvény értéke;
`frexp(x, int *exp)`	a függvény az x argumentum értékét az [1/2, 1) intervallumba eső normált törtrésszé alakítja és ezzel az értékkel tér vissza. A 2 hatványaként értelmezett kitevő a *exp című változóba tárolódik. Ha x=0, akkor az eredmény törtrésze és kitevője egyaránt nulla lesz; **result = frexp (8, &n); -> result=0.5, n=4**
`modf(x, double *ip)`	az x argumentum eredeti x előjelével azonos előjelű egész- és törtrészre bontása. Az egészrész az *ip című változóban tárolódik, a függvény visszatérési értéke a törtrész lesz; **fractpart = modf (3.14 , &intpart); -> fractpart=0.14,intpart=3**
`fmod(x, y)`	az x/y lebegőpontos osztás lebegőpontos maradéka, ami ugyanolyan előjelű mint x. Ha y=0, akor az eredmény a gépi megvalósítástól függ. **fmod (5.3,2)->1.3**
`pow(x, y)`	az x^y alakú hatványfüggvény, értelmezési tartomány hiba lép fel, ha x=0 és y<0, vagy ha x<0 és y értéke nem egész szám;
`sqrt(x)`	az x argumentum négyzetgyöke, x>0;
`ceil(x)`	az x argumentumnál nem kisebb legkisebb egész szám, `double` típusra konvertálva;
`floor(x)`	az x argumentumnál nem nagyobb legnagyobb egész szám, `double` típusra konvertálva;

4.11.2. Makróhelyettesítés

Egy definíció általánosan

```
#define név helyettesítő szöveg
```

alakú, és hatására a makróhelyettesítés egyik legegyszerűbb formája indul el: a névvel megadott kulcsszó minden előfordulási helyére beíródik a helyettesítő szöveg.

4.11.2. Makróhelyettesítés

A `#define` utasításban szereplő névre ugyanazok a szabályok érvényesek, mint a változók neveire, a helyettesítő szöveg pedig tetszőleges lehet.

A helyettesítés csak **az önálló kulcsszavakra** (nevekre) vonatkozik és nem terjed ki az idézőjelek közötti karaktersorozatokra sem. Például hiába egy definiált név az, hogy YES, nem jön létre a helyettesítés a printf("YES") utasításban vagy a YESMAN szövegben.

4.11.2. Makróhelyettesítés, példa


```
#define EOS '\0'
#define TRUE 1
#define boole int
int main(void)
{
  boole vege=0;
  char s[]="hello";
  int i=0;
  do
  {
     if (s[++i]==EOS)
vege=TRUE;
  }
  while (!vege);
  printf("%d",i);
  getchar();
}
```

```
int main(void)
{
  int vege=0;
  char s[]="hello";
  int i=0;
  do
  {
     if (s[++i]=='\0')
vege=1;
  }
  while (!vege);
  printf("%d",i);
  getchar();
}
```

4.11.2. Makróhelyettesítés példa, veszélyek


```
#define abs(x) ((x) < 0 ? (-(x)) : (x))
#define min(a, b) ((a) < (b) ? (a) : (b))

int i=-5; int j=-6;
    printf("%d",min(abs(i),abs(j)));
```

Veszélyek:

a fentiek szerint definiált min() makró kétszer alkalmazza a beírt kifejezést:

```
min(i++,j++)
```

helyettesítő szövege:

```
((i++) < (j++) ? (i++) : (j++))
```

azaz a változók értéke kétszer inkrementálódik.

4.11.2. Makróhelyettesítés hibás használat

További példa hibás használatra:

```
#define MARGO 2
#define SZELESSEG 8
#define MAGASSAG 10
#define TELJESSZEL SZELESSEG+MARGO
#define TERULET TELJESSZEL*MAGASSAG
```

Hibás eredmény Zárójelezéssel kiküszöbölhető

```
#define square(x) x*x
square(i+1);
```

helyettesítő szövege `i+1*i+1`

4.11.2. Makróhelyettesítés hibás használat kiküszöbölése

További példa hibás használatra:

```
#define MARGO 2
#define SZELESSEG 8
#define MAGASSAG 10
#define TELJESSZEL (SZELESSEG+MARGO)
#define TERULET TELJESSZEL*MAGASSAG
```

```
#define square(x) (x)*(x)
square(i+1);
```

helyettesítő szövege **(i+1)*(i+1)**

5.3. Tömbök (Statikus tömbök)

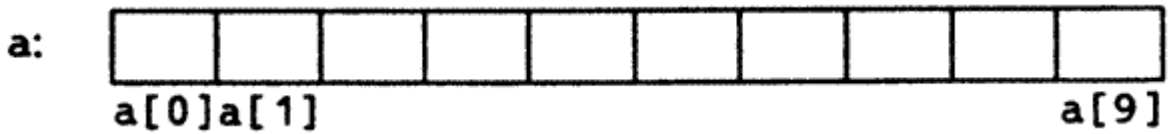
A tömbök segítségével azonos elemekből álló adathalmazt tárolhatunk a memóriában és műveleteket hajthatunk végre rajta. Általános alakjuk:

típus tömbnév[elemszam];

```
Például:  int a[10];
```

deklaráció egy tízelemű tömböt jelöl ki, azaz tíz egymást követő, a[0]...a[9] névvel ellátott objektumot.

Az `a[i]` jelölés az `a` tömb i-edik elemére hivatkozik.



Tömbök inicializálása


```
int t[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
// teljesen feltöltve

int t[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
// ugyanaz, mint az előző

int t[10] = {1, 2, 3, 4};
// csak 4 elemet inicializáltunk, de a tömb 10 méretű
```

Több dimenziós tömbök


```
int t[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

int t[3][4] = {0, 1, 2, 3,
               4, 5, 6, 7,
               8, 9,10,11};

int t[3][4] = {{0, 1, 2, 3},
               {4, 5, 6, 7},
               {8, 9,10,11}};
```

Karakter tömbök


```
char s[200] = "Hello";
char s[200] = {'H', 'e', 'l', 'l', 'o', '\0'};
// Az előzővel egyenértékő.
// Figyeljük meg, hogy az utolsó karakter után még van egy 0
// kódú karakter, az zárja a stringet.

char s[] = "Hello";
// a tömb mérete 5+1=6 lesz a lezáró 0 miatt
```

Gyakorló példa

- Lottószelvény ellenőrzött kitöltése

Köszönöm a figyelmet