Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedésautomatikai Tanszék

Járműfedélzeti rendszerek II.

2. előadás

Dr. Bécsi Tamás

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

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.

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)**

4.11.3. Feltételes fordítás

Az előfeldolgozási folyamat közben kiértékelt feltételes utasításokkal lehetőségünk van magának az előfeldolgozásnak feltételektől függő vezérlésére is. Ennek hatására szelektíven iktathatunk be sorokat a programba, a fordítás (előfeldolgozás) során kiértékelt feltételek értékétől függően.

4.11.3. Feltételes fordítás

Az `#if` sor hatására az utána álló állandó egész kifejezés (amely nem tartalmaz sizeof, enum vagy kényszerített típusú [cast] állandókat) kiértékelődik, és ha ennek értéke nem nulla, akkor a következő sorok az első `#endif, #elif` vagy `#else` utasításig beépülnek a programba. (Az előfeldolgozó `#elif` utasítása hasonló a C else if utasításához.) A `defined(név)` kifejezés értéke 1 az `#if` utasításban, ha a *név* már definiálva volt, és 0 különben.

4.11.3. Feltételes fordítás

Például, ha biztosak szeretnénk lenni abban, hogy a hdr.h állomány tartalma csak egyszer, de egyszer legalább beépül a programba, akkor a hdr.h állomány beépítési helyének környezetébe az alábbi utasításokat kell elhelyezni:

```
#if !defined(HDR)
#define HDR
/* ide épül be a hdr.h tartalma */
#endif
```

4.11.3. Feltételes fordítás

Az `#ifdef` és `#ifndef` sorok speciális vizsgálatot végeznek: azt ellenőrzik, hogy az adott név definiált-e vagy sem. Ezek felhasználásával az előző példát úgy is írhattuk volna hogy

```
#ifndef HDR
#define HDR
/* ide épül be a hdr.h tartalma */
#endif
```

3.5. Ciklusszervezés
while utasítás


```
while (kifejezés)
 utasítás;
```

Amíg a kifejezés igaz (nem 0) addig futtatja az utasítást.

`break;` *utasítás kilép a ciklusból*

`continue;` *utasítás a következő ellenőrzésre ugrik*

3.5. while példa


```
char c=0;
     while(++c<10)
        {


        printf("%d ",c);
        }
```

Eredmény: 1 2 3 4 5 6 7 8 9

3.5. while példa continue


```
char c=0;
     while(++c<10)
        {
        if (c==3) continue;

        printf("%d ",c);
        }
```

Eredmény: 1 2 4 5 6 7 8 9

3.5. while példa break


```
char c=0;
     while(++c<10)
        {
        if (c==3) continue;
        if (c==6) break;
        printf("%d ",c);
        }
```

Eredmény: 1 2 4 5

3.5. Ciklsszervezés
for utasítás


```
for (kif1; kif2; kif3)
     utasítás
```

Ilyen formájában ekvivalens:

```
kif1;
while (kif2){
    utasítás
    kif3;
}
```

Bármelyik kifejezés elhagyható: `for (;;)`

3.5. for példa


```
for(c='a';c<'f';c++)
{

        printf("%c ",c);
 }
        printf("Ertek ciklus utan: %c ",c);
```

Eredmény: a b c d e Ertek ciklus utan: f

3.5. for példa continue


```
for(c='a';c<'f';c++)
{
        if (c=='b') continue;

        printf("%c ",c);
 }
        printf("Ertek ciklus utan: %c ",c);
```

Eredmény: a c d e Ertek ciklus utan: f

3.5. for példa
break


```
for(c='a';c<'f';c++)
{
        if (c=='b') continue;
        if (c=='e') break;
        printf("%c ",c);
 }
        printf("Ertek ciklus utan: %c ",c);
```

Eredmény: a c d Ertek ciklus utan: e

3.5. for példa
több elemű kifejezés (char[] reverse)


```
char tmp,s[]="abcde";
int i,j;
for (i=0,j=4;i<j;i++,j--)
    {
        tmp=s[i]; s[i]=s[j]; s[j]=tmp;
    }
printf("%s",s);
```

Output: **edcba**

3.6. Ciklusszervezés
do-while utasítás


```
do
    utasítás
while (kifejezés);
```

Hátultesztelés (amíg a kifejezés igaz), ellentétben a pascal repeat-until ciklusával.

3.6. do-while példa


```
int i=1;
 do
 {
     printf("%d",++i);
 } while(i<4);
```

Eredmény: 234

3.6. do-while példa


```
int i=1;
 do
 {
     printf("%d",i++);
 } while(i<4);
```

Eredmény: 123

3.8. goto utasítás és címkék

Példa:

```
for (i=0;i<10;i++)
 {
     printf("%d",i);
     if (i==5) goto vege;
 }
   vege: printf("vege");
```

4. Függvények

Forma:

```
visszatérési-típus függvénynév(argumentum deklarációk)
{
    deklarációk és utasítások
}
```

Függvény visszatérés:

return kifejezés;

4. Függvények példa


```
int fakto(int i)
{
    int lfakt=1;
    while (i>1)
    {
        lfakt*=i--;
    }
    return lfakt;
}
```

4. Függvények
Rekurzív hívás (megemlítés)


```
int fakt (int i)
{
    return (i<2)?1:i*fakt(i-1);
}
```

Vége

Köszönöm a figyelmet!