Budapesti Műszaki és Gazdaságtudományi Egyetem

Közlekedésautomatikai Tanszék

Járműfedélzeti rendszerek II.

1. előadás

Dr. Aradi Szilárd

A tárgy órái

- Előadás hetente on-line Dr. Aradi Szilárd
  - C elmélet, hálózatok
  - Ajánlott irodalom
    - Dennis Ritchie: A C programozási nyelv
- Gyakorlat hetente csüt. 12:15 Fehér Árpád
  - C gyakorlat Atmel AVR

Követelmények

- 1 Zh (1 Pótzh)
- Házi feladat
- Szóbeli vizsga

C nyelv

- Strukturált (C++ objektumorientált)
- Szabványos: minden processzor platformra van külön fordító program, maga a kód hordozható
- Hatékony: a fordító assembly-re fordít
- Beágyazott rendszerekben nem szabványos bővítéseket használnak a perifériák, regiszterek és a speciális aritmetika kezelésre.

C fordító

- Előfeldolgozó (pre-processor)
  - Kiszedi a felesleges karaktereket
  - Elvégzi a #-tel kezdődő előfeldolgozói direktívákat
- Fordító (compiler)
  - Előállítja az assembly kódot a C kódból
- Assembler
  - Előállítja gépi kódú utasításokat: tárgykód (object code)
  - A külső hivatkozások még nincsenek feloldva
    - Könyvtári függvények, saját függvények másik állományban, külső változók
- Összefűző (linker)
  - Feloldja a külső hivatkozáskat
  - Előállítja a futtatható állományt (natív gépi kód)
  - Beágyazott rendszerek esetén előállítja a letölthető állományt (Intel HEX)
- Atmel AVR fordító: AVR-GCC

Programstruktúra

#include-ok

#define-ok

deklarációk:

- konstansok
- globális változók
- külső változók
- függvények

int main(void){}

függvény definíciók

```
#include <avr/io.h>
#define TRUE 1
const int j=12;
int x = 1, z[10];
extern int y;

void kiir(int a[],int n);

int main(void){
z[0]=1; z[1]=2; z[2]=3;
kiir(z,3);
}

void kiir(int a[],int n){
  int i;
  for (i=0;i<n;i++)
      printf("%d ",a[i]);
}
```

Azonosítók

- A C programnyelv Case Sensitive
- Azonosító tartalmazhat:
  - angol abc betűi (’a’..’z’ , ’A’..’Z’)
  - számok (’0’..’9’)
  - ’_’ karakter
- Nem kezdődhet számmal

Típusok

A C nyelv viszonylag kevés alapvető adattípust használ:

Típus	Leírás	AVR-GCC méret
char	egyetlen bájt, a gépi karakterkészlet egy elemét tárolja	1
int	egész szám, mérete általában a befogadó számítógép egészek ábrázolásához használt mérete	2
short	rövid egész	2
long	hosszú egész	4
float	egyszeres pontosságú lebegőpontos szám	4
double	kétszeres pontosságú lebegőpontos szám	4 vagy 8

Emellett léteznek minősítők:

unsigned	Előjel nélküli
signed	előjeles
short	Rövid
long	Hosszú

Állandók

Állandók	
1234	int állandó
1234L	long állandó
1234UL	unsigned long állandó
0x1f2	hexa állandó
0x1f2UL	hexa unsigned long
1234.5	double állandó
1234.5f	float állandó
’c’	char állandó
”szoveg”	char[] (string) állandó

Escape Szekvenciák	
\a	figyelmeztető jelzés (bell, csengő)
\b	visszalépés (backspace)
\f	lapdobás (formfeed)
\n	új sor (new line)
\r	kocsi vissza (carriage return)
\t	vízszintes tabulátor (horizontal tab, HTAB)
\v	függőleges tabulátor (vertical tab, VTAB)
\\	fordított törtvonal (backlash)
\?	kérdőjel
\'	aposztróf
\"	idézőjel
\ooo	oktális szám
\xhh	hexadecimális szám

Változó deklarációk

Deklarációk:

```
int i;
float f,g;
char c;
```

Inicializálás:

```
char c='a';
char szoveg[]="szöveg";
const int j=12;
```

Aritmetikai operátotok

A C nyelv kétoperandusú aritmetikai operátorai a +, -, * és /, valamint a % modulus operátor. Az egészek osztásakor a törtrészt a rendszer levágja, ezért van szükség a % modulus operátorra. Az x % y kifejezés az x/y egészosztás (egész) maradékát adja, és értéke nulla, ha x osztható y-nal.

A % operátor nem alkalmazható float és double típusú adatokra. Negatív operandusok esetén az egészosztás eredmény és a modulus előjele gépfüggő.
Az osztás eredménye lehet a legnagyobb egész szám, amely kisebb, mint az algebrai hányados, vagy a legkisebb egész szám, amely nagyobb, mint az algebrai hányados.

Az egyoperandusú (unáris) + és - operátorok precedenciája a legmagasabb. A kétoperandusú + és - operátorok precedenciája kisebb a *, / és % precedenciájánál. Az aritmetikai operátorok mindig balról jobbra haladva hajtódnak végre (a precedencia figyelembevételével).

Relációs és logikai operátorok

A C nyelv relációs operátorai: >, >=, <, <=. Ez a sorrend egyben a precedenciájuk sorrendje is. Ezeknél eggyel alacsonyabb precedenciájúak az egyenlőség operátorok: ==, !=.
Egy relációs vagy logikai kifejezés számértéke definíció szerint 0, ha a kifejezés hamis, és 1, ha igaz.
Az && és || (ÉS illetve VAGY) operátorokkal összekapcsolt kifejezések kiértékelése balról jobbra történik, és a kiértékelés azonnal félbeszakad, ha az eredmény igaz vagy hamis volta ismertté válik.
A ! unáris (egyoperandusú) negáló operátor a nem nulla (igaz) operandust 0 értékűvé (hamissá), a 0 értékű (hamis) operandust 1 értékűvé (igazzá) alakítja.

Inkrementáló és dekrementáló operátorok

A C nyelv két szokatlan operátort használ a változók inkrementálására (eggyel való növelésére) és dekrementálására (eggyel való csökkentésére). A ++ inkrementáló operátor egyet ad az operandushoz, a -- dekrementáló operátor pedig egyet kivon belőle.

A ++ és -- szokatlan vonatkozása, hogy prefix formában (a változó előtt elhelyezve, pl. ++n) és postfix formában (a változó után elhelyezve, pl. n++) egyaránt létezik. A kétféle változat egyaránt növeli (vagy csökkenti) a változót, de a ++n a felhasználás előtt, az n++ pedig utána növeli az n értékét (a -- operátor hasonlóan működik). Ebből következően minden olyan esetben, amikor a változó értékét is felhasználjuk (nem csak a növelésre vagy csökkentésre, azaz számlálásra van szükség), a ++n és az n++ különbözik. Ha pl. n értéke 5, akkor

```
x = n++;
```

hatására x értéke 5 lesz, amíg az

```
x = ++n;
```

hatására x értéke 6 lesz.

Bitenkénti logikai operátorok

A C nyelvben hat operátor van a bitenkénti műveletekre. Ezek az operátorok csak egész típusú adatokra, azaz char, short, int és long típusokra használhatók, akár előjeles, akár előjel nélküli változatban. Az egyes operátorok és értelmezésük a következő:

&	bitenkénti ÉS-kapcsolat
|	bitenkénti megengedő (inkluzív) VAGY-kapcsolat
^	bitenkénti kizáró (exkluzív) VAGY-kapcsolat
<<	balra léptetés
>>	jobbra léptetés
~	egyes komplemens képzés (unáris)

Értékadás operátor

- *Értékadás:*
- `i=2;`

  Az olyan kifejezések esetén, ahol a bal oldali érték megjelenik a jobb oldalon, pld.:
- `i=i+2;`

  tömörebb formában is írhatjuk:
- `i+=2;`

  Az értékadásnak van visszatérő értéke!

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

Operátorok összefoglalás precedencia és asszociativitás

Operátor	Asszociativitás
() [] ->	balról jobbra
! ~ ++ -- + - * & (típus) sizeof	jobbról balra
* / %	balról jobbra
+ -	balról jobbra
<< >>	balról jobbra
< <= > >=	balról jobbra
== !=	balról jobbra
&	balról jobbra
^	balról jobbra
|	balról jobbra
&&	balról jobbra
||	jobbról balra
?:	jobbról balra
= += -= *= /= %= &= ^= |= <<= >>=	balról jobbra

Vezérlési szerkezetek
Utasítások és blokkok

- Egy kifejezés utasítássá válhat, ha pontosvesszővel lezárjuk. Pld.:

```
x = 0;
i++;
printf();
```

A pontosvessző tehát utasításlezáró karakter

- A {} deklarációk és utasítások csoportját foglalja össze egyetlen összetett utasításba, vagy blokkba, amely szintaktikailag egy utasításnak felel meg.

Az if-else utasítás

- *Formája:*

```
if (kifejezés)
        1.utasítás
  else
        2.utasítás
```

Ahol az else rész opcionális.

- *Az else mindig a legközelebbi if-hez tartozik:*

```
if (n>0)
      if (a>b)
        z=a;
      else
        z=b;
```

A switch utasítás


```
switch (kifejezés){
   case állandó kifejezés:
      utasítások;
      break;
   case állandó kifejezés:
      utasítások;
      break;
   .
   .
   default:
      utasítások;
}
```

A `default` *ág opcionális;*

Az egyes eseteket `break` *utasítással törhetjük meg. Használata opcionális.*

Switch példa 1


```
float n1, n2;
char operator;
switch (operator){
 case '+':
  printf("%.1f + %.1f = %.1f",n1, n2, n1+n2);
  break;
 case '-':
  printf("%.1f + %.1f = %.1f",n1, n2, n1-n2);
  break;
 case '*':
  printf("%.1f + %.1f = %.1f",n1, n2, n1*n2);
  break;
 case '/':
  printf("%.1f + %.1f = %.1f",n1, n2, n1/n2);
  break;
 default:
  printf ("Hibás operátor!");
}
```

Switch példa 2


```
char c;
switch (c){
 case 'a': case 'e': case 'i': case 'o':
case 'u':
    printf("maganhangzo"); break;
 case ' ':
    printf("Space"); break;
default: printf ("Egyik sem");
}
```

Vége

Köszönöm a figyelmet!