Tartalomjegyzék

- 1 A C nyelv alapjai
  - 1.1 A C nyelv tulajdonságai
  - 1.2 Matematikusként miért tanulunk C-t?
  - 1.3 Egy C program futtatása
  - 1.4 Egy C program elemei
    - 1.4.1 Függvények
    - 1.4.2 Változók és típusaik
    - 1.4.3 Vezérl? struktúrák
    - 1.4.4 A C program formázása

A C nyelv alapjai

A C nyelv tulajdonságai

A C általános célú magas szint? nyelv (valójában a magasszint? és az alacsonyszint? nyelvek között helyezkedik el), melyet a Unix operációs rendszerhez Dennis Ritchie fejlesztett ki. 1989-ban szabványosítva lett: ANSI C (American National Standards Institute).

- Strukturált nyelv
- Szabványos könyvtári függvények
- Hordozható (a C kód számtalan gépen lefordítható és futtatható)
- Hatékony programok írására alkalmas (a magas szint? nyelvek közül a leghatékonyabbak egyike)
- Egyszer? nyelv, tömör szintaktika
- El?feldolgozó (makroprocesszor)

A nyelv gyengéi: könny? nehezen olvasható kódot írni, nehéz a hibák megtalálása, nincs automatikus memóriakezelés, tömbb?l kiírás veszélye,...

Matematikusként miért tanulunk C-t?

- A C nyelv több programnyelv alapja (C++, C#, Java,...)
- Más programnyelven írt kódok hatékonnyá tehet?k, ha a legtöbb gépid?t használó függvényeket C-ben újra írjuk. Kritikusan sok gépid?t igényl? algoritmusoknál segíthet.

Egy C program futtatása

A forrás egyszer? szövegfájl .c kiterjesztéssel, mely bármely szövegszerkeszt?vel szerkeszthet? (pl gedit, kate, emacs, vi,...). Például a hello.c program kódja a következ?:

```
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

Ez kiírja, hogy „Hello, World!"http://wiki.math.bme.hu. Miután elmentettük a programot, fordítsuk le, majd futtassuk:

```
$ gcc -W -Wall -o hello hello.c
$ hello
Hello, World!
$
```

Elhagyható a -o kapcsoló, ekkor egy a.out nev? fájl lesz a futtatható program, a -W és a -Wall több figyelmeztet? hibaüzeneteket ír ki, ami segíthet a hibák megtalálásában.

A fordítás lépései külön-külön is elvégezhet?k:

```
gcc -E hello.c >hello.ee    (csak az el?feldolgozó fut le           -> standard outputra ír)
gcc -S hello.c              (fordít, de az assembler nem            -> hello.s -- assembly kód)
gcc -c hello.c              (fordít, de nem szerkeszt (nem linkel)  -> hello.o -- object file)
gcc hello.c                 (el?feldolgoz+fordít+szerkeszt (linkel) -> a.out -- futtatható állomá
```

Néhány a megadható számtalan opció közül csak tájékoztatásképp:

```
$ gcc -W -Wall -s -O2 -lm -o <program> <program>.c
```

- *gcc*: a fordítóprogram neve. A *GNU C Compiler* és egyben a *GNU Compiler Collection* rövidítése. (Mi az a GNU?)
- *-W*: a legfontosabb figyelmeztet? üzeneteket (warning) bekapcsolja
- *-Wall*: még néhány fontos figyelmeztet? üzenetet bekapcsol
- *-s*: a fölösleges részeket (pl. nyomkövetési információk és szimbólumok) eltávolítja kimenetb?l. Nélküle nagyobb lenne a kimenet.
- *-O2*: bekapcsolja a második szint? optimalizálást. (A harmadik szint? esetén már lassabban fordul, és túl nagy lehet a kód. Az els? szint? esetén lassabb lesz a futó program, és esetleg túl nagy is. Vannak egyéb szintek is, például alapból a nulladik szint érvényes. Nem nulla, hanem nagy O bet? van a kapcsolóban.)
- *-lm*: a matematikai függvényeket (pl. *sqrt* és *cos*) tartalmazó matematikai függvénykönyvtárat teszi elérhet?ve. Emellett a forráskódban szerepelnie kell még az `#include <math.h>`-nak is a matematikai függvényekhez. Nem egyes, hanem kis l bet? van a kapcsolóban.
- *-o <program>*: a kimeneti futtatható fájl nevét adja meg
- *<program>*.c: bemeneti forrásfájl nevét adja meg

Egy C program elemei

Függvények

A hello.c programunk kódja meglep?en hosszú. Mit jelentenek a sorai? Az

```
int main(void)
```

vagy

```
int main()
```

vagy

```
 main()
```

sor minden C programban kötelez?, minden programban van egy main nev? függvény, és ennek meghívásával indul a program végrehajtása. Ha 0 a visszatérési értéke, akkor a program normálisan lefutott -- ez jelzés a külvilágnak. Ezt jelzi a return 0; parancs. Az

```
#include <stdio.h>
```

sorral azt jelezzük, hogy a „standard input-output"http://wiki.math.bme.hu függvények valamelyikét használni szeretnénk (itt pl. printf, azaz formázott kiírás).

Változók és típusaik

A következ? programban aritmetikai számítást is végzünk:

```
#include <stdio.h>
int main(void) {
    int n;
    n = 5;
    printf("az eredmeny: %d\n", 3*n + 1);
    return 0;
}
```

Itt változót deklarálunk, azaz megadtuk típusát (lehet pl. int, float, double, char, void,....), majd értéket adunk neki. Ez egy sorban is elvégezhet?:

```
int n = 5;
```

A kiírásban az idéz?jelbe tett szöveg mellett a kiírandó kifejezés értékének formátumát is megadjuk: %d az egész számok kiírását jelenti. Következzen a standard bemenet egyszer? használata, olvassunk be az 'a' és 'b' nev? változóba egy-egy egész számot, és ezzel számoljunk (kód: muveletek.c). A m?veletek: összeadás (+), kivonás (-), szorzás (*), egész osztás (/), maradékképzés (%), bitenkénti és (&), bitenkénti vagy (|), logikai és (&&), logikai vagy (||), ahol 0 a hamis, minden más igaz.

```
#include <stdio.h>

int main(void) {
    int a,b;

    printf("kerek egy pozitiv szamot: ");
    scanf("%d", &a);
    printf("kerek meg egy pozitiv szamot: ");
    scanf("%d", &b);

    printf("a ket szam osszege: %d\n", a+b);
    printf("a ket szam kulonbsege: %d\n", a-b);
    printf("a ket szam szorzata: %d\n", a*b);
    printf("a ket szam egesz hanyadosa: %d\n", a/b);
    printf("az osztasi maradek: %d\n", a%b);
    printf("a ket szam lebegopontos hanyadosa:%f\n", (float)a/b);
```

```
    printf("bitenkenti ES %d\n" ,a&b);
    printf("bitenkenti VAGY %d\n", a|b);
    printf("logikai ES %d\n" ,a&&b);
    printf("logikai VAGY %d\n", a||b);

    return 0;
}
```

A beolvasásnál nem 'a', hanem '&a' szerepel, ami az 'a' változó fizikai címét jelenti, erre kés?bb visszatérünk.

Vezérl? struktúrák

És akkor következzen egy program, amir?l nem is tudjuk, hogy véges id?ben leáll-e minden bemenetre, a Collatz-probléma collatz.c:

```
#include <stdio.h>
int main(void) {
    int x;
    printf("legyen x = ");
    scanf("%d", &x);
    while (x>1)
    {
        if (x % 2)
            x = 3*x + 1;
        else
            x /= 2;
        printf("%d ", x);
    }
    printf("\n");
    return 0;
}
```

Ebben szerepel egy elöl tesztel?s while ciklus és egy elágazás!

Felsoroljuk a C vezérl?struktúráit, hogy mindjárt egyszer?bb programokat írhassunk:

- if és if-else:

```
if (feltétel) {
    ...
}
```

vagy

```
if (feltétel) {
    ...
}
else {
    ...
}
```

vagy több feltétel esetén (itt is elhagyható az utolsó "http://wiki.math.bme.huelse"http://wiki.math.bme.hu ág)

```
if (feltétel1) {
    ...
}
else if (feltétel2) {
    ...
```

```
}
...
else if (feltételN) {
    ...
}
else {
    ...
}
```

- while-ciklusok:
  - ♦ hátultesztel?s

```
do {
    ...
} while (feltétel);
```

- ♦ elöltesztel?s

```
while (feltétel) {
    ...
}
```

- for-ciklus:

```
for (inicializálás; feltétel; végén) {
    ...
}
```

A for ciklus az *inicializálás*-ban megadott parancsokat végrehajtja, majd minden ciklus elején ellen?rzi a *feltétel*-t, és kilép, ha az hamis, végül minden ciklus végén végrehajtja a *végén* utasításait. Például

```
int i;
for (i=1; i<6; i++) {
    printf("%d ", i^2);
}
```

kiírja 1-t?l 5-ig a számok négyzeteit.

A C program formázása

A sor elején lev? szóközök számára vonatkozó beljebb kezdési szabály (sokféle változata lehetséges, itt csak a tárgy keretében használatosat értelmezzük):

- Beljebb kezdésre csak szóközöket használjunk, tabulátort nem.
- A nyitó és a záró kapcsost is írjuk külön sorba, és azonos oszlopban legyenek. (Ha esetleg a nyitó kapcsos a sor végére kerülne, a záró kapcsos elé pontosan annyi szóköz kerüljön, ahány szóköz volt a hozzá tartozó nyitó kapcsos zárójelet tartalmazó sor elején.)
- Minden egyéb sor elején pontosan négyszer (esetleg pontosan kétszer) annyi szóköz van, ahány (bezáratlan) kapcsos zárójelen belül van az adott sor.

További segítséget jelentenek a megjegyzések, ezeket /* és */ közé kell zárni, egy szokásos alkalmazásuk:

```
/* megjegyzés
  * megjegyzés
  * megjegyzés
  */
```

Egy másik elterjedt szabvány szerint // után írva adhatjuk meg a megjegyzést, mely a sor végéig tart.

A C programok olvashatóvá tétele sokat segít a kód kés?bbi megértésében, erre vonatkozó tanácsokat érdemes betartani. Egy ilyen leírás pl. itt található: http://www.gidnetwork.com/b-38.html Érdemes lesz kés?bb is visszatérni rá, és követni.

A kódban van némi lazaságra lehet?ség, pl. nem kiírni main típusát, mert az úgyis kötelez?en int, vagy nem visszaadni a 0 értéket, de ezekkel legyünk óvatosak.

```
#include <stdio.h>
main() {
    printf("Hello, World!\n");
}
```

A behúzás sem része a szintaktikának, a kód folyamatosan is írható, vagy más logika szerint tördelhet?, de ezt sose tegyük:

```
#include <stdio.h>
int main(){printf("Hello, World!\n");return(0);}
```

Egy szándékosan elrettent? példa mistery.c

```
#include <stdio.h>

main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){n\
l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```