Tartalomjegyzék

Egyéb hasznosságok C programozáshoz

Konstansok és makrók

A konstansok olyan "http://wiki.math.bme.huváltozók"http://wiki.math.bme.hu amiknek az értékét nem lehet megváltoztani. Például hasznos lehet ilyen, ha a programban a pi értékét sok helyen szeretnénk használni (ehhez elég lenne egy globális változó..), és még véletlenül sem szeretnénk felülírni az értékét a program futása során, hogy minden számítás ugyanazzal a pontossággal számolódjon. Többféle megoldást is használhatunk erre.

El?fordítóval

Konstansokat az eredeti C-ben csak az el?fordító segítségével hozhattunk létre. Minden olyan kódsor ami '#' jellej kezd?dik, a preprocesszornak vagyis az el?feldolgozónak szóló utasítás. (Kicsit lejjebb lesznek még példák.)

A "http://wiki.math.bme.hu#define"http://wiki.math.bme.hu preprocesszor-utasítással definiálhatunk konstansokat, vagy más néven makrókat. Csak abban a file-ban érvényes egy konstans/makró ahol deklaráltuk.

```
/* Konstansdeklaráció: eredeti C-s megoldás, el?fordítónak szóló utasítással */
#define PI 3.14159
#define NULLA 0
#define SZOVEG "Ha ezt a sort el tudja olvasni, nincs szüksége szemüvegre \n"
```

Itt nem kell pontosvessz? a sorok végére. Az el?fordító a program lefordítása el?tt a makró-szimbólumok minden el?fordulási helyére behelyettesíti a megfelel? értéket (bet?r?l bet?re, kiértékelés nélkül, lecseréli pl. a "http://wiki.math.bme.huPI"http://wiki.math.bme.hu-t "http://wiki.math.bme.hu3.14"http://wiki.math.bme.hu-re).

A konstansneveket mindig nagybet?vel kell írni!

C-ben egy konstansot megsemmisíteni is lehet az "http://wiki.math.bme.hu#undef konstansnév"http://wiki.math.bme.hu paranccsal, illetve ellen?rizhetjük az "http://wiki.math.bme.hu#ifdef"http://wiki.math.bme.hu ill. "http://wiki.math.bme.hu#ifndef"http://wiki.math.bme.hu preprocesszor-parancsokkal, hogy van-e (ill. nincs-e) már definiálva adott névvel konstans (ne felejtsük el lezárni az if-et):

```
#ifndef PI
#define PI 3.14
#endif
```

Makró-függvények

Akár kicsi *makró-függvények* írásásra is használhatjuk ezt a behelyeetesít? mechanizmust. De vigyáznunk kell arra hogy az el?fordító, mint nyelven kívüli eszköz mindent gondolkodás nélkül helyettesít, ráadásul az eredményt egy sorba írva azt sem teszi lehet?vé, hogy a makróhelyettesítést lépésenként nyomkövessük.

Egy elrettent? példa:

```
// a háromoperandusú feltételes operátort használjuk, mert rövid
#define abs(val) (val < 0) ? -val : val

int y, x = 3;
y = abs( x++ );     // Várt eredmény: x = 4, y = 3;
```

Az abszolút érték makró fenti alkalmazása esetén az el?fordító a hívás helyén (ahol y-nak adunk értéket) behelyettesíti az *abs()* makrot, a belsejébe pedig paraméterként behelyettesíti a "http://wiki.math.bme.huval"http://wiki.math.bme.hu helyére az "http://wiki.math.bme.hux++"http://wiki.math.bme.hu-t. Els? ránézésre azt várnánk, hogy az *y = abs(x++)* végrehajtása után, mivel el?tte x értéke 3 volt, x értéke 4 lesz, míg y értéke 3. Ez így is lenne, ha az abs-ot függvényként realizálnánk. Ezzel szemben a el?fordító ebb?l a sorból a következ?t készíti:

```
y = (x++ < 0) ? - x++ : x++;
```

azaz az x-et kétszer növeli, minek következtében az utasítás végrehajtása után x értéke 5, míg y-é 4 lesz. A el?fordítóval definiált makrók tehát veszélyesek lehetnek.

A const típusmódosítóval

Az ANSI (szabványos) C-ben (és majd C++-ban) a *const* típusmódosító szó segítségével bármely memóriaobjektumot definiálhatunk konstansként, vagyis "http://wiki.math.bme.hucsak olvasható változó"http://wiki.math.bme.hu-ként. Ez azt jelenti, hogy a fordító figyelmeztet, ha a változó nevét értékadás bal oldalán szerepeltetjük, vagy ebb?l nem konstansra mutató pointert inicializálunk.

```
/* ANSI C megoldás PI definiálására */
```

```
const float PI = 3.14;
```

Mutatók esetén lehet?ség van annak megkülönböztetésére, hogy a mutató által megcímzett objektumot, vagy magát a mutatót szeretnénk csak olvashatóvá tenni:

```
const char * p;  // p által címzett karakter nem módosítható, (const char) * p
char * const q;  // q mutatót nem lehet megváltoztatni, (char *) const q
```

Fájl olvasás és írás

Egy komoly adatfeldolgozást vagy bármilyen számításokat végz? program praktikus ha fájlokból olvassa be az adatokat, és az eredményeket is fájlba írja.

Sajnos a file-kezelés C-ben nem túl egyszer? és nem "http://wiki.math.bme.huprogramozóbarát"http://wiki.math.bme.hu, de néhány m?köd? példából kiindulva nem is olyan nehéz megoldani.

A fájlkezeléshez szükséges függvényeket az "http://wiki.math.bme.hustdio.h"http://wiki.math.bme.hu függvénykönvtár include-olásával tudjuk használni (ez kellett már korábban ahhoz is hogy a felhasználótól bekérjünk adatokat (vagyis a standard inputról olvassunk), vagy hogy a képreny?re (standard output) írjunk).

Két sorban így néz ki egy fájl megnyitása (az "http://wiki.math.bme.hur"http://wiki.math.bme.hu itt azt jeneti hogy olvasásra nyitjuk meg a szöveges fájlt):

```
FILE *fp;
fp = fopen("alma.txt", "r");
```

Az "http://wiki.math.bme.hur"http://wiki.math.bme.hu helyett lehetne
"http://wiki.math.bme.hub"http://wiki.math.bme.hu (bináris file olvasása),
"http://wiki.math.bme.huw"http://wiki.math.bme.hu (írás),
"http://wiki.math.bme.hua"http://wiki.math.bme.hu (a fájl végére írás (append)).

A "http://wiki.math.bme.huFILE * fp"http://wiki.math.bme.hu egy olyan mutató ami "http://wiki.math.bme.huFILE"http://wiki.math.bme.hu típusnev? struktúrára mutat. Ez a struktúra olyan információkat tartalmaz amik segítségével megvalósulhat a kapcsolat a programunk és az operációs rendszer (ami kezeli a fájlrendszert) között. A belsejét nem kell ismerni, elég azt tudni hogy hogy használjuk.

Fájl olvasás fscanf()-fel

A *scanf()* függvényt arra használtuk hogy a felhasználótól (a standard bemenetr?l) olvassunk vele egy-egy szót. Ehhez hasonló a *fscanf()*, ami fájlból (*input*) tud olvasni, amit olvasott azt az *output* változó(k)ba írja, itt egy karaktertömbbe(stringbe):

```
fscanf(input, "%s", output);
```

Azonban vigyázni kell vele, mert így csak az els? szóközig (egy szót) olvas be! általános szöveges fájl beolvasására kevéssé alkalmas, inkább akkor hasznos, ha ismert szerkezet? fájlt kell beolvasni. Például egy 3 dimenziós pont-koordinátákat tartalmazó fájlt beolvashatunk így, ha tudjuk hogy minden sorban egy pont adatai vannak, a sorokon belül pedig pontosan egy szóközzel vannak elválasztva a számok:

```
float x, y, z;
FILE *fp;
fp = fopen("koordinatak.txt", "r");
fscanf(fp, "%f %f %f", &x, &y, &z);
```

A fenti példában az fscanf() csak az els? sort olvassa be a fájlból, még egy ciklus kell köré hogy az egész fájlt feldolgozhassuk.

Példa: fileread1.c

Fájl olvasás fread()-del

Az *fread()* függvénnyel általánosabb file-beolvasót is írhatunk. Az *fread()* adott méret? blokkokat olvas be egy fájlból, elég nagy méret? blokk esetén akár az egészet. Visszatérési értéke hogy hány egységet (ált. karaktert) sikerült beolvasnia.

```
  size_t fread(void *ptr,     // ide olvassa be a fájl tartalmát
               size_t size,   // ekkora méret? elemekbe (tipikusan sizeof(char) lesz itt)
               size_t nmemb,  // ennyi darab (ennyi karakter) beolvasása maximum
               FILE *stream); // ebb?l a fájlból
```

A beolvasott adatoknak a heap-en kell dinamikusan memóriát foglalnunk, ehhez pedig meg kell tudnuk a fájl méretét:

```
// fp egy megnyitott file mutatója
 fseek(fp, 0L, SEEK_END);      // a file végére ugrunk az "olvasófejjel"
 long lFileLen = ftell(fp);    // a file hosszát elkérjük az ftell()-lel
 rewind(fp);                   // visszatekerünk a file elejére az "olvasófejjel"
```

A memóriába beolvasás után úgy dolgozzuk fel a fájl tartalmát (egy nagy karaktertömböt), ahogy akarjuk.

(Ha nagyon nagy fájllal dolgozunk ami nem fér a memóriába, akkor kisebb blokkokat olvassunk be egy ciklusban, és blokonként dolgozzuk fel a tartalmat.)

Példa: fileread2.c

(Megjegyzés: van még néhány módja a file olvasásnak (karakterenként is olvashatjuk a fájlt például), de általában a fenti két módszer egyikével elég jól megoldható bármilyen fájl-olvasási feladat.)

Fájl írás

Nézzünk egy teljes példát fájl írására is. A következ? kis példaprogram 0-tól kezdve számokat ír ki egy fájlba, a fájl nevét és a számok számát a felhasználótól kéri be:

```
#include <stdio.h>

int main() {
    int i, N;
    FILE *fp;        // fájl mutató
    char fname[80];  // a fájl neve lesz itt
    printf("Mennyi számot generáljak?\n");
    scanf("%d", &N);
    printf("A fájl neve ahova írjam:\n");
    scanf("%s", fname);
    fp = fopen(fname, "w");   // itt nyitjuk meg a fájlt, írásra
    if (fp == NULL) {         // hibakezelés
        printf("Nem sikerült megnyitni a fájlt: %s", fname);
        return 1;
    }
    for (i = 0; i < N; i++) {
        fprintf(fp, "%d\n", i);   // az "fprintf()" fájlba ír
    }
    fclose(fp);   // be is zárjuk a fájlt !
```

```
}
```

A programunk paraméterezése

Ha mindig fix számú bemenetre van szüksége a programunknak, akkor sokkal egyszer?bb ha azt nem mindig a program futása közben kell a felhasználónak bepötyögnie, hanem a program már az indulásakor megkapja. Így a programunk egy scripttel (kicsi futtatóprogrammal) is könnyebben futtatható és paraméterezhet? lesz.

Ehhez csak arra van szükség, hogy a *main()* függvényünknek legyen 2 paramétere:

- az els?: *int argc*: az argumentumok darabszáma ("http://wiki.math.bme.huargc"http://wiki.math.bme.hu : "http://wiki.math.bme.huargument count"http://wiki.math.bme.hu)
- a második: *char **argv* vagy *char *argv[]*: az argumentumokat(stringeket vagyis karakter-mutatókat) tartalmazó tömb, amib?l az els? (vagyis a nullás index?) maga a program neve amivel meghívták

```
#include <stdio.h>

int main(int argc, char **argv) {
    printf("A kapott argumentumok száma: %d\n", argc);
    while(argc--) {
        printf("%s\n", *argv++);
    }
    return 0;
}
```

Írjuk át kicsit a fenti számkiírós programunkat:

- az els? argumentumként a kiírandó számok darabszámát várja
- másodikként pedig a fájl nevét ahova ír
- ahelyett hogy 1-t?l (*n*-1)-ig ír ki számokat, random számokat írjon a fájlba

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

int main(int argc, char *argv[]) {
    // ellen?rizzük hogy a felhasználó jól használja-e a programot:
    // megkaptuk-e a várt paramétereket/argumentumokat?
    if (argc < 3) {
        printf("Használat: %s <file-nev> <darabszam>\n", argv[0]);
        return 1;
    }
    int N = atoi(argv[1]);    // ez egy string->int konvertáló függvény az stdlib.h-ból
    FILE *fp;                 // fájl mutató
    char *fname = argv[2];    // 2. argumentumként kaptuk a fájl nevét
    fp = fopen(fname, "w");   // itt nyitjuk meg a fájlt, írásra
    if (fp == NULL) {
        printf("Nem sikerült megnyitni a fájlt: %s", fname);
        return 1;
    }
    int i;
    for (i = 0; i < N; i++) {
        fprintf(fp, "%d\n", rand());   // a rand() függvény az stdlib.h része, random számokat gen
    }
    fclose(fp);   // be is zárjuk a fájlt !
    return 0;
```

```
}
```

Több fájlból álló program

Nagyobb programok esetén már nem lesz átlátható a kód ha egyetlen, sokezer soros C programfájlba írunk. Praktikus a kódot valamilyen logika mentén széttördelni: például a fájlkezel? függvényeket tehetjük egy külön fájlba, a program lényegi részének megvalósításától kicsit elszeparáltan. Nagy projektek esetén akár több száz forrásfájl tartalmazza a program m?ködéséhez szükséges összes kódsort.

A forráskódokat a fordító egyesével lefordítja (ún. object fájlokat készít), majd összelinkeli. A *gcc* mindkét lépést megteszi nekünk.

Több forrásfájlból álló program esetén figyelni kell néhány dologra:

- csak egy main() függvény lehet, ez lesz a program "http://wiki.math.bme.hubelépési pontja"http://wiki.math.bme.hu, itt kezd el futni
- a globális változók ütközhetnek, vagyis linkelési hibához vezet ha több fájlban is definiálva van ugyanolyan nev? változó vagy függvény (ilyenkor a hibaüzenetet az "http://wiki.math.bme.huld"http://wiki.math.bme.hu program küldi, ez végzi a linkelést)

Egy kicsi példa: két C forrásfájl összefordítása

Ha több c forráskódunk is van, amik hivatkoznak egymás függvényeire vagy globális változóira:

```
// prog1.c
#include <stdio.h>

int main() {
    int x = 3;
    int eredmeny = bar(x) + 15;
    printf("Az eredmeny: %d", eredmeny);
    return 0;
}

// myutils1.c
int bar(int j) {
    return j*j;
}
```

A fordításnál meg kell adni az összes szükséges forrásfájlt:

```
  gcc -o prog1 prog1.c myutils1.c
```

Header fájlok használata

A globális nevek ütközésének elkerülésére egy megoldás az ún. "http://wiki.math.bme.huheader"http://wiki.math.bme.hu fájlok használata, ezek "http://wiki.math.bme.hu.h"http://wiki.math.bme.hu kiterjesztés?ek, és általában deklarációkat, egyszer?bb definíciókat tartalmaznak. Direkt arra vannak tervezve, hogy ezekre a "http://wiki.math.bme.hu.c"http://wiki.math.bme.hu fájlokból hivatkozzunk *#include "http://wiki.math.bme.huakarmi.h"http://wiki.math.bme.hu* paranccsal (itt idéz?jelek vannak a "http://wiki.math.bme.hu<>"http://wiki.math.bme.hu zárójelek helyett a standard könvtárakkal ellentétben). A header fájlok általában kicsik, hogy hatékonyan lehessen include-olni ?ket.

A headerállományunk definiálja a bar() függvényt:

```
// myutils2.h
int bar(int j) {
    return j*j;
}
```

A C forrásfájlunk pedig include-olja a fentit:

```
// prog2.c
#include <stdio.h>
#include "myutils.h"

int main() {
    int x = 3;
    int eredmeny = bar(x) + 15;
    printf("Az eredmeny: %d", eredmeny);
    return 0;
}
```

A fordításnál elég megadni a prog.c-t (bele van írva hogy a "http://wiki.math.bme.humyutils2.h"http://wiki.math.bme.hu-t include-oljuk, ez kb olyan mintha a myutils2.h tartalmát a #include utasítás helyére behelyettesítenénk):

```
  gcc –o prog prog.c
```

És m?ködni fog.

A fontosabb C függvénykönyvtárak

Matematikai függvények

```
#include <math.h>
```

Trigonometriai függvények, stb. link

Standard I/O

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

Standard bemenet és kimenet (kapcsolattartás a felhasználóval), illetve fájlkezel? függvények. Pl:

- printf - képerny?re ír
- fprintf - fájlba ír
- scanf - felhasználótól beolvas
- fscanf - fájlból szót / formázottan olvas
- fopen - fájlt megnyit
- fread - fájlból blokkot olvas

link

Hasznos dolgok könyvtára: stdlib

```
#include <stdlib.h>
```

Sok hasznosság, pl a stringb?l számértékké konvertáló függvények, amiket gyakran használunk a program argumentumainak feldolgozásakor:

- int atoi(char* s)
- float atof(char* s)
- long atol(char* s)

És a dinamikus memória kezeléséhez használt függvények:

- malloc
- calloc
- free

Valamint a random számok generálásához:

- srand
- rand

link

Stringek

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

Stringek kezelése, másolása, összehasonlítása, stb link

Az el?adáshoz tartozó példakódok

Letölthet?k innen.

Ellen?rz? kérdések

Amit a zh-ra is tudni kell ebb?l az el?adásból:

- konstans értékek kétféle definiálása
- mi a makrófüggvény
- main paraméterezése
- file olvasás, file írás felismerés szintjén

ZH-infók

Id?pont: márc. 27. kedd 16-18:00.

A pontok eloszlása a feladattípusokon:

- 6 pont "http://wiki.math.bme.humitírki"http://wiki.math.bme.hu (3 kisebb C-kód értelmezése)
- 10 pont függvényírás: 3 kis függvény (lesz közötte olyan (talán kett? is) ami gyakorlaton is volt vagy ahhoz nagyon hasonló)
- 6 pont elmélet : 4 kérdés (2 könny? (kb mint a gyakorlatok elején voltak a kisZh-k) + 2 kicsit kifejt?sebb)
- 4 pont hibakeresés : egy picit hosszabb, elvileg teljes C-kódban 4 hiba lesz elrejtve

Források és további olvasnivalók:

- http://www.eet.bme.hu/publications/e_books/progr/cpp/node12.html
- http://prog.hu/cikkek/292/Konstansok+operatorok+ciklusok.html
- http://www.tankonyvtar.hu/informatika/objektum-orientalt-080905-103
- http://www.math.utah.edu/~carlson/c/cbook.pdf
- http://www.cprogramming.com/tutorial/c/lesson14.html
- http://publications.gbdirect.co.uk/c_book/chapter10/arguments_to_main.html
- http://en.wikipedia.org/wiki/Include_guard
- http://www.delorie.com/djgpp/doc/ug/larger/multisrc.html
- The C library reference guide: http://www.acm.uiuc.edu/webmonkeys/book/c_guide/index.html