Karakterláncok (sztringek) kezelése

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2021 október 20.

Karakterláncok (stringek)

Emlékeztető: egy bájton 256 különböző értéket lehet tárolni $\Rightarrow$ felhasználhato karakterkódolásra

- a szövegek karakterenként tárolódnak
- egyszerűbb eset: 1 bájt = 1 betű
- a bájtok értékeit betűkhöz kell rendelni
- az alapkaraktereket az ASCII szabvány definiálja
- az ékezetes betűket valamilyen kódlap szerint osztjuk ki
  pl. magyar nyelvhez: ISO 8859-2 (Latin-2), Windows-1250

Unicode karakterek (kitekintés)

- gyakorlatilag tetszőleges nemzetközi szöveg reprezentálására
- a szabvány majdnem 140 ezer karaktert definiál
- plusz speciális vezérlőjelek
- legalább 4 bájt kellene az egyedi karakterek tároláshoz
- egy nyelv egyszerre sosem használja az összes karaktert
- ezért helyette általában kódolás: UTF-8, stb.

Alapvető speciális vezérlőkarakterek

Ezekhez nem tartozik betű, hanem valamilyen *hatásuk* van

- eredetileg a nyomtató vezérlésére használták
- a terminálok emulálják a nyomtató működését
- a C nyelv ehelyett \-sel kezdődő ún. escape-szekvenciákat is elfogad

Gyakran használt vezérlőkarakterek
Ezeket és a szóközt karaktert együtt *whitespace*-nek hívjuk

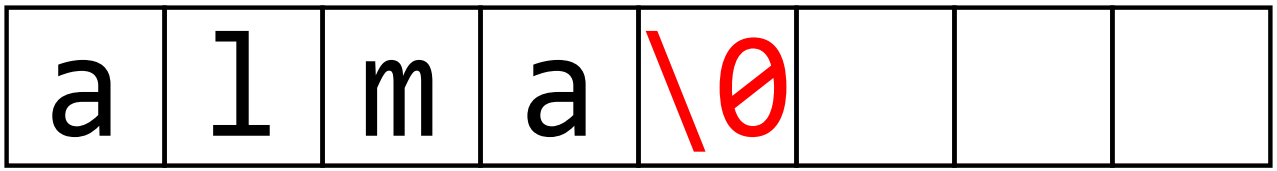
- `\t`: tabulátor (pl. fájlokban oszlopok között)
- `\n`: új sor
- `\r`: kocsi vissza (ld. nyomtatók)[1]

[1] Újsor jel Windows-on: `\r\n`, régi Mac-en: `\r`

Karakterláncok (stringek) a C nyelvben

A C nyelvben nincsen külön `string` típus!

- helyette: `char*` típusú pointer mutat az első karakterre
- a karakterek egymás után folytonosan a memóriában
- a legutolsó karakter *után* kötelezően áll egy `\0`, ún. NULL karakter
- figyelem! emiatt mindig eggyel több bájtot kell allokálni, mint a szöveg maximális hossza



Szöveges konstansok

A C nyelvben ugyanakkor van *string konstans*!

- a stringkonstansokat dupla idézőjel jelzi, pl.: `"alma"`
- a stringkonstansok végére a fordító automatikusan kiteszi a lezáró `\0`-t
- létezik *üres string*: `""`: ez egyetlen bájtból áll, aminek 0 az értéke

```
1  int main() {
2    char *s = "alma";
3    printf("%s\n", s);
4    return 0;
5  }
```

Figyelem!

- a karakterkonstansokat szimpla idézőjel jelzi, pl.: `'a'`, `'\t'` stb.
- tehát az egyetlen karakterből álló sztring különbözik a karakterkonstantól! sztring esetén van egy lezáró `\0`
- üres karakter nincs, `''` hibás!

Műveletek stringekkel

Mivel nincsen `string` típus, ezért ezen ható operátorok sincsenek

- helyette függvények a szöveges adatok kezelésére
- külön könyvtárak különböző karakterkódolásokhoz

Néhány alapvető függvény

- `strlen`: string hossza, az utolsó `\0`-t nem beszámítva
- `strcmp`: két string összehasonlítása
- `strcpy`: egyik string másolása másikba
- `strcat`: egyik string hozzáfűzése egy másikhoz
- `sprintf`: formátumozott string készítése
- ezen függvények használatához a `<string.h>` header szükséges

Figyelem!

- a stringkezelő függvények mindig a `\0` karaktert várják a végén
- a bemenetük `char*`, és nem tudják, hogy mennyi memória lett foglalva a szövegnek

Szöveges fájlok kezelése

A szöveges fájlok valamilyen karakterkódolással vannak tárolva

- a C nyelv alapértelmezésben a rendszer egybájtos kódolását használja
- adatfájloknál a legegyszerűbb valami külső programmal egybájtosra konvertálni
- linuxon pl. az `iconv` programmal

A szöveges fájlok sorokból állnak

- a sorok végén új sort jelölő karakter (`\r`, `\n` stb.)
- de egy sor *akármilyen hosszú lehet*!

Alapvető függvények szöveges fájlok olvasására

- `fscanf`: formátumozott olvasás (korábban már láttuk)
- `fgets`: egyetlen sor beolvasása
  első új sor jelig, a fájl végéig, vagy korlátos karakterszámig
- `fgetc`: egyetlen karaktert olvas be

Bájt szintű, bináris olvasás

- `fread`: megadott számú bájtot olvas (vagy a fájl végéig)

Szöveges fájlformátumok

Néhány alapvető adatfájl-formátum

- CSV: comma-separated values: oszlopok, vesszőkkel elválasztva
- tabular: oszlopok, általában `\t` karakterekkel elválasztva
- fix számú karakterből álló oszlopok
  itt a számok fix tizedesjeggyel, lehetnek jobbra igazítva

Általában előfordulnak speciális sorok

- fejléc, ami az oszlopok nevét, esetleg típusát tartalmazza
- megjegyzés, amit többnyire speciális karakter vezet be
- pl üres sor, mátrix egyes sorait tartalmazó adatblokkok között

Egyszerű beolvasás `fscanf`-fel

Ha a feladat csupán azonos típusú adatok egymás utáni beolvasása

- legegyszerűbben a `fscanf` függvény hívogatásával
- `fscanf` átugorja a whitespace-eket!

Pl. *n* szám beolvasása egy sorból
1.2 24.6 100.01 ...

```
1
2   int main () {
3
4      int n=20;
5      double act_num;
6      FILE *fp;
7
8      fp = fopen("inpfile.txt", "r");
9
10     for (int i=0; i<n;i++)
11     {
12     fscanf(fp, "%lf", &act_num);
13     printf("current number %lf",act_num)
14     }
15     close(f);
16   return(0);
17  }
```

Egyszerű beolvasás `fscanf`-fel

Pl: dátumok beolvasása, amelyek egymás alatti sorokban helyezkednek el “év hónap nap” formátumban

2017 augusztus 7
2019 február 8
:
:
stb.

```c
#inlcude<stdio.h>
#inlcude<stdlib.h>

int main () {
   char str1[20];
   int year, day;
   FILE *fp;

   fp = fopen("inpfile.txt", "r");

   while (fscanf(fp, "%d  %s  %d", &year, str1, &day)==3)
   {
     // fscanf visszateresi erteke integer
     printf("%d  %s  %d\n",year, str1, day);
     // beolvasott adatok feldolgozasa
   }

 return(0);
}
```

Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor
- legfeljebb `maxstrl` karaktert olvasunk be soronként

Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor
- legfeljebb `maxstrl` karaktert olvasunk be soronként
- ha az első karakter '#', akkor azt a sort írja ki

Mátrix beolvasása szöveges fájlból soronként

Tekintsük a következő példa fájlt:

- a fájl elején fejléc, ahol a sorok '#'-sel kezdődnek
- az adatok egy oszlopban helyezkednek el
- a mátrix egy sorába tartozó számok egymás alatt vannak
- a különböző sorokhoz tartozó számokat tartalmazó adattömbök között egy üres sor van

```
1
2     if(finp == NULL) {
3        printf("Error opening input file");
4        return(-1);}
5
6
7     while (fgets(str,maxstrl,finp)!=NULL )
8     {
9       if (str[0]=='#')
10      {printf("%s",str);}
11      else if (strlen(str)>1)
12      {
13       if (vektindx>maxdim)
14       {
15        printf("Max vector size reached\n");
16        exit(-1);
17       }
18       else {
19       matr[vektindx]=atof(str);
20       vektindx++;
21       }
22      }
23    }
```

- bemeneti fájl neve parancssori argumentum
- `fgets`-sel olvasunk be sorokat, amig a fájl végére nem érünk.
- `char str[maxstrl]` stringtömbe kerül az beolvasott sor
- legfeljebb `maxstrl` karaktert olvasunk be soronként
- ha az első karakter '#', akkor azt a sort írja ki
- ha a sor nem üres, akkor alakítsa a stringet `float` számmá és helyezze el