Karakterláncok (sztringek) kezelése

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2019. október 7.

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.

Az ASCII kódtábla

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

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

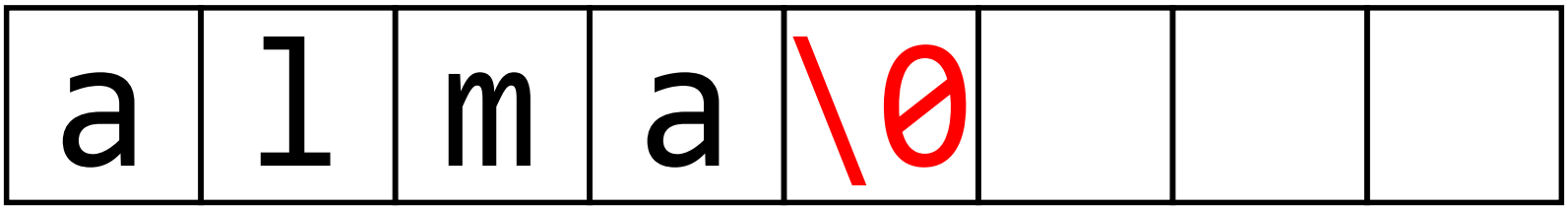
- `\t`: tabulátor (pl. fájlokban oszlopok között)
- `\n`: új sor
- `\r`: kocsi vissza (ld. nyomtatók)[1]

[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 viszont szimpla idézőjel jelzi, pl.: `'a'`, `'\t'` stb.
- üres karakter nincsen, `''` 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: dátumok beolvasása, amelyek egymás alatti sorokban helyezkednek el "év hónap nap" formátumban

2017 augusztus 7
2019 február 8
⋮
stb.

```
1  #inlcude<stdio.h>
2  #inlcude<stdlib.h>
3
4  int main () {
5     char str1[20];
6     int year, day;
7     FILE *fp;
8
9     fp = fopen("inpfile.txt", "r");
10
11    while (fscanf(fp, "%d  %s  %d", &year, str1, &day)==3)
12    {
13      // fscanf visszateresi erteke integer
14      printf("%d  %s  %d\n",year, str1, day);
15      // beolvasott adatok feldolgozasa
16    }
17
18  return(0);
19 }
```

Bonyolultabb adatfájl formátum

Ha bonyolultabb az adatfájl formátuma, pl tartalmaz fejlécet, megjegyzést, üres sorokat stb akkor probálkozhatunk soronkénti beolvasással és a sorok feldolgozásával

Bonyolultabb adatfájl formátum

Ha bonyolultabb az adatfájl formátuma, pl tartalmaz fejlécet, megjegyzést, üres sorokat stb akkor probálkozhatunk soronkénti beolvasással és a sorok feldolgozásával

Pl. A fájl elején fejléc, ahol a sorok '#'-sel kezdődnek.

Pl. a `gnuplot` a következő formátumban olvas be egy mátrixot:
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.

Pl: Mátrix beolvasása szöveges fájlból soronként

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

Pl: Mátrix beolvasása szöveges fájlból soronként

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

Pl: Mátrix beolvasása szöveges fájlból soronként

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

Pl: Mátrix beolvasása szöveges fájlból soronként

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 és van még hely a lefoglalt adatvektorban, akkor alakítsa a stringet `float` számmá és helyezze el

Mátrix beolvasása szöveges fájlból soronként

A példaprogram:

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


double *alloc_vec(int n){
  double *v = (double *)malloc(n * sizeof(double));
  if (v == 0) { printf(" Not enough memory\n");
  exit(-1);}
 return v ;
 }


int main(int argc, char *argv[])
 {
  int maxstrl=300;
  int vektindx=0;
  int maxdim=200;
  char str[maxstrl];

  double *matr=alloc_vec(maxdim);

  FILE *finp = fopen(argv[1],"r");

   if(finp == NULL) {
      printf("Error opening input file");
      return(-1);}
   while (fgets(str,maxstrl,finp)!=NULL )
   {
     if (str[0]=='#')
     {printf("%s",str);}
     else if (strlen(str)>1)
     {
      if (vektindx>maxdim)
      {
       printf("Max vector size reached\n");
       exit(-1);
      }
      else {
      matr[vektindx]=atof(str);
      vektindx++;
      }
     }
   }

   free(matr);
  fclose(finp);

  return 0;
}
```

Mátrix beolvasása szöveges fájlból soronként

A példaprogram:

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


double *alloc_vec(int n){
  double *v = (double *)malloc(n * sizeof(double));
  if (v == 0) { printf(" Not enough memory\n");
  exit(-1);}
 return v ;
 }


int main(int argc, char *argv[])
 {
  int maxstrl=300;
  int vektindx=0;
  int maxdim=200;
  char str[maxstrl];

  double *matr=alloc_vec(maxdim);

  FILE *finp = fopen(argv[1],"r");

   if(finp == NULL) {
      printf("Error opening input file");
      return(-1);}
   while (fgets(str,maxstrl,finp)!=NULL )
   {
     if (str[0]=='#')
     {printf("%s",str);}
     else if (strlen(str)>1)
     {
      if (vektindx>maxdim)
      {
       printf("Max vector size reached\n");
       exit(-1);
      }
      else {
      matr[vektindx]=atof(str);
      vektindx++;
      }
     }
   }

   free(matr);
  fclose(finp);

  return 0;
}
```

Mi történik, ha nem tudjuk előre, hogy mekkora lesz a beolvasott mátrix mérete ?

Ismeretlen nagyságú adatsor beolvasása vektorba

Dinamikusan foglalt memória terület nagysága a program futása során változtatható `realloc` függvénnyel

```
1
2   double *dynvect=(double *)malloc(init_size); //dinamikus adatfoglalas  egy init_size
3                                                //meretu memoriablokkra
4
5   // egy act_size valtozoban taroljuk, hogy mennyi adat  van pillanatnyilag
6   // a lefoglalt memoriablokkban
7
8     if (act_size < init_size)
9       {
10      // beolvasott adat elhelyezese a dynvect-be
11     act_size++;
12     }
13    else
14    {
15      dynvect=(double *)realloc(dynvect,new_vect_size);
16      // kiterjesztjuk az eredetileg lefoglalt memoriatombot
17      // beolvasott adat elhelyezese a dynvect-be
18    }
```

Még bonyolultabb adatelrendezés

`fgetc` fügvénnyel karakterenként beolvasni és feldolgozni az adatokat