AS-0.1103 C-ohjelmoinnin peruskurssi
Aalto-yliopiston sähkötekniikan korkeakoulu
Tentti 05.01.2012, Raimo Nikkilä

Ohjeet

Kaikki ohjelmointitehtävät tulee toteuttaa C-kielellä hyvää ohjelmointityyliä noudattaen.

Tentti arvostellaan asteikolla 0-25p ja kaikkien tehtävien painoarvo on sama (5 pistettä / tehtävä).

Vastaa **viiteen** vapaavalintaiseen tehtävään.

Tentissä saa käyttää funktiolaskinta sekä tentissä jaettua C-kielen standardikirjaston minireferenssiä. Kaikki tentin tehtävät ovat tehtävissä minireferenssissä listatuilla funktioilla mutta kaikkia C-kielen standardikirjaston funktioita saa halutessaan käyttää.

Tavussa (`char`) on aina 8 bittiä kaikissa tehtävissä ja eniten merkitsevä bitti on aina vasemmalla.

Kirjoita edes opiskelijanumerosi selkeällä käsialalla tenttipapereihin.

1. Tehtävä

Mitä alla oleva ohjelma tulostaa?

Vastaukseksi riittää pelkkä tuloste ilman perusteluja.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  size_t function(int* arr, size_t n)
5  {
6    size_t w = 0;
7    for (size_t r = 0;r < n;r++)
8      if (arr[r] % 2)
9        arr[w++] = arr[r];
10   return w;
11 }
12
13 int main(void)
14 {
15   int arr1[6] = {1,2,3,4,5,6};
16   int arr2[6] = {-1,-2,-3,-4,-5,-6};
17
18   int **arr = malloc(4 * sizeof(int *));
19   arr[0] = arr1;
20   arr[1] = arr2;
21   arr[2] = arr1 + 3;
22   arr[3] = arr2 + 3;
23
24   printf("A:");
25   for (size_t i = 0;i < 3;i++)
26     printf(" %d", arr[0][i] + arr[2][i]);
27   printf("\n");
28
29   printf("B: %ld %d\n", arr[1] - arr[3], arr[3][0]);
30
31   printf("C: %d %d\n", **(arr + 1), *(arr[2] + 2));
32
33   printf("D:");
34   size_t n = function(arr[0], 6);
35   for (size_t i = 0;i < n;i++)
36     printf(" %d", arr[0][i]);
37   printf("\n");
38
39   printf("E:");
40   for (size_t i = 0;i < 6;i++)
41     printf(" %d", arr1[i]);
42   printf("\n");
43
44   free(arr);
45 }
```

2. Tehtävä

Morsekoodaus on yksinkertainen tapa välittää tietoa käyttäen vain kahta merkkiä, pistettä ja viivaa.

Toteuta funktio `morseEncode()`, joka tekee morsekoodauksen biteillä, siten että jokaisen merkin esitykseen koodauksessa käytetään kaksi bittiä.

```
1 unsigned char* morseEncode(const char* str);
```

Funktio ottaa parametrina morsekoodauksen merkkijonona, jossa on '-', '.' ja ' ' merkkejä, ja tuottaa bittikoodatun esityksen merkkijonosta.

Sanojen välit ilmaistaisiin normaalisti tauoilla mutta bittiesityksessä väli vaatii oman koodinsa. Samoin viestin lopun ilmaisemiseen käytetään erillistä koodia.

Toteuta morsekoodaus käyttäen oheista koodausta:

Merkki	Koodaus bitteinä
Viiva	10
Piste	11
Sanaväli	01
Viestin lopetus	00

- Voit olettaa että `malloc()`-, `calloc()`- ja `realloc()`-funktiot onnistuvat aina.
- Voit olettaa että annetussa merkkijonossa ei ole ylimääräisiä merkkejä.
- Mahdolliset ylimääräiset bitit koodauksen lopussa asetetaan nolliksi.

Esimerkki koodauksesta:

```
morseEncode("-- --- .-. ... ."); // MORSE
Tuottaa koodauksen: {0xA6, 0xA7, 0xB7, 0xF7, 0x00}
```

A	.-	M	--	Y	-.--	6	-....
B	-...	N	-.	Z	--..	7	--...
C	-.-.	O	---	Ä	.-.-	8	---..
D	-..	P	.--.	Ö	---.	9	----.
E	.	Q	--.-	Ü	..--	.	.-.-.-
F	..-.	R	.-.	Ch	----	,	--..--
G	--.	S	...	0	-----	?	..--..
H	....	T	-	1	.----	!	..--.
I	..	U	..-	2	..---	:	---...
J	.---	V	...-	3	...--	"	.-..-.
K	-.-	W	.--	4	....-	'	.----.
L	.-..	X	-..-	5	.....	=	-...-

Kansainvälinen Morse aakkosto

3. Tehtävä

Selitä lyhyesti ja korkeintaan kahdella lauseella mitä kukin kolmesta alla olevasta funktiosta yleisesti ottaen tekee.

Älä kuvaa funktion sisäistä toimintaa vaan kerro minkä toiminnon funktio toteuttaa tai minkä arvon se laskee syötteistään.

```
1  include <string.h>
2
3  void function1(char *str, size_t n)
4  {
5    if (n < strlen(str))
6      str[n] = '\0';
7  }
8
9  void function2(char *str, size_t n)
10 {
11   size_t s = strlen(str);
12   if (n >= s)
13   {
14     *str = '\0';
15     return;
16   }
17   for (size_t i = 0;i + n <= s;i++)
18     str[i] = str[i + n];
19 }
20
21 size_t function3(char **arr, const char *str)
22 {
23   size_t c = 0;
24   for (char** p1 = arr;*p1;p1++)
25     for (const char *p2 = *p1;*p2;p2++)
26     {
27       int f = 0;
28       for (const char *s = str;*s;s++)
29         if (*p2 == *s)
30         {
31           f = 1;
32           break;
33         }
34       c += f;
35     }
36   return c;
37 }
```

4. Tehtävä

Toteuta funktio `readMatrix()`, joka lukee liukulukumatriisin tiedostosta.

Funktio ottaa parametrinaan tiedostonnimen, josta matriisi luetaan.

Funktio palauttaa dynaamisesti (`malloc()`) varatun `struct Matrix` rakenteen luetusta matriisista.

```
1  #include <stddef.h>
2
3  struct Matrix
4  {
5    double **values;
6    size_t nrows; // Rivit
7    size_t ncols; // Sarakkeet
8  };
9
10 struct Matrix* readMatrix(const char* fname);
```

- Voit olettaa että `malloc()`-, `calloc()`- ja `realloc()`-funktiot onnistuvat aina.
- Mikäli tiedostoa ei saada avattua, funktio palauttaa `NULL` arvon.
- Tiedoston kaksi ensimmäistä arvoa ovat kokonaislukuja jotka kertovat matriisin rivien ja sarakkeiden määrät.
- Voit olettaa, että tiedosto on aina oikeassa muodossa.

Esimerkiksi syötetiedoston rakenteesta 2x3 kokoisella matriisilla:

```
2 3
1.0 -2.5 7.0
0.5 12.0 2.0
```

5. Tehtävä

Ohessa on yksinkertaisia tekstilistoja käsittelevä (kohtuu huono) ohjelma. Vastaa **lyhyesti** seuraaviin kysymyksiin annetun ohjelman perusteella.

A) Millä kahdella muulla tavalla tietueen vaatima muistin määrä voitaisiin ilmaista rivillä 12?

B) Miksei rivillä 13 tarvita `sizeof(char)` kerrointa?

C) Mikä selkeä toiminnallinen rajoite tai vika rivin 22 `findText()` funktiossa on?

D) Entä rivin 32 `removeText()` funktiossa?

E) Mikä muistinvapautus tai mitä muistinvapautuksia pitäisi tehdä `removeText()` funktiossa ja missä?

```
1  #include <stdlib.h>
2  #include <string.h>
3
4  typedef struct text_s
5  {
6    char *content;
7    struct text_s *next;
8  } Text;
9
10 void addText(Text *txt, const char *s)
11 {
12   Text *new = malloc(sizeof(*new));
13   new->content = malloc(strlen(s) + 1);
14   new->next = NULL;
15   strcpy(new->content, s);
16
17   while (txt && txt->next)
18     txt = txt->next;
19   txt->next = new;
20 }
21
22 const char* findText(Text *txt, const char *s)
23 {
24   while (txt && txt->next)
25     if (!strcmp(txt->content, s))
26       return s;
27     else
28       txt = txt->next;
29   return NULL;
30 }
31
32 void removeText(Text *txt, const char *s)
33 {
34   const char* p = findText(txt, s);
35   if (!p)
36     return;
37   for (;txt->next->content != s;txt = txt->next);
38   Text *tmp = txt->next->next;
39   txt->next = tmp;
40 }
```

6. Tehtävä

Käsitellään yksinkertaisia liukuvuja käyttäviä funktioita. Kaikki tällaiset funktiot ottavat parametrinaan liukuluvun ja palauttavat liukuluvun.

Tarkoitus on kerätä näitä funktioita taulukkoon `Functions` tietotyypin sisälle, jolloin kaikkia funktioita on mahdollista kutsua järjestyksessä jollekin arvolle.

Määrittele tätä tarkoitusta varten tietotyyppi `Functions`, johon kuuluu ainakin funktio-osoitintaulukko edellämääriteltyjä funktioita. Toteuta lisäksi funktiot `addFunction()` ja `callFunctions()`.

Funktio `addFunction()` ottaa parametrinaan osoitteen `Functions` tietotyyppiin, sekä yhden funktio-osoittimen. Funktio lisää tämän funktio-osoittimen tietotyypin taulukkoon viimeiseksi jäseneksi.

Funktio `callFunctions()` ottaa parametrinaan osoitteen `Functions` tietotyyppiin, sekä yhden liukuluvun. Funktio kutsuu kaikkia `Functions` tietotyypin funktioita järjestyksessä tälle liukuluvulle, siten että edellisen funktion paluuarvo annetaan aina seuraavan funktion parametriksi. Toisinsanoen, esimerkiksi kolmella funktiolla: $f_1(f_2(f_3(parametri)))$; Funktio palauttaa saadun lopputuloksen.

- Voit olettaa että `malloc()`-, `calloc()`- ja `realloc()`-funktiot onnistuvat aina.
- Voit ilmaista `Functions` tietotyypin taulukon koon haluamallasi tavalla.

C99 minireferenssi

Tentissä tarvittavat tietotyypit ja niiden koot

Tyyppi	Koko tavuina	Lukualue	I/O muotoilumääre
`char`	1	-127 ... 127	%c
`unsigned char`	1	0 ... 255	%c
`int`	4	-2,147,483,647 ... 2,147,483,647	%d
`unsigned int`	4	0 ... 4,294,967,295	%u %x %X
`long`	8	Riittävän laaja $(-2^{63}...2^{63}-1)$	%ld
`size_t`	8	Riittävän laaja $(0...2^{64}-1)$	%zu
`double`	8	Riittävän laaja	%lf
`void*`	8	Kaikki (objekti-)osoitinmuuttujat	%p

Operaattoripresedenssi korkeimmasta matalimpaan ja assosioituvuus

`() [] . → ++ --`	vasemmalta oikealle	postfix ++ ja --
`++ -- + - ! ~ (tyyppi) * & sizeof`	oikealta vasemmalle	prefix ++ ja --, * ja & muistioperaattoreina
`* / %`	vasemmalta oikealle	* aritmeettisena operaattorina
`+ -`	vasemmalta oikealle	
`<< >>`	vasemmalta oikealle	
`< <= > >=`	vasemmalta oikealle	
`== != & ^ | && ||`	vasemmalta oikealle	& loogisena operaattorina
`?:`	oikealta vasemmalle	
`= op=`	oikealta vasemmalle	kaikki sijoitusoperaatiot
`,`	vasemmalta oikealle	pilkku operaattorina

C99 varatut sanat

`auto`	`break`	`case`	`char`	`const`	`continue`	`default`	`do`
`double`	`else`	`enum`	`extern`	`float`	`for`	`goto`	`if`
`inline`	`int`	`long`	`register`	`restrict`	`return`	`short`	`signed`
`sizeof`	`static`	`struct`	`switch`	`typedef`	`union`	`unsigned`	`void`
`volatile`	`while`	`_Bool`	`_Complex`	`_Imaginary`			

Tentissä tarvittavat standardikirjaston funktiot

ctype.h

```
int isalnum(int ch);   int isalpha(int ch);   int isdigit(int ch);   int islower(int ch);
int ispunct(int ch);   int isspace(int ch);   int isupper(int ch);   int tolower(int ch);
int toupper(int ch);
```

math.h

```
double pow(double base, double exponent);   double sqrt(double value);
```

stdio.h

```
int printf(const char* format, ...);
int sprintf(char* str, const char* format, ...);
int fscanf(FILE* stream, const char* format, ...);
FILE* fopen(const char* path, const char* mode);
int getchar(void);
int putchar(int ch);
int puts(const char* str);
char* fgets(char* str, int size, FILE* stream);
int fprintf(FILE *stream, const char* format, ...);
int scanf(const char* format, ...);
int sscanf(const char* str, const char* format, ...);
int fclose(FILE* fp);
int fgetc(FILE *stream);
int fputc(int ch, FILE* stream);
int fputs(const char* str, FILE* stream);
```

stdlib.h

```
void* calloc(size_t nmemb, size_t size);   void free(void* ptr);   void abort(void);   int abs(int);
void* realloc(void* ptr, size_t size);   void* malloc(size_t size);   void exit(int);
void qsort(void* base, size_t nmemb, size_t size, int (*cmp)(const void*, const void*));
```

string.h

```
char* strcat(char* dest, const char* src);
int strcmp(const char* str1, const char* str2);
char *strstr(const char* haystack, const char* needle);
size_t strxfrm(char* dest, const char* src, size_t n);
char* strchr(const char* str, int ch);
char* strcpy(char* dest, const char* src);
size_t strlen(const char *str);
void* memset(void* s, int c, size_t n);
```