AS-0.1103 C-ohjelmoinnin peruskurssi
Aalto-yliopiston teknillinen korkeakoulu
Tentti 06.07.2010, 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.

Jaa vastauksesi konsepteille siten, että:

Tehtävät **1. ja 2.** ovat omalla konseptillaan

Tehtävät **3. ja 4.** ovat omalla konseptillaan

Tehtävät **5. ja 6.** ovat omalla konseptillaan

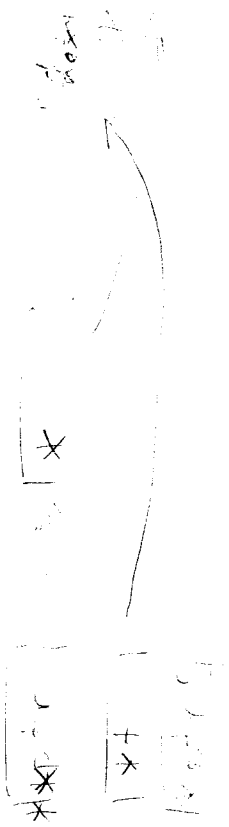
Tavussa (`char`) on aina 8 bittiä kaikissa tehtävissä ja eniten merkitsevä bitti on aina vasemmalla.

Kirjoita edes opiskelijanumerosi selkeällä käsialalla tenttipapereihin.

1. 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 <stdio.h>
2  #include <stdlib.h>
3
4  void function1(void)
5  {
6      unsigned int s;
7
8      if (!scanf("%u", &s))
9          return;
10
11     for (unsigned int i = 0;i < 2*s + 1 ;i++)
12     {
13         for (unsigned int j = 0;j < 2*s + 1 ;j++)
14             if (i == j || ((2 * s - i) == j))
15                 putchar('*');
16             else
17                 putchar(' ');
18         putchar('\n');
19     }
20 }
21
22 size_t function2(const char *ptr)
23 {
24     static int s = 0;
25     if (!*ptr)
26     {
27         size_t t = s;
28         s = 0;
29         return t;
30     }
31     s++;
32     return function2(ptr + 1);
33 }
34
35 struct s
36 {
37     void* (*d)(const void*);
38     struct s* next;
39 };
40
41 void function3(struct s** ptr)
42 {
43     struct s* t = *ptr;
44
45     if (!*ptr || !(*ptr)->next)
46         return;
47
48     *ptr = (*ptr)->next;
49     t->next = NULL;
50
51     struct s* ptr2;
52     for (ptr2 = *ptr;ptr2->next;ptr2 = ptr2->next);
53
54     ptr2->next = t;
55 }
```



2. Tehtävä

Tarkastellaan lyhyitä alle 127 merkin mittaisia `SmallStr` merkkijonoja. Tällainen `SmallStr` merkkijono voidaan yleisesti esittää siten, että `char`-taulukon ensimmäinen arvo on merkkijonon pituus $0 \leq n \leq 127$, jota seuraa varsinainen merkkijono. Merkkijonon lopettavaa loppunollaa ei tällöin tarvita. Toteuta näiden `SmallStr` merkkijonojen käsittelyyn alla määritellyt kolme funktiota. Voit olettaa että `malloc()`-, `calloc()` ja `realloc()`-funktiot onnistuvat aina.

Esimerkki `SmallStr` merkkijonosta:

```
1 /* Vaihtoehtoisesti vain: char* normal = "String"; */
2 char* normal = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
3 char* smallStr = {0x06, 'S', 't', 'r', 'i', 'n', 'g'};
```

```
1  typedef char* SmallStr;
2  /* Muuntaa normaalin C-kielen char* merkkijonon SmallStr merkkijonoksi
3   * Parametri str on merkkijono jonka oletetaan olevan ei-NULL
4   * Palauttaa dynaamisesti varatun SmallStr merkkijonon tai NULL arvon
5   *  mikali parametrina saatua merkkijonoa ei voida esittaa SmallStr
6   *  merkkijonona, eli jos merkkijonon pituus on yli 127 merkkia. */
7  SmallStr str2smallStr(const char* str);
8
9  /* Muuntaa SmallStr merkkijonon C-kielen char* merkkijonoksi
10  * Parametri sstr on SmallStr merkkijono jonka oletetaan olevan ei-NULL
11  * Palauttaa dynaamisesti varatun merkkijonon tai NULL arvon mikali
12  *  parametri on virheellinen, eli jos sen koko on negatiivinen. */
13 char* smallStr2str(const SmallStr sstr);
14
15 /* Tulostaa SmallStr merkkijonon stdout virtaan.
16  * Kaikki merkkijonon merkit tulostetaan virtaan ilman mitaan
17  *  ylimaaraisia merkkeja.
18  * Parametri sstr on SmallStr merkkijono jonka oletetaan olevan ei-NULL
19  *  ja virheeton. */
20 void printSmallStr(const SmallStr sstr);
```

3. Tehtävä

Mitkä 12 riviä alla oleva ohjelma tulostaa? Vastaukseksi riittää pelkkä tuloste ilman mitään perusteluja. Oikeasta vastauksesta saa ja väärästä vastauksesta menettää puoli pistettä. Vastaamatta jättäminen on ±0 pistettä.

Vihje: Loogisten operaattorien tulos on C-kielessä aina 0 tai 1.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct SwordThing_s
6  {
7      char name[32];
8      char *description;
9      struct SwordThing_s *next;
10 } SwordThing;
11
12 int main(void)
13 {
14     char *def = "Something involving sharp objects";
15
16     SwordThing *st1 = malloc(sizeof(SwordThing));
17     strcpy(st1->name, "Iaido");
18     st1->description = "Real weapons against imaginary opponents";
19
20     SwordThing sts[2];
21     strcpy(sts[0].name, "Kendo");
22     sts[0].description = "\"Weapons\" against real opponents";
23     strcpy(sts[1].name, "Naginatajutsu");
24     sts[1].description = def;
25
26     SwordThing st4 = {"Tantojutsu", def, NULL};
27     st1->next = sts;
28     sts[1].next = &sts[0];
29     sts[0].next = &sts[1];
30     sts[1].next = &st4;
31
32     printf("A: %d\n", *def == 'S');
33     printf("B: %s\n", sts->next->name);
34     printf("C: %d\n", 10 ^ 12);
35     printf("D: %d\n", sizeof(def) != sizeof(st1->description));
36     printf("E: %d\n", st4.description == sts[1].description);
37     printf("F: %d\n", sizeof(st1->description) == 41);
38     printf("G: %zu\n", sizeof(st1->name));
39     printf("H: %d\n", strcmp(st1->name + 3, st1->next->name + 3) == 0);
40     printf("I: %d\n", &(sts[1]) - &(sts[0]) == sizeof(struct SwordThing_s));
41     printf("J: %d\n", sts[1].next->next != NULL);
42     printf("K: %d\n", st1->next->next == sts + 1);
43     printf("L: %s\n", (*(sts + 1)).name);
44
45     free(st1);
46 }
```

4. Tehtävä

Siirrettäessä bittejä epäluotettavan yhteyden yli tarvitaan usein menetelmiä siirtovirheiden havaitsemiseen ja korjaamiseen. Yksi erittäin yksinkertainen tapa tähän on tiedon, eli bittien, toistaminen lähetyksen yhteydessä. Yksinkertaisin mahdollinen toistokoodaus on sellainen, jossa jokainen bitti on toistettu kerran. Tällöin 0-bitti esitetään kahdella 0-bitillä 00 ja 1-bitti taas vastaavasti kahdella 1-bitillä 11. Bittikuviot 01 ja 10 ilmaisevat siirrossa tapahtuneita virheitä.

Toteuta funktio `decode()` joka purkaa edellä kuvatun toistokoodauksen. Funktio saa parametrinaan mahdollisesti virheitä sisältävän toistokoodatun taulukon sekä taulukossa olevien koodattujen bittien määrän ja palauttaa puretun virheettömän datan sekä sen pituuden tietueessa. Virheelliset arvot datasta ohitetaan, eikä niitä sisällytetä purettuun dataan tai sen kokoon.

Voit olettaa että `malloc()`-, `calloc()` ja `realloc()`-funktiot onnistuvat aina ja voit varata muistia ennakoidusti riittävän määrän datalle jossa ei ole virheitä.

Esimerkkinä koodauksesta data { 0x3A, 0xF1 } on koodattuna { 0x0F, 0xCC, 0xFF, 0x03 }.

```
1  #include <stddef.h>
2
3  struct decodedBits
4  {
5      unsigned char *data;
6      size_t len;
7  };
8
9  /* Purkaa toistokoodatun ja mahdollisesti virheellisen datan ja palauttaa
10  *  puretun virheettoman osan datasta seka sen pituuden.
11  * Ensimmainen parametri on toistokoodattu data
12  * Toinen parametri on toistokoodatun datan koodiparien (ei bittien) maara */
13 struct decodedBits decode(unsigned char*, size_t);
```

5. Tehtävä

Ohessa on yksinkertainen ohjelma neliöiden käsittelemiseen. Ohjelma osaa lukea ja tulostaa neliöitä sekä laskea neliötaulukon neliöiden yhteispinta-alan. Valitettavasti ohjelma on "it compiles - ship it."-laatua, joten siinä on muutamia selkeitä virheitä. Etsi nämä virheet ja kerro lyhyesti miten korjaisit ne. Toimivan koodin "korjaamisesta" rikkinäiseksi voi menettää pisteitä. `main()`-funktiossa ei ole virheitä ja muun ohjelman tulee toimia `main()`-funktion kanssa ohjelmassa olevien kommenttien mukaan. `%zu` on `size_t` tietotyypin muotoilumääre ja `malloc()`, `calloc()` ja `realloc()` funktioiden oletetaan onnistuvan aina.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct Square_s
5  {
6      size_t x; // Alkupisteen X-koordinaatti
7      size_t y; // Alkupisteen Y-koordinaatti
8      size_t s; // Sivujen pituus
9  } Square;
10
11 // Lisaa nelion neliotaulukkoon
12 Square** squareAdd(Square** squares)
13 {
14     Square *new = malloc(sizeof(Square));
15     printf("Input Square (x, y, s)\n");
16     if (!scanf("%zu %zu %zu", &new->x, &new->y, &new->s))
17         return squares;
18
19     size_t size = 0;
20     for (;squares && squares[size];size++);
21
22     squares = realloc(squares, sizeof(Square) * (size + 2));
23     squares[size] = new;
24     squares[size + 1] = NULL;
25
26     return squares;
27 }
28
29 // Vapauttaa neliotaulukon kaiken muistin
30 void squaresDestruct(Square** squares)
31 {
32     for (Square** sq = squares;*sq;sq++)
33         free(*sq);
34     free(squares);
35 }
36
37 // Laskee neliotaulukon nelioiden pinta-alan
38 size_t squaresArea(Square** squares)
39 {
40     size_t area = 0;
41     size_t i = 0;
42     while (squares && squares[i])
43     {
44         area += squares[i]->x * squares[i]->y;
45         i++;
46     }
47     return area;
48 }
49
50 // Tulostaa neliotaulukon annettuun tiedostovirtaan
51 void squaresPrint(FILE* out, Square** squares)
52 {
53     for (size_t i = 0;squares && squares[i];i++)
54         printf("%zu: (%zu, %zu, %zu)\n", i,
55                squares[i]->x,squares[i]->y,squares[i]->s);
56 }
57
58 int main(void)
59 {
60     Square** sq = NULL; // Tyhja neliotaulukko
61     for (size_t i = 0;i < 2;i++)
62         sq = squareAdd(sq);
63     printf("Area: %zu\n", squaresArea(sq));
64     squaresPrint(stdout, sq);
65     squaresDestruct(sq);
66 }
```

6. Tehtävä

Tarkastellaan kokonaisluvuilla (`int`) toimivia lineaarisia funktioita jotka ottavat parametrinaan kokonaisluvun ja palauttavat kokonaisluvun, eli funktioita $f : \mathbb{Z} \to \mathbb{Z}$ jotka kaikki ovat sisäisesti muotoa $f(x) = a * x + b$, jossa a ja b ovat kokonaislukuvakioita. Lineaarisen funktion sanotaan lisäksi olevan lineaarimuunnos, mikäli vakio b on arvoltaan nolla. Lineaariset funktiot, joissa vakio b on erisuuri kuin nolla ovat affiinimuunnoksia.

Toteuta funktio `filterFunctions()`, joka poistaa taulukosta funktio-osoittimia kaikki affiinimuunnosfunktiot jättäen taulukkoon pelkästään lineaarimuunnosfunktiot. Funktio ottaa parametrinaan osoittimen funktio-osoitintaulukkon jonka viimeinen alkio on NULL ja palauttaa uuden dynaamisesti varatun funktio-osoitintaulukon jossa on jäljellä pelkät lineaarimuunnosfunktiot ja lopussa NULL arvo. Voit olettaa `malloc()`, `calloc()` ja `realloc()` funktioiden onnistuvan aina. Taulukon funktiot oletetaan sivuvaikutuksettomiksi, eli voit halutessasi kutsua taulukon funktioita useammin kuin kerran.

Vihje: Lineaari- ja affiinimuunnosfunktiot voi erottaa helposti toisistaan tarkastelemalla niiden paluuarvoa parametrin arvolla 0.

C99 Minireferenssi

Tentissä tarvittavat tietotyypit ja niiden koot

Tyyppi	Koko	Lukualue
`char`	1	-128 ... 127
`unsigned char`	1	0 ... 255
`int`	4	-2,147,483,648 ... 2,147,483,647
`long`	8	Riittävän laaja ($-2^{63} \ldots 2^{63} - 1$)
`double`	8	Riittävän laaja
`void*`	8	

Operaattoripresedenssi korkeimmasta matalimpaan

```
!   ++ --   * / %   << >>   < > <= >=   == !=   |   ^   &&   ||   ?:   = op=
```

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

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);
```