C-ohjelmoinnin peruskurssi, Tentti 20.5.2014

Lyhyt referenssi funktioista tehtäväpaperin lopussa. Paperilla on 5 tehtävää, joista useimmissa on muutama alikohta. Maksimipistemäärä on 30 pistettä.

Kirjoita vastaukset konseptipapereille seuraavasti: tehtävät 1 ja 2 yhdelle konseptiarkille, tehtävät 3 ja 4 toiselle konseptille, ja tehtävä 5 omalle konseptilleen. Merkitse konseptille tehtävän numero selkeästi, ja kirjoita selkeällä käsialalla

1. Mitä seuraava ohjelma tulostaa? Vastaukseksi riittää yksi rivi joka esittää tulosteen. (Tosin sanoen, pääosa tehtävästä on päätellä mitä muuttujat a--f saavat arvokseen) (6 p)

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

int main(void)
{
	int a = (5 > 7);
	int b; for (b = 0; b < 10; b++);
	int c = strlen("text");
	float d = 5.0 / 2;
	int e = 0xACDC & 0x00FF;
	int arr[] = {10, 100, 200}; int *p = arr; p++; int f = *p;
	printf("a: %d, b: %d, c: %d, d: %f, e: %x, f: %d\n",
		a, b, c, d, e, f);
}
```

2. Toteuta vastauspaperille seuraavat funktiot.

a) **`int calcsum(void)`**, joka kysyy käyttäjältä syötteenä 10 kokonaislukua (käyttäen scanf - funktiota), ja palauttaa niiden summan. Voit olettaa että käyttäjä syöttää pelkästään valideja kokonaislukuja. (2p)

b) **`unsigned int strlen(const char *str)`**, joka laskee merkkijonon ***str*** pituuden ja palauttaa sen. Funktion toteutuksessa **ei saa** käyttää `string.h` - otsaketta, eli valmiiksi määriteltyä samannimistä funktiota. (2 p)

c) **`void make7bit(unsigned char *s, unsigned int n)`**, joka nollaa kahdeksannen (eniten merkitsevän) bitin jokaisessa muistialueen ***s*** tavussa. Muistialueen koko on ***n*** tavua. Esimerkiksi muistialueessa olevasta luvusta 0xA3 (1010 0011) tulee funktion seurauksena 0x23 (0010 0011), mutta luku 0x7E (0111 1110) ei muutu miksikään. (2p)

3. Seuraavissa funktioissa on virheitä. Kerro kustakin virheestä virheellisen rivin numero, virheen syy lyhyesti, ja anna korjattu versio kyseisestä ohjelmarivistä. Mikäli haluat lisätä uuden rivin, kerro minkä rivin jälkeen se tulee lisätä. (Kolme kohtaa, huomaa c-kohta seuraavalla sivulla)

a) funktio, joka laskee annetun taulukon (***array***, jossa ***n*** alkiota) keskiarvon ja palauttaa sen. Koodissa on ainakin kaksi virhettä tai puutetta. (2 p)

```
1: int countAverage(int *array, unsigned int n)
2: {
3:    int avg;
4:    for (unsigned int i = 0; i < n; i++) {
5:         avg = avg + *array + i;
6:    }
7:    return avg / n;
8: }
```

b) funktio joka varaa muistista dynaamisen taulukon **size** kokonaisluvulle ja alustaa sen numeroilla jotka kasvavat 0:sta ylöspäin. Funktio palauttaa osoittimen luodun taulukon alkuun. Älä murehdi muistin vapauttamisesta: kutsuvan funktion oletetaan olevan siitä vastuussa. Koodissa on ainakin kaksi virhettä. (2 p)

```
10: #include <stdlib.h>
11:
12: int *createArray(unsigned int size)
13: {
14:   int *array = malloc(size);
15:   for (unsigned int i = 0; i < size; i++) {
16:        array[i] = i;
17:   }
18:   return *array;
19: }
```

Kääntäjä (gcc) palauttaa seuraavan varoituksen:

```
testi2.c: In function 'createArray':
testi2.c:18:2: warning: return makes pointer from integer without a cast
```

Valgrind palauttaa funktiota suorittaessa seuraavaa:

```
==25876== Invalid write of size 4
==25876==    at 0x400616: createArray (testi2.c:16)
==25876==    by 0x40069C: main (testi2.c:44)
==25876==  Address 0x51ba048 is 8 bytes inside a block of size 10 alloc'd
==25876==    at 0x4C28BED: malloc (vg_replace_malloc.c:263)
==25876==    by 0x4005F3: createArray (testi2.c:14)
==25876==    by 0x40069C: main (testi2.c:44)
```

c) Funktio joka konvertoi annetun merkkijonon ***str*** isoiksi kirjaimiksi. Alkuperäistä merkkijonoa ei muuteta, vaan uusi merkkijono varataan muistista dynaamisesti. Älä murehdi muistin vapauttamisesta: kutsuvan funktion oletetaan olevan siitä vastuussa. Koodissa on ainakin kaksi virhettä tai puutetta. (2 p)

```
20: #include <stdlib.h>
21: #include <ctype.h>
22:
23: char *createUpperCase(const char *str)
24: {
25:   char *newS = malloc(sizeof(str));
26:   for (unsigned int i = 0; i < sizeof(str); i++)
27:         newS[i] = toupper(str[i]);
28:   return newS;
29: }
```

4. Mitä seuraavat funktiot (function_A, function_B, function_C) tekevät? Älä kuvaile toiminnallisuutta rivi riviltä, vaan kuvaile lyhyesti (1-2 lausetta) mutta täsmällisesti funktion tarkoitus ja mahdollinen paluuarvo. (2 pistettä kunkin funktion oikeasta ja täsmällisestä kuvauksesta)

```
char *function_A(char *a, const char *b)
{
	char *oa = a;
	while(*b != '\0') {
		*a = *b;
		a++;
		b++;
	}
	*a = '\0';
	return oa;
}


char function_B(char *a)
{
	unsigned int c[128] = { 0 };
	while (*a) {
		c[(int)*a]++;
		a++;
	}
	unsigned int s = 0;
	for (int i = 1; i < 128; i++) {
		if (c[i] > c[s])
			s = i;
	}
	return s;
}


void function_C(const unsigned char *a, unsigned int n, char *b) {
	while (n--) {
		for (int i = 7; i >= 0 ; i--) {
			if ((*a >> i) & 1)
				*b++ = '1';
			else
				*b++ = '0';
		}
		a++;
	}
	*b = 0;
}
```

5. Toteuta alla mainitut funktiot yksinkertaista sukupuuta mallintavaan ohjelmaan. Funktiokommenteissa on annettu avuksi myös malli, jonka pohjalta voit tehdä toteutuksen (tehtäväpohjan orjallinen noudattaminen ei ole välttämätöntä, mutta person-tietorakennetta tai annettuja funktiorajapintoja ei saa muuttaa). Voit olettaa että muistin varaus onnistuu aina.

a) **newPerson**, joka luo uuden henkilön. Henkilöllä on parametrina annettu nimi, sekä (mahdollisesti) annetut vanhemmat. Mikäli vanhempi ei ole tiedossa, kyseisen parametrin arvo on NULL. Aluksi henkilöllä ei ole lapsia. Kuten tehtäväpohja ehdottaa, on suositeltavaa luoda erillinen funktio vanhempien lapsien päivitystä varten. 'children' on dynaamisesti skaalautuva taulukko, jossa on **nChildren** alkiota. Kukin taulukon alkio on **osoitin person-tietueeseen** (4 p)

b) **printDescendants**, joka tulostaa henkilön nimen, sekä kaikkien henkilön jälkeläisten nimet (lapset, lapsenlapset, lapsenlapsenlapset, ...). Saman henkilön nimi saa tulostua toistuvasti, ja tulostusjärjestyksellä ei ole väliä. (2p)

```
struct person {
      char name[30];
      struct person *father;
      struct person *mother;
      unsigned int nChildren; // number of children
      struct person **children;  // dynamic array of children pointers
};

struct person *newPerson(const char *name,
                    struct person *dad, struct person *mom)
{
      // allocate memory;

      // initialize fields based on parameters.
      // Pay attention to avoiding overflowing the reserved fields
      // (number of children is initially 0)

      // add new child for mom and dad (if they are not NULL)
      // (separate function, such as 'addChild' recommended)

      // return pointer to newly allocated person
}

void printDescendants(struct person *p)
{
      // print name of person p

      // iterate through children and print their names
      // go through their children, etc. (recursion recommended)
}
```

Mahdollisesti hyödyllisiä funktioita

Merkkijonojen käsittelyyn (määritelty string.h - otsakkeessa):

- `size_t strlen(const char *s);` palauttaa annetun merkkijonon **s** pituuden.
- `char *strcpy(char *dest, const char *src);` kopioi merkkijonon **src** paikkaan **dest**
- `char *strncpy(char *dest, const char *src, size_t n);` kopioi enintään **n** merkkiä merkkijonosta **src** merkkijonoon **dest**. Jos merkkijono on lyhyempi kuin **n**, loput tavut täytetään '\0' - merkillä.
- `char *strcat(char *dest, const char *src);` liittää merkkijonon **src** merkkijonon **dest** perään
- `char *strncat(char *dest, const char *src, size_t n);` liittää enintään **n** merkkiä merkkijonosta **src** merkkijonoon **dest**
- `int strcmp(const char *s1, const char *s2);` palauttaa 0 jos annetut merkkijonot ovat samat, erisuuri kuin 0 jos merkkijonot eroavat

Muistinhallinta (määritelty stdlib.h - otsakkeessa, memset string.h:ssa):

- `void *malloc(size_t size);` Varaa **size** tavua muistia, palauttaa osoitteen varattuun muistialueeseen
- `void *calloc(size_t nmemb, size_t size);` Varaa **nmemb** kertaa **size** tavua muistia, nollaa varatun muistialueen
- `void *realloc(void *ptr, size_t size);` Muutaa muistialueen **ptr** koon **size**:ksi, palauttaa uuden osoittimen muistialueeseen
- `void free(void *ptr);` vapauttaa annetun muistialueen
- `void *memset(void *s, int c, size_t n);` asettaa muistialueen **s**, jonka koko on **n**, kaikki tavut **c**:ksi

Merkkien käsittelyyn (määritelty ctype.h - otsakkeessa):

- `int toupper(int c);` muuta merkki isoksi kirjaimeksi
- `int tolower(int c);` muuta merkki pieneksi kirjaimeksi
- `int isalnum(int c);` onko merkki kirjain tai numero?
- `int isalpha(int c);` onko merkki kirjain?
- `int isspace(int c);` onko merkki tyhjä väli?
- `int islower(int c);` onko merkki pieni kirjain?
- `int isupper(int c);` onko merkki iso kirjain?

Muotoiltu I/O (määritelty stdio.h - otsakkeessa):

- `int printf(const char *format, ...);` tulostaa muotoiltua tulostetta annetusta merkkijonosta ja parametreista
- `int scanf(const char *format, ...);` lukee muotoiltua syötettä annettuihin osoitteisiin. Parametrit ovat siis muistiosoitteita

Muotoilumääreitä printf- ja scanf-funktioihin:

- %d: kokonaisluku
- %f: liukuluku
- %u: etumerkitön kokonaisluku
- %x: heksadesimaaliluku
- %c: merkki
- %s: merkkijono

Lukuja binäärimuodossa

0x0: 0000	0x8: 1000
0x1: 0001	0x9: 1001
0x2: 0010	0xA: 1010
0x3: 0011	0xB: 1011
0x4: 0100	0xC: 1100
0x5: 0101	0xD: 1101
0x6: 0110	0xE: 1110
0x7: 0111	0xF: 1111