C Programming, exam 16.5.2016

There is a short reference of some standard library functions at the end of the exam sheet. There are five tasks, of which most have a few subtasks. The total maximum score is 30 points.

Write your answers on separate papers in the following way: Tasks 1 and 2 should be on one sheet. Tasks 3 and 4 should be on another sheet, and task 5 should be on its own sheet. Mark the task number clearly, and use clear handwriting. Remember to write your name and student number on each sheet.

1. What does the following program print out? A sufficient answer is one line that shows the output. (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");
	int d = 1 << 1;
	int e = 0xACDC & 0x00FF;
	int arr[] = {10, 100, 200}; int *p = arr; p++; int f = *p;
	printf("a: %d, b: %d, c: %d, d: %d, e: %x, f: %d\n",
		a, b, c, d, e, f);
}
```

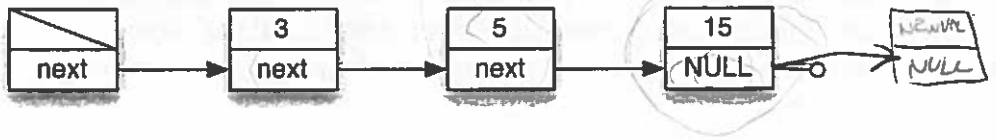
2. Implement the following functions on the answer sheet.

a) `char *mystrcat(char *dest, const char *src)` that appends string *src* at the end of string *dest*. The function returns pointer to the beginning of the combined string. (2p)

b) `int *read_numbers(void)` that reads positive integers from the user, and adds each number to the end of a dynamic array. The reading ends when user gives 0 or a negative value. The function returns a pointer to the beginning of the dynamically allocated array. (2p)

c) `void set_bit(unsigned char *buffer, unsigned int n, int bit)` that goes through *n* bytes starting from address *buffer*, and sets the bit number '*bit*' on in each byte. The most significant bit is number 7, and the least significant bit is number 0. (2p)

3a) Below is a function that adds a new integer (**newval**) at the end if linked list (**l**), but the function contains at least two errors. Tell which lines need to be corrected (or where something would need to be added), and how to correct it. In the beginning of the list there is an "empty" element that should be ignored (see graph). Don't worry about releasing memory: it will be done somewhere else in the program. You can also assume that memory allocation always succeeds. (2p)




```
10: #include <stdlib.h>
11:
12: struct list
13: {
14:     int val;
15:     struct list *next;
16: };
17:
18: void add_to_list(struct list *l, int newval)
19: {
20:     if (!l) return;
21:     while (l->next != NULL)
22:         l=l->next;
23:     l->next = malloc(sizeof(int));
24:     l->next->val = newval;
25: }
```

STRUCT LIST

l->next->next = NULL;

Valgrind outputs the following when the function is executed:

```
==10788== Invalid read of size 8
==10788==    at 0x400C06: add_to_list (tentti.c:21)
==10788==    by 0x400FD6: main (tentti.c:229)
==10788==  Address 0x51ba538 is 4 bytes after a block of size 4 alloc'd
==10788==    at 0x4C28BED: malloc (vg_replace_malloc.c:263)
==10788==    by 0x400C18: add_to_list (tentti.c:23)
==10788==    by 0x400FC2: main (tentti.c:228)
==10788==
==10788== Invalid write of size 8
==10788==    at 0x400C20: add_to_list (tentti.c:23)
==10788==    by 0x400FD6: main (tentti.c:229)
==10788==  Address 0x51ba538 is 4 bytes after a block of size 4 alloc'd
==10788==    at 0x4C28BED: malloc (vg_replace_malloc.c:263)
==10788==    by 0x400C18: add_to_list (tentti.c:23)
==10788==    by 0x400FC2: main (tentti.c:228)
```

...in addition there are couple of other similar outputs that have been dropped because of space constraints.

3b) and c) The following functions are missing some program lines. Choose the correct lines from the given options. For each missing program line, write the line number and the letter that represents the right choice.

b) Function that allocates needed amount of memory, and copies string '**src**' to the allocated space. Function returns pointer to the allocated memory buffer. (2p)

```
20: #include <stdlib.h>
21: char *allocopy_str(const char *src)
22: {
23:     char *ptr = malloc(strlen(src) + 1);
24:     char *origptr = ptr;
25:     if (ptr) {
26:         ???
27:               ???
28:         }
29:        *ptr = 0;
29:     }
30:     return origptr;
31: }
```

Line 26	Line 27
i) while (src) {	m) ptr = src;
j) while (*src) {	n) *ptr = *src;
k) while (ptr) {	o) ptr++ = src++;
l) while (*ptr) {	p) *ptr++ = *src++;

c) Function that gets a string of 8 characters as its parameter. The string consists of values '1' and '0'. The function should return the binary number presented by the string. For example: "00010001" returns 17 (i.e., 0x11 in hexadecimal format). (2p)

```
40: int read_binary(const char *bits)
41: {
42:     int val = 0;
43:     for (int i = 7; i >= 0; i--) {
44:         ???
45:             ???
46:     }
47:     return val;
48: }
```

Line 44	Line 45
q) if (*bits++ == '1')	u) val[i] = 1;
r) if (bits++ == '1')	v) val |= 1 & i;
s) if (*bits++ == 0x1)	w) val |= (1 << i);
t) if (bits++ == 0x1)	x) *val += (1 << i);

4. What do the following functions (function_A, function_B, function_C) do? Don't describe each line, but just give a short, but specific description (1 - 2 sentences) about the purpose of the function, and what does it return. If the function outputs something, describe what it shows. (2 points for each function that is described correctly and specifically)

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

unsigned int function_A(const char *a)
{
    unsigned int c = 0;
    while (*a) {
      if (*a == '\n') c++;
      a++;
    }
    return c;
}


int function_B(const char *a)
{
      FILE *f = fopen(a, "r");
      if (!f) return -1;
      int c;
      int d = 0;
      while ((c = fgetc(f)) != EOF) {
            printf("%02x ", c);
            d++;
            if (d % 8 == 0)
                  fputc('\n', stdout);
      }
      return d;
}


char function_C(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;
}
```

5. In the following we sketch a program that maintains a registry of vehicles. The registry is implemented as a linked list. For each vehicle, the registration number of max. 7 characters is stored, along with the vehicle model description, which is a free form string. A vehicle is stored in the following structure:

```
struct vehicle {
    char regnro[8];
    char *model;
    struct vehicle *next;
};
```

Function **add_vehicle** adds a new vehicle in the beginning of linked list by adding two lines of input from the user. On the first line the registration number (*regnro*) is given, and on the second line the vehicle model is given.

```
struct vehicle *add_vehicle(struct vehicle *v)
{
    struct vehicle newcar;
    newcar->next = v;
    fgets(newcar->regnro, 8, stdin);
    scanf("%s", newcar->model);
    return &newcar;
}
```

Argument *v* points to the beginning of linked list, and it can be also NULL, if the list is empty. The function returns a pointer to the beginning of the list.

a) Implementation of function **add_vehicle** is not very successful. Rewrite the function so that it works, but do not change the function call interface (i.e., the argument list or return value type). You could think, for example, whether the function should use dynamic memory somehow. You must not change the vehicle structure, but you may discuss whether the structure or function interface could be improved somehow. You can assume that the needed C-library headers a included elsewhere in the program. (2p)

b) Implement function `char *tell_model(struct vehicle *v, const char *r)` that finds registration number *r* from the linked list starting from address *v*, and returns the model of the vehicle. If the registration number is not found, the function returns NULL. (2p)

c) Implement function `void delete_all(struct vehicle *v)` that removes the linked list that starts from address v, and releases all memory used by the list. (2p)

Possibly useful functions

For operating with strings (defined in string.h header):

- `size_t strlen(const char *s);` Returns length of string **s**.
- `char *strcpy(char *dest, const char *src);` copy string **src** to address **dest**
- `char *strncpy(char *dest, const char *src, size_t n);` copy at most **n** characters from string **src** to string **dest**. If the string is shorter than **n**, the remining bytes are filled with '\0' character.
- `char *strcat(char *dest, const char *src);` concatenates string **src** after string **dest**.
- `int strcmp(const char *s1, const char *s2);` returns 0 if the given strings are same, different than 0 if the two strings differ.

Memory management (defined in stdlib.h header, memset in string.h):

- `void *malloc(size_t size);` Allocates **size** bytes of memory, returns address to the allocated memory block.
- `void *calloc(size_t nmemb, size_t size);` Allocates **nmemb** times **size** bytes of memory, zeroes the allocated memory space
- `void *realloc(void *ptr, size_t size);` Resizes memory block **ptr** to have size **size**, returns pointer to the reallocated memory space
- `void free(void *ptr);` releases the allocated memory space
- `void *memset(void *s, int c, size_t n);` sets each byte in memory block **s** with size **n** to have value **c**.

For handling characters (defined in ctype.h header):

- `int toupper(int c);` convert character to upper case letter
- `int tolower(int c);` convert character to lower case letter
- `int isalnum(int c);` is the character either alphabetical or number?
- `int isalpha(int c);` is the character alphabetical?
- `int isspace(int c);` is the character whitespace?
- `int islower(int c);` is the character a lower case letter?
- `int isupper(int c);` is the character an upper case letter?

Formatted I/O (defined in stdio.h header):

- `int printf(const char *format, ...);` prints formatted output based on the given string and parameter list
- `int scanf(const char *format, ...);` reads formatted input to the given addresses. Parameters are memory addresses.
- `char *fgets(char *s, int size, FILE *stream);` reads at most **size**-1 characters from **stream** and writes the characters to address **s**.

Format specifiers for printf and scanf functions:

- %d: integer
- %f: floating point number
- %u: unsigned integer
- %x: hexadecimal integer
- %c: character
- %s: string

Binary numbers

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