CSE3100 - System Programming — October 9, 2017

Midterm 1

This is a 120 minutes exam, commencing at 17:00 and finishing at 19:00. You may use your notes, textbook, PDF of textbooks and laptop to access your UITS-provided virtual machine. *Please read every question carefully before answering* and express your answers as completely as you can.

The computer you access should be used exclusively to access your UITS VM and to use a PDF reader. Any other applications used (in particular web-browsers) is a violation of the policy and will lead to an immediate 'F' for the course.

There are 5 questions on the exam, 4 of them (Q2-Q5) must be done on your UITS provided VM in your git repository. Start by running the usual `git pull` to retrieve the source handout. **Do not forget to commit and push your changes to the folder** `mt1`.

Read the questions attentively before starting!

Name: _____

Question	Max	Score
1	10	
2	10	
3	20	
4	20	
5	40	
Total	100	

1. Assume the following declarations

```
int i=1,j=2,k=3;
int a[] = {1,2,3};
```

For each expression below, give the value of i, j, k and the expression if it gets executed immediately after the declarations above.

	i	j	k	$expr$
`i && j`				

	i	j	k	$expr$
`i % j`				

	i	j	k	$expr$
`i / j`				

	i	j	k	$expr$
`2 * i + 1 < k`				

	i	j	k	$expr$
`++j == k++`				

	i	j	k	$expr$
`i < j || ++j == k`				

	i	j	k	$expr$
`a[i] == *(a+i)`				

	i	j	k	$expr$
`&a[k] == (a + ++j)`				

	i	j	k	$expr$
`k / j`				

	i	j	k	$expr$
`(double)k / j`				

2. Consider the classic binomial coefficient ($n \geq k$)

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

While it can be computed from the above definition, a nicer inductive definition is

$$\binom{n}{k} = \begin{cases} \frac{n}{k} \cdot \binom{n-1}{k-1} & \Leftrightarrow \quad k > 0 \\ 1 & \Leftrightarrow \quad k = 0 \end{cases}$$

Write (in the `q2` folder) an implementation of the binomial based on this second recurrence (do **not** implement and use a factorial function!). For your convenience, the function prototype is shown here:

```
int binomial(int n,int k)
```

Note that we provide a main function for testing, so your task is limited to the binomial function.

3. Consider the following task: You are to write a function

```
int length(char* s)
```

which, when given a C string (an array of character with a terminal '`\0`'), returns the length of the string. Clearly, an empty string has length `0`, while, for instance, the string "hello" has length `5`. Naturally, you **cannot** use *any* of the builtin C function (in particular, `strlen` is off-limit!). Once again, you are provided with a test program that you should leave alone and all your work should be in the `length` function. Answer the question in the folder `q3`.

4. Armed with your `length` function from the previous question, you should now implement (in folder `q4`) a function `strrev` with the following prototype:

```
char* strrev(char* s)
```

The function is meant to take a string s as input and return a heap allocated string r which is the reverse of the input s. For instance, for the input "hello", `strrev` should return a heap allocated string "olleh". Note how the string s and its reverse have the same length. Also note how the input string s is left unmolested by the call to `strrev`.

Once again, we provide a test program and the main function should be left alone. All your work should be done within `strrev`. Feel free to use your implementation of `length` as well as the C library function `malloc`. All other functions from the C library are, once again, off-limit.

5. You are expected to implement an Abstract Data Type for a *resizable stack of integers*. Your work should be carried out in the folder `q5`. The ADT is defined with the following API

```
typedef struct Stack {
   int* data;
   int  top;
   int  sz;
} Stack;

Stack* makeStack(int sz);
void freeStack(Stack* s);
void pushStack(Stack* s,int v);
int  popStack(Stack* s);
int  topStack(Stack* s);
define emptyStack(s) ((*s).top == 0)
```

The `Stack` structure defines 3 attributes:

data a pointer to the actual storage as an array of integers.

top the current top of the stack index (where the next push should deposit an element).

sz the maximum number of elements one can store in *data* without resizing.

The ADT offers 5 functions and a macro. As you might expect, the functions allow you to create a new empty stack (on the heap) with a given initial maximal size, deallocate a stack, push an element on the stack (possibly resizing it via doubling if the storage is full), popping the element at the top of the stack and peeking at the element of the stack (without popping it). The macro `emptyStack` is provided for you and simply returns true if and only if the top of the stack is `0`.

All you work is focused on implementing the five functions of the ADT. To ease your task, we do provide a main function meant to exercise all the APIs you must implement. For your convenience, the main program (that you should *not* modify) is shown below

```
int main() {
   int msz = 0,nbPush = 0;
   scanf("%d %d",&msz,&nbPush);
   Stack* s = makeStack(msz);
   for(int i=0;i<nbPush;i++) {
      pushStack(s,i);
      printf("pushed: %d\n",topStack(s));
   }
   while (!emptyStack(s))
      printf("on Stack: %3d\n", popStack(s));
   printf("freeing\n");
   freeStack(s);
   return 0;
}
```

You are expected to hand in a solution with no memory leaks and no memory bugs. Use your tools!