French Azerbaijani University
Arash Habibi
Second year
2017-2018

Advanced C Programming

Practical Work 2 : random functions and stacks

1 Stacks

A *stack* is a data structure in which the data is stacked as in a pile. This means that there are three possible operations on a stack :

— store a new element `(push)` ;

— remove the last element `(pop)` ;

— recall the value of the last element `(top)` ;

— only the top of the stack can be accessed.

Here is an example of possible operations on a stack, with the corresponding state on figure 1.

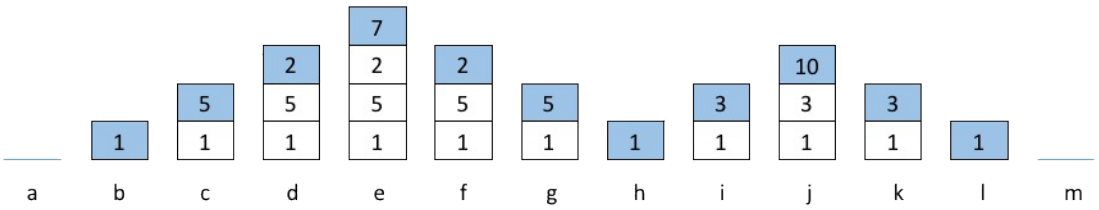

FIGURE 1 – The state of a stack. The top of the stack is represented in blue. Only the top of a stack can be accessed.

(a) let `st` be a new empty stack `// size of stack : 0, top undefined`

(b) `st = push(st,1);` `// size of stack : 1, top is 1`

(c) `st = push(st,5);` `// size of stack : 2, top is 5`

(d) `st = push(st,2);` `// size of stack : 3, top is 2`

(e) `st = push(st,7);` `// size of stack : 4, top is 7`

(f) `st = pop(st);` `// size of stack : 3, top is 2`

(g) `st = pop(st);` `// size of stack : 2, top is 5`

(h) `st = pop(st);` `// size of stack : 1, top is 1`

(i) `st = push(st,3);` `// size of stack : 2, top is 3`

(j) `st = push(st,10);` `// size of stack : 3, top is 10`

(k) `st = pop(st);` `// size of stack : 2, top is 3`

(l) `st = pop(st);` `// size of stack : 1, top is 1`

(m) `st = pop(st);` `// size of stack : 0, top undefined`

We need to make a stack data structure, with the functions operating on it. We assume that there is a maximum number of data elements N in a stack. Let us choose $N = 100$.

You may wonder why storing a data element in a stack is called *push* and removing an element is called *pop*. These names were inspired by the *plate dispenser* analogy (figure 2) : *push* to put a new plate on the top and *pop* to take a plate from the top. You cannot access the plates that are in the middle of the stack.
Sometimes, a stack is also called a LIFO : *Last In, First Out*, which means that the last element that entered the stack is the first one to get out.

FIGURE 2 – Plate dispenser

In this exercice, we choose to represent the stack by an array of $N = 100$ integers associated with one integer representing the size.

For the following questions, do not forget to test each function before you start to write the next functions. Testing a function means : calling it with different values and seeing if the function behaves as expected. The `ST_print` function is used precisely to check the contents of a stack. Try the operations illustrated on figure 1. Between each operation, print the contents of the stack (`ST_print`) to check if the operation was performed as expected.

1. Define a data structure called `struct stack` composed of :
   — an array of $N = 100$ integer elements ;
   — an integer representing the size of the stack.

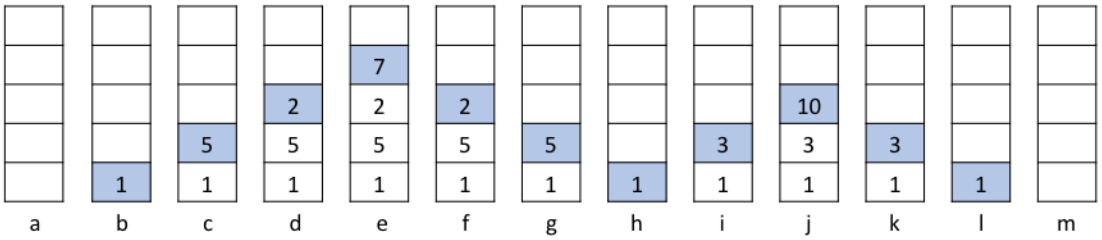

FIGURE 3 – In this exercice, we implement a stack as an array of integers + one integer representing the size.

2. Write a function with prototype `struct stack ST_new();` which returns a stack structure with size 0 ;
3. Write a function with prototype `void ST_print(struct stack);` which outputs the contents of the stack (from bottom to top, not all N) ;
4. Write a function with prototype `int ST_size(struct stack st);` which, for a given stack, returns the number of elements in the stack ;
5. Write a function with prototype `struct stack ST_push(struct stack st, int n);` which, for a given stack, adds `n` on top of the stack and returns the new stack, unless if the stack already contains N elements. In that case, the function should print an error message ;
6. Write a function with prototype `struct stack ST_pop(struct stack st);` which, for a given stack, removes the top of the stack and returns the new stack, unless if the stack is empty. In that case, the function should print an error message.
7. Write a function with prototype `int ST_top(struct stack st);` which, for a given stack, returns the value of the top element of the stack, unless if the stack is empty. In that case, the function returns an error message.

2 Random

The `random()` function returns an integer number chosen at random in the `[0,RAND_MAX]` interval.

```
#include <stdlib.h>
int random();
```

1. What is the value of `RAND_MAX` ? (You can simply use `printf` to find out) ;
2. Write a program which makes N calls of `random` and prints the results (make a `for` loop with $N = 20$ for example.) ;
3. Run the program several times. Why does it produce always the same results ? (Ask google. Keyword : pseudo-random) ;
4. Find a way to have something different each time. *Hint :*
   — You need to know the use of function `srandom` (type `man srandom` in a terminal) ;
   — Next, each process is characterized by a *pid (process identifier)* which is an integer. When a program is launched several times, it generally has a different *pid*. Type `man getpid` in a terminal.
5. Write a function in C with prototype `float random01();` which returns a floating point number chosen at random in the [0,1[interval. You can use the `random` function. But how can you process the output of `random` in order to obtain a `float` between 0 and 1 ?
6. Write a function in C with prototype `float random0n(int n);` which returns a floating point number chosen at random in the [0,n[interval. You can use the output of `random01()` ;
7. Write a function in C with prototype `int tossOfOneDice();` which returns an integer number chosen at random in {1,2,3,4,5,6} (and not in {0,1,2,3,4,5}). You can use the output of `random0n(...)`. Make sure you have read appendix A at the end of this document ;
8. Write a function in C with prototype `int tossOfTwoDice()` which simulates the toss of two six-sided dice and returns the sum of both dice. This means that the returned number should be between 2 and 12. But be careful : tossing two six-sided dice is not the same thing as tossing one twelve sided dice. First because with a 12-sided dice you can get 1 but with two 6-sided dice, you cannot have less than 2. Moreover, with a 12-sided dice, the probability of getting 2, 12 or 6 is the same. But with two 6-sided dice, there is only one way of having 2 (1+1) or 12 (6+6). But there are 5 ways of getting a 6 : (1+5, 2+4, 3+3, 4+2, 5+1). So the probability of getting 6 is much higher than the probability of getting 2 or 12.

FIGURE 4 – Two six-sided dice

FIGURE 5 – One twelve-sided dice

A Integer part

We remind that, in order to get the *integer part* of a float, you can simply use the cast operation :

```
float f;
int n;

f=5.634;
n=(int)f; // n is the integer part of f

printf("f=%f n=%d\n", f, n);
```

This code outputs : `f=5.634 n=5`.