Stacks – Implementations

In this lab, we will implement stacks in many different ways and then use the different implementations interchangeably. The interface we'll be using is the following :

```
#ifndef __STACK__H__
#define __STACK_H__

#ifdef __STATIC__STACK__

#define MAX 100

typedef struct stack {
  int tab[MAX];
  int top;
} stack_t;

#elif __DYN__STACK__

typedef struct node {
  int val;
  struct node * next;
} node_t;

typedef struct stack {
  node_t * top;
} stack_t;

#elif __FILE__STACK__

#define MAX_FLNM 100

typedef struct {
  char filename[MAX_FLNM];
  int fd; // file descriptor of the open file
  int top_off; // final offset
} stack_t;

void set_file(stack_t *, char *); // special function to determine the file to be used

#endif


void init_stack(stack_t*); // initalize the data structure
int push_stack(stack_t *, int); // push value into stack
int pop_stack(stack_t *, int *) // pops top value from stack;
int top_stack(stack_t *, int *); // returns top value of stack
void display_stack(stack_t *); // displays the contents of the stack

#endif
```

To test the implementation, we're going to use the following code :

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

int main() {
  stack_t p;

  #ifdef __FILE__STACK__
  set_file(&p, "persistentStack");
  #endif

  init_stack(&p);
  push_stack(&p, 5);
  push_stack(&p, 6);
  push_stack(&p, 7);
  display_stack(&p);

  scanf("%*c");
  int val;
  pop_stack(&p, &val);
  printf("popped value = %d\n", val);
  display_stack(&p);

  pop_stack(&p, &val);
  printf("popped value = %d\n", val);
  display_stack(&p);

  pop_stack(&p, &val);
  printf("popped value = %d\n", val);
  display_stack(&p);

  pop_stack(&p, &val); // fails
  printf("popped value = %d\n", val); // prints out the old value
  display_stack(&p);

}
```

Ex 1 - Array Implementation

Use the folowing part of the header file for this implementation

```
#define MAX 100

typedef struct stack {
  int tab[MAX];
  int top;
} stack_t;
```

A variant of this would be to dynamically allocate space for the array and reallocate it when more space is needed (double the size of ur initial table using the « realloc » method).

Ex 2 – Linked List Implementation

Use the following part of the header file for this implementation

```
typedef struct node {
  int val;
  struct node * next;
} node_t;

typedef struct stack {
  node_t * top;
} stack_t;
```

Ex 3 – File Implementation

Use the following part of the header file for this implementation

```
#define MAX_FLNM 100

typedef struct {
  char filename[MAX_FLNM];
  int fd; // file descriptor of the open file
  int top_off; // final offset
} stack_t;

void set_file(stack_t *, char *); // special function to determine the file to be used
```