Name : Sreya Sudheeran

Division : 2 (S4)

MIS : 612303175

Applications of stack

Q1. Two stacks in one array:

Create a data structure twoStacks that represents two stacks. Implementation of twoStacks should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported by twoStacks.

push1(int x) –> pushes x to first stack

push2(int x) –> pushes x to second stack

pop1() –> pops an element from first stack and return the popped element

pop2() –> pops an element from second stack and return the popped element

Implementation of twoStack should be space efficient

Ans: stack.h

```
typedef struct twoStacks{
   int top1;
   int top2;
   int *arr;
   int size1;
   int size2;
} twoStacks;

/*Global declaration*/
extern twoStacks stack;

/* Function declarations */
void init(int size1, int size2);
void push1(int x);
void push2(int x);
int pop1();
int pop2();
```

main.c

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

int main() {
   /* Initialize the two stacks with respective sizes */
   init(5, 5);
   /* Push values into the stacks */
   push1(9);
   push2(10);
   push1(3);
   push2(8);
   push2(7);
   /*Print the value of poped elements in the stacks*/
   printf("%d\n",pop1());
   printf("%d\n",pop1());
   printf("%d\n",pop2());
   printf("%d\n",pop2());
   printf("%d\n",pop2());
   return 0;
}
```

stack.c

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

/* Global variable of twoStacks */
twoStacks stack;
```

```
/* Initialize the two stack */
void init(int size1, int size2) {
   stack.size1 = size1;
   stack.size2 = size2;
   stack.top1 = -1;
   stack.top2 = size1;
   stack.arr = (int *)malloc(sizeof(int) * (size1 + size2));
   if (stack.arr == NULL) {
      return;
   }
}

/* Push an element onto the first stack */
void push1(int x) {
   /*If stack empty it will simply return */
   if (stack.top1 + 1 == stack.top2) {
      return;
   }
   stack.arr[++stack.top1] = x;
}

/* Push an element onto the second stack */
void push2(int x) {
   /*If stack empty it will simply return */
   if (stack.top1 + 1 == stack.top2) {
      return;
   }
   stack.arr[--stack.top2] = x;
}

/* Pop an element from the first stack */
```

```
int pop1() {
   /*If stack empty it will return int min*/
   if (stack.top1 == -1) {
      return INT_MIN;
   }
   return stack.arr[stack.top1--];
}

/* Pop an element from the second stack */
int pop2() {
   /*If stack empty it will return int min*/
   if (stack.top2 == stack.size1) {
      return INT_MIN;
   }
   return stack.arr[stack.top2++];
}
```

Output

```
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Two Stack> gcc -c main.c stack.c -Wall
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Two Stack> gcc main.o stack.o -o stack
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Two Stack> ./stack
3
9
7
8
10
```

Q2. Check for balanced parentheses in an expression

Given an expression string exp , write a program to examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in exp. For example, the program should print true for exp = “[()]{}{[()()]()}” and false for exp = “[(])”

Ans. stack.h

```
typedef struct stack{
   char *symbol;
   int top;
   int size;
}stack;

/*Initialise the fuctions*/
void push(stack * s, char bracket);
void init(stack * s, int size);
void pop(stack * s);
char peek(stack s);
```

main.c

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

int main(){
   /*Flag to decide if correct representation or not*/
   int ans = 1;
   char c;
   char k;
   stack open;
   init(&open,10);
   /*Takes the character value until the user presses enter*/
   while((c = getchar()) != '\n'){
```

```
    /*Pushes each character into stack if it is a opening bracket*/
    if(c == '{'|| c == '['|| c == '('){
        push(&open,c);
    }
    else{
        k = peek(open);
        if ((k == '{' && c == '}') || (k == '(' && c == ')') || (k == '[' && c == ']')) {
            /*Pops character from stack if brackets complemented*/
            pop(&open);
        }
        else {
            /*Sets flag = 0 since not correct form*/
            ans = 0;
            break;
        }

    }
}
if (open.top != -1) {
    /*If stack not empty it means imbalance in opening and closing bracket*/
    ans = 0;
}
if(ans == 1){
    /*If correct form then print True*/
    printf("True");
    return 0;
}
else{
    /*If not correct form then print False*/
    printf("False");
    return 0;
}
```

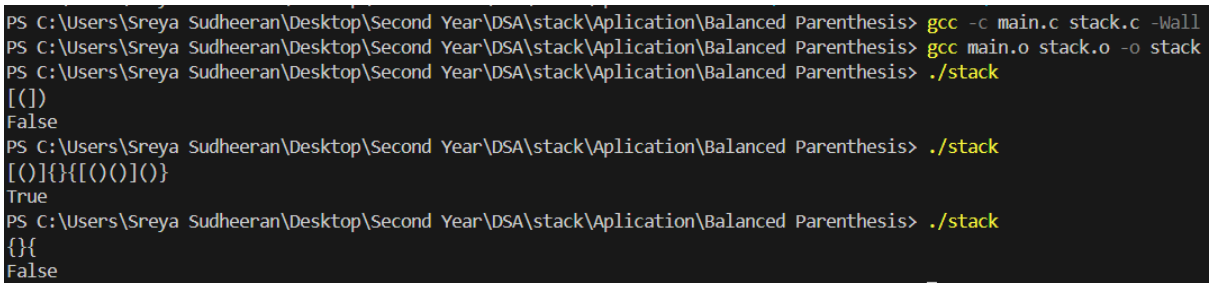
```
}
```

stack.c

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

/*Initialise the stack*/
void init(stack * s, int size){
   s->size = size;
   s->top = -1;
   s->symbol = (char*)(malloc(sizeof(char)*size));
}

/*Push the contents into stack*/
void push(stack * s, char bracket){
   /*If stack full simply return*/
   if(s->top+1 == s->size){
      return;
   }
   s->symbol[++s->top] = bracket;
   return;
}

/*Pop the contents from stack*/
void pop(stack * s){
   /*If stack empty simply return*/
   if(s->top == -1){
      return;
   }
   s->top--;
}
```

```
/*Show the content in top of the stack*/
char peek(stack s){
   /*If stack empty return character 'n'*/
   if(s.top == -1){
      return 'n';
   }
   return s.symbol[s.top];
}
```

Output



Q3. Reverse a string using stack

Given a string, reverse it using stack. For example “Data Structures” should be converted to “serutcurtS ataD”.

Ans

stack.h

```
typedef struct stack{
   int *symbol;
   int top;
   int size;
}stack;

/*Initialise the functions*/
void push(stack * s, int num);
void init(stack * s, int size);
void pop(stack * s);
int peek(stack s);
```

main.c

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

int main(){
   char c;
   stack s;
   init(&s,100);
   /*Takes the string value until the user presses enter*/
   while((c = getchar())!= '\n'){
      /*Pushes each character into stack*/
      push(&s,c);
   }
    /*Pop until the end of stack*/
   while(peek(s) != '%'){
```

```
        printf("%c",peek(s));
        /*Pops each character from stack after printing it*/
        pop(&s);
    }
    return 0;
}
```

stack.c

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

/*Initialise the stack*/
void init(stack * s, int size){
    s->size = size;
    s->top = -1;
    s->symbol = (char*)(malloc(sizeof(char)*size));
}

/*Push the contents into stack*/
void push(stack * s, char bracket){
    /*If stack full simply return*/
    if(s->top+1 == s->size){
        return;
    }
    s->symbol[++s->top] = bracket;
    return;
}

/*Pop the contents from stack*/
void pop(stack * s){
    /*If stack empty simply return*/
    if(s->top == -1){
```

```
        return;
    }
    s->top--;
}

/*Show the content in top of the stack*/
char peek(stack s){
    /*If stack empty return character '%'*/
    if(s.top == -1){
        return '%';
    }
    return s.symbol[s.top];
}
```

Output

```
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Reverse String> gcc -c main.c stack.c -Wall
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Reverse String> gcc main.o stack.o -o stack
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Reverse String> ./stack
Data Structure
erutcurtS ataD
```

Q4. Convert a base 10 integer value to base 2

Ans:

stack.h

```
typedef struct stack{
    int *symbol;
    int top;
    int size;
}stack;

/*Initialise the functions*/
void push(stack * s, int num);
void init(stack * s, int size);
void pop(stack * s);
```

```
int peek(stack s);
```

main.c

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

int main(){
   int c;
   stack s;
   init(&s,100);
   /*Takes the number which has to be converted*/
   scanf("%d",&c);
   while(c != 0){
      /*Pushes remainder after dividing with 2 into stack*/
      push(&s,c%2);
      /*Divide the number by 2*/
      c = c/2;
   }
   while(peek(s) != INT_MIN){
      printf("%d",peek(s));
      /*Pops each character from stack after printing it*/
      pop(&s);
   }
   return 0;
}
```

stack.c

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

```
/*Initialise the stack*/
void init(stack * s, int size){
   s->size = size;
   s->top = -1;
   s->symbol = (int*)(malloc(sizeof(char)*size));
}

/*Push the contents into stack*/
void push(stack * s, int num){
   /*If stack full simply return*/
   if(s->top+1 == s->size){
      return;
   }
   s->symbol[++s->top] = num;
   return;
}

/*Pop the contents from stack*/
void pop(stack * s){
   /*If stack empty simply return*/
   if(s->top == -1){
      return;
   }
   s->top--;
}

/*Show the content in top of the stack*/
int peek(stack s){
   /*If stack empty return character INT_MIN*/
   if(s.top == -1){
      return INT_MIN;
   }
```

```
    return s.symbol[s.top];
}
```

Output

```
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Binary Conversion> gcc -c main.c stack.c -Wall
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Binary Conversion> gcc main.o stack.o -o stack
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Binary Conversion> ./stack
10
1010
PS C:\Users\Sreya Sudheeran\Desktop\Second Year\DSA\stack\Aplication\Binary Conversion> ./stack
8
1000
```