Assignment 3 – Stack Applications

Name : Yashwant C. Bhosale
MIS : 612303039
SY Comp. Div 1

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.

Solution:
Two pointer approach:
1. First pointer is at the start of the array and starts filling first stack from there
2. Second pointer is at the end of the array and starts filling the second stack by decrementing the pointer

Initially:

[___ ___ ___ ___ ___ ___ ___ ___ ___ ___]

After inserting elements:

[10 20 13 23 ___ ___ 40 30 47 55]

Stack 1 Stack 2

After popping 1 element from each stack:

[10 20 13 X ___ ___ X 30 47 55]

Stack 1 Stack 2

: Top of the stack, X : Position of popped element in the stack code:

```
/* two_stack.h : Contains struct declaration and function prototypes for the stack
ADT */

typedef struct {
    int *arr;
    int top1;
    int top2;
    int size;
} stack;

void init_stack(stack *s, int size);
void push1(stack *s, int d);
void push2(stack *s, int d);
int pop1(stack *s);
int pop2(stack *s);
void print_stack_1(stack s);
void print_stack_2(stack s);

/* two_stack.c: Contains function definitions for the stack related operations */

// Function to initialize the stack
void init_stack(stack *s, int size) {
    if (!s) return;
    s -> arr = (int *) malloc(sizeof(int) * size);
    s-> top1 = -1; // as we are incrementing top1 first and then inserting element
    s-> top2 = size; // as we are decrementing top2 first and then inserting elt
    s-> size = size;
    return;
}

// Push operation for stack 1
void push1(stack *s, int d) {
    if(s->top1 >= s->top2-1 || s->top1 >= s->size-1) return; // stack 1 full
    s->arr[++s->top1] = d;
    return;
}

// Push operation for stack 2
void push2(stack *s, int d) {
    if(s->top2 <= s->top1+1 || s->top2 <= 0) return; // stack 2 full
    s->arr[--s->top2] = d;
    return;
}

// Pop operation for stack 1
int pop1(stack *s) {
    return s->arr[s->top1--]; // decrement top for stack 1
}

// Pop operation for stack 2
int pop2(stack *s) {
    return s->arr[s->top2++]; // increment top for stack 2
}
```

```c
// Function to print stack 1
void print_stack_1(stack s) {
    printf("[\t");
    for(int i = 0; i <= s.top1; i++) {
        printf("%d\t", s.arr[i]);
    }
    printf("\b<-top\t");
    printf("]\n");
    return;
}

// Function to print stack 2
void print_stack_2(stack s) {
    printf("[\t");
    for(int i = s.size-1; i >= s.top2; i--) {
        printf("%d\t", s.arr[i]);
    }
    printf("\b<-top\t");
    printf("]\n");
    return;
}

/* main.c: Contains main flow of the program */

#include <stdio.h>
#include <stdlib.h>
#include "two_stack.h"

int main() {
    stack s;
    int size = 10;
    init_stack(&s, size);

    push1(&s, 10);
    push1(&s, 20);
    push1(&s, 13);
    push1(&s, 23);
    printf("Stack 1: ");
    print_stack_1(s);

    push2(&s, 55);
    push2(&s, 47);
    push2(&s, 30);
    push2(&s, 40);
    printf("Stack 2: ");
    print_stack_2(s);

    printf("popping from stack 2 : %d\n", pop2(&s));
    printf("Stack 2: ");
    print_stack_2(s);
    free(s.arr);
    return 0;
}
```

Output:

```
  $  gcc -Wall main.c two_stack.c
  $  ./a.out
Stack 1: [      10      20      13      23      <-top    ]
Stack 2: [      55      47      30      40      <-top    ]
popping from stack 2 : 40
Stack 2: [      55      47      30      <-top    ]
```

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 = “[(])”

Solution:

1. If we encounter an opening parenthesis **push it to the stack.**
2. If we encounter a closing parenthesis, then **pop the opening parenthesis from the stack,** and check if it matches with the corresponding closing parenthesis **if it doesn’t we return false**
3. Finally, we check if the stack is empty. If it is empty then everything went well and parenthesis expression is valid, else it is invalid.

Code:
/* stack.h: Struct declaration and function prototypes for stack related operations */

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

void init_stack(stack *s, int size);
void push(stack *s, char c);
char pop(stack *s);
char peek(stack s);
short int is_empty(stack s);

/* stack.c: Function definitions for stack related functions */
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

// Function to initialize stack ADT
void init_stack(stack *s, int size) {
    if(!s) return;
    s->arr = (char *) malloc(sizeof(char) * size);
    s->size = size;
    s->top = -1;
```

```c
    return;
}
// Function to push an element to the stack
void push(stack *s, char c) {
    if(s->top >= s->size-1) // if top == size-1 then stack is full
        return;
    s->arr[++s->top] = c;
    return;
}

// Function to pop an element from the stack
char pop(stack *s) {
    return s->arr[s->top--];
}

// Function to view the top element of the stack
char peek(stack s) {
    return s.arr[s.top];
}

// Function to check if the stack is empty
short int is_empty(stack s) {
    return s.top == -1;
}

/* main.c: Contains the main flow of the programs and essential functions */

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

// Function to check if the entered character is an opening parenthesis of any type
short int is_opening_paranthesis(char c) {
    return (c=='(' || c=='[' || c=='{');
}

// Function to check if the entered character is an closing parenthesis of any type
short int is_closing_paranthesis(char c) {
    return (c==')' || c==']' || c=='}');
}

// Function to match the parenthesis based on type of the parenthesis
char match(char c) {
    switch(c) {
        case '(':
            return ')';
        case '[':
            return ']';
        case '{':
            return '}';
        default:
            break;
    }
    return 0;
}
```

```c
// Main function to validate the parenthesis
short int valid_paranthesis(char *s) {
    int len = strlen(s);

    stack char_stack;
    init_stack(&char_stack, len);
    for(int i = 0;i < len; i++) {
        /* If the character is opening parenthesis push it to the stack */
        if(is_opening_paranthesis(s[i]))
            push(&char_stack, s[i]);

        /* If the character is closing parenthesis, match it with the popped
           character from the stack and return if it doesn't match */
        if(is_closing_paranthesis(s[i]) && match(pop(&char_stack)) != s[i])
            return 0;

    }

    /* Finally check if the stack is empty, if it is then it means that all
       parenthesis in the expression are balanced */
    if(is_empty(char_stack))
        return 1;
    else
        return 0;
}

int main() {
    char str[64];
    printf("Enter string: ");
    scanf("%s", str);
    if(valid_paranthesis(str))
        printf("True\n");
    else
        printf("False\n");
    return 0;
}

/* test_cases.txt: File containing test cases for the program */
()
()[]
{[()]}
([)]
(((
[[[[]]]]
{[]}
{[(])}
[]
{[{}()]}
exit
```

Output:

```
  $  gcc -Wall main.c stack.c
  $  cat test_cases.txt | ./a.out
Enter string or enter 'exit' to exit the program:
Expression: ()
Result: True

Enter string or enter 'exit' to exit the program:
Expression: ()[]
Result: True

Enter string or enter 'exit' to exit the program:
Expression: {[()]}
Result: True

Enter string or enter 'exit' to exit the program:
Expression: ([)]
Result: False

Enter string or enter 'exit' to exit the program:
Expression: (((
Result: False

Enter string or enter 'exit' to exit the program:
Expression: [[[[]]]]
Result: True

Enter string or enter 'exit' to exit the program:
Expression: {[]}
Result: True

Enter string or enter 'exit' to exit the program:
Expression: {[(])}
Result: False

Enter string or enter 'exit' to exit the program:
Expression: []
Result: True

Enter string or enter 'exit' to exit the program:
Expression: {[{}()]}
Result: True
```

3. Reverse a string using stack

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

Solution:
1. Push all the characters in the stack
2. Pop characters from the stack and insert them into the string from the beginning.

Code:

```
/* stack.h: Struct declaration and function prototypes for stack related operations
*/

typedef struct {
    char *arr;
    int size;
    int top;
} stack;

void init_stack(stack *s, int size);
void push(stack *s, char c);
char pop(stack *s);
char peek(stack s);
short int is_empty(stack s);

/* stack.c: Function definitions for stack related functions */

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

// Function to initialize stack ADT
void init_stack(stack *s, int size) {
    if(!s) return;
    s->arr = (char *) malloc(sizeof(char) * size);
    s->size = size;
    s->top = -1;
    return;
}
// Function to push an element to the stack
void push(stack *s, char c) {
    if(s->top >= s->size-1) // if top == size-1 then stack is full
        return;
    s->arr[++s->top] = c;
    return;
}

// Function to pop an element from the stack
char pop(stack *s) {
    return s->arr[s->top--];
}

// Function to view the top element of the stack
char peek(stack s) {
    return s.arr[s.top];
}
```

```c
// Function to check if the stack is empty
short int is_empty(stack s) {
    return s.top == -1;
}

/* main.c: Contains main flow of the programs and essential functions */
#include <stdio.h>
#include <string.h>
#include "stack.h"

// Function to reverse the string
void reverse(char *s) {
    stack char_stack;
    int len = strlen(s);
    init_stack(&char_stack, len);

    // Push all characters to the stack
    for(int i = 0; i < len; i++)
        push(&char_stack, s[i]);

    // Pop all characters into the string from beginning
    for(int j = 0; j < len; j++)
        s[j] = pop(&char_stack);
    return;
}

// Function to read strings including white space character
void read_string(char *str) {
    int i = 0;

    // EOF because we will be reading a file of testcases
    while((str[i] = getchar()) != '\n' && str[i] != '\0' && str[i] != EOF)
        i++;
    str[i] = '\0';
    return;
}

int main() {
    char str[64];
    while(1){
        printf("Enter string or 'exit' to Exit: \n");
        read_string(str);
        if(strcmp(str, "exit") == 0)
            break;
        printf("Entered string: %s\n", str);
        reverse(str);
        printf("After reversing: %s\n", str);
        printf("\n");
    }
    return 0;
}
```

/* test_cases.txt: File containing test cases for the program */

```
Data Structures
binaryTree
LinkedLIST
HashMAP
GraphAlgorithm
queueingSYSTEM
DepthFirstSearch
dataSTRUCTURE
ArrayIndexOutOfBounds
helloWorld
```

Output:

```
  $  gcc -Wall main.c stack.c
  $  cat test_cases.txt | ./a.out
Enter string or 'exit' to Exit:
Entered string: Data structures
After reversing: serutcurts ataD

Enter string or 'exit' to Exit:
Entered string: binaryTree
After reversing: eerTyranib

Enter string or 'exit' to Exit:
Entered string: LinkedLIST
After reversing: TSILdekniL

Enter string or 'exit' to Exit:
Entered string: HashMAP
After reversing: PAMhsaH

Enter string or 'exit' to Exit:
Entered string: GraphAlgorithm
After reversing: mhtiroglAhparG

Enter string or 'exit' to Exit:
Entered string: queueingSYSTEM
After reversing: METSYSgnieueuq

Enter string or 'exit' to Exit:
Entered string: DepthFirstSearch
After reversing: hcraeStsriFhtpeD

Enter string or 'exit' to Exit:
Entered string: dataSTRUCTURE
After reversing: ERUTCURTSatad

Enter string or 'exit' to Exit:
Entered string: ArrayIndexOutOfBounds
After reversing: sdnuoBfOtuOxednIyarrA

Enter string or 'exit' to Exit:
Entered string: helloWorld
After reversing: dlroWolleh
```

Q4. Convert a base 10 integer value to base 2

Solution:
1. Push remainder at each step to the stack
2. Finally pop it into the string and return binary representation string.

Code:

```c
/* stack.h: Struct declaration and function prototypes for stack related operations
*/

typedef struct {
    char *arr;
    int size;
    int top;
} stack;

void init_stack(stack *s, int size);
void push(stack *s, char c);
char pop(stack *s);
char peek(stack s);
short int is_empty(stack s);

/* stack.c: Function definitions for stack related functions */

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

// Function to initialize stack ADT
void init_stack(stack *s, int size) {
    if(!s) return;
    s->arr = (char *) malloc(sizeof(char) * size);
    s->size = size;
    s->top = -1;
    return;
}
// Function to push an element to the stack
void push(stack *s, char c) {
    if(s->top >= s->size-1) // if top == size-1 then stack is full
        return;
    s->arr[++s->top] = c;
    return;
}

// Function to pop an element from the stack
char pop(stack *s) {
    return s->arr[s->top--];
}

// Function to view the top element of the stack
char peek(stack s) {
    return s.arr[s.top];
}
```

```
// Function to check if the stack is empty
short int is_empty(stack s) {
    return s.top == -1;
}

/* main.c: contains main flow of the program and essential functions */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "stack.h"

char *to_binary(int num) {
    /* Maximum length that the binary representation string of a decimal number can
       have is [n/2] or floor(n/2).*/
    int len = num <= 1 ? 2 : ((num/2)+1); // +1 for null character
    int i, top;
    char *result = (char *) malloc(sizeof(char) * len);
    stack bin_stack;
    init_stack(&bin_stack, len);
    while(num) {
        push(&bin_stack, (num%2+'0')); // push ascii value of digits to string
        num = num/2;
    }
    top = bin_stack.top;
    for(i = 0; i <= top; i++) {
        result[i] = pop(&bin_stack); // pop digits in result string
    }
    result[i] = '\0';
    return result;
}

int main() {
    int num;
    char *bin = NULL;
    printf("Enter Number: ");
    scanf("%d", &num);
    if(num >= 0){
        bin = to_binary(num);
        printf("Binary: %s\n", bin);
        free(bin);
    }
    return 0;
}

/* test_cases.txt: contains test cases for the program */
1
2
7
15
31
64
128
255
1024
2047
```

Output:

```
  $  gcc -Wall main.c stack.c
  $  cat test_cases.txt | ./a.out
Enter Number or 'Ctrl + C' to Exit:
Entered Number: 1
Binary: 1

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 2
Binary: 10

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 7
Binary: 111

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 15
Binary: 1111

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 31
Binary: 11111

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 64
Binary: 1000000

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 128
Binary: 10000000

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 255
Binary: 11111111

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 1024
Binary: 10000000000

Enter Number or 'Ctrl + C' to Exit:
Entered Number: 2047
Binary: 11111111111
```