Karnatak Law Society’s

GOGTE INSTITUTE OF TECHNOLOGY

Udyambag Belagavi -590008

Karnataka, India.



A Course Project Report on

**STACK ADT**

Submitted for the requirements of 3rd semester B.E. in CSE

for **“Datastructures with C (18CS32)”**

**Submitted by**

|  |  |
| --- | --- |
| **NAME** | **USN** |
| **Shradha Patil** | **2GI20CS144** |
| **Shivani shivapur** | **2GI20CS141** |
| **Shrushti gani** | **2GI20CS150** |
| **Siddesh gurunath naik** | **2GI20CS152** |

**Under the guidance of**

**Prof. Gouri Khadabadi**

**Asst. Prof., Dept. of CSE**

**Academic Year 2021 - 22(Odd semester)**

**Table of contents**

**S.No. Content Page No.**

1. Title of the project 3
2. Statement of the Problem 3
3. Objectives and scope of the project 3
4. Hardware and software to be used 3
5. Methodology / Algorithm / Pseudocode 4
6. Screenshots 19
7. Conclusion 20
8. References 20

**TITLE OF THE PROJECT:**

Stack ADT

**PROBLEM STATEMENT:**

Develop and execute C program to implement stack ADT.

**OBJECTIVES AND SCOPE OF THE PROJECT:**

1.Understand the Stack Data Structure

2. Understand basic operations on stack

3.Understand the method of defining Stack ADT and Implement the basic operations

4.Understand how to import ADT in a program

**HARDWARE AND SOFTWARE TO BE USED**

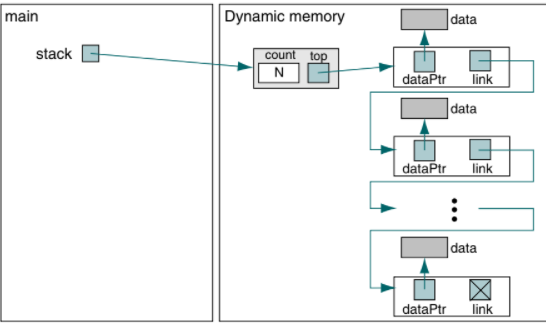
Hardware: A computer used to run the program

Software: Windows OS, Code Blocks 20.03 (with GCC compiler)

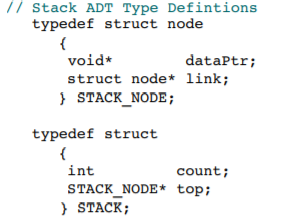
**Methodology / Algorithm / Pseudocode**

The stack ADT implementation in C is straightforward. Rather than store data in each node, we store a pointer to the data. It is the application program’s responsibility to allocate memory for the data and pass the address to the stack ADT.

The stack head structure also contains only two elements, a pointer to the top of the stack and a count of the number of entries currently in the stack.

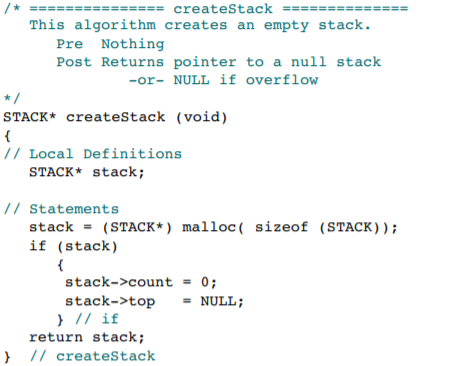


Stack ADT Definitions



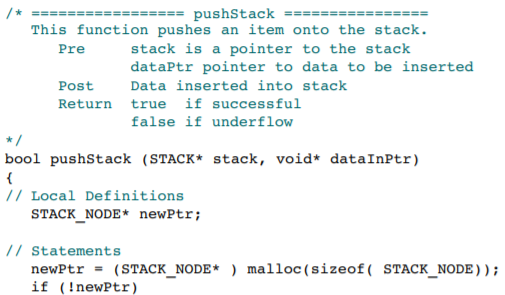
Create Stack

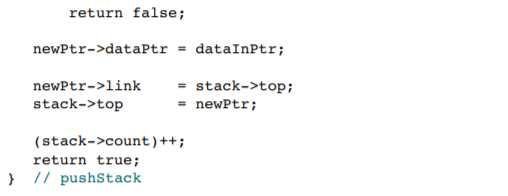
Create stack allocates a stack head node, initializes the top pointer to null, and zeros the count field. The address of the node in the dynamic memory is then returned to the caller.



Push Stack

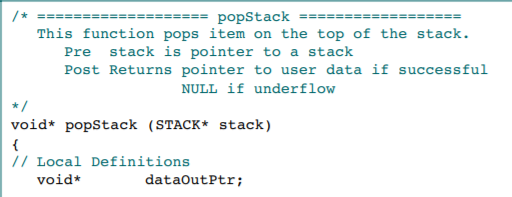
we allocate memory from the heap using malloc. Once the memory is allocated, we simply assign the data pointer to the node and then set the link to point to the node currently indicated as the stack top.

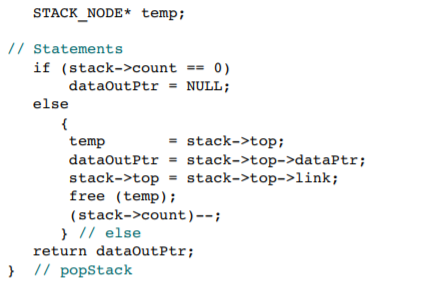




Pop Stack

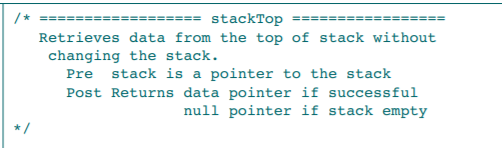
Pop stack returns the data in the node at the top of the stack. It then deletes and recycles the node. After the count is adjusted by subtracting 1, the function returns to the caller. Note the way underflow is reported. In statement 15 we set the data pointer to NULL. If the stack is empty, when we return the data pointer in statement 24 we return NULL.

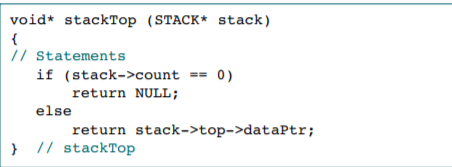




Stack Top

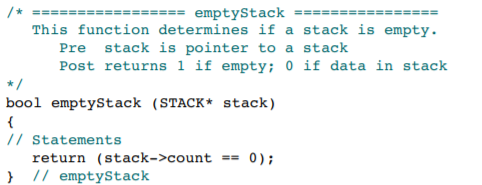
The stack top function returns the data at the top of the stack without deleting the top node.

****

****

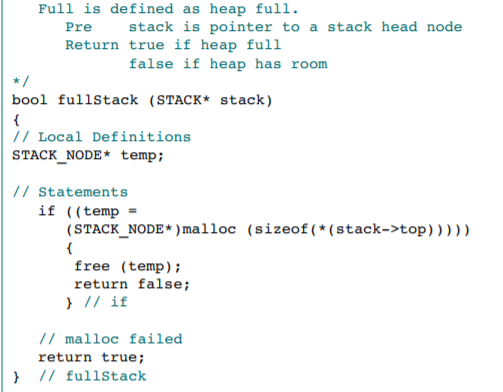
Empty Stack

Because the calling function has no access to the data structure, it cannot determine if there are data in the stack without actually trying to retrieve them.



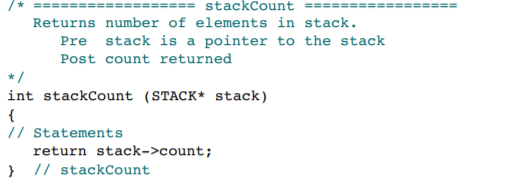
Full Stack

Full stack is one of the most complex of the supporting functions. There is no straightforward way to tell if the next memory allocation is going to succeed or fail. All we can do is try it. But by trying to make an allocation, we use up part of the heap. Therefore, after allocating space for a node, we immediately free it so that it will be there when the program requests memory.



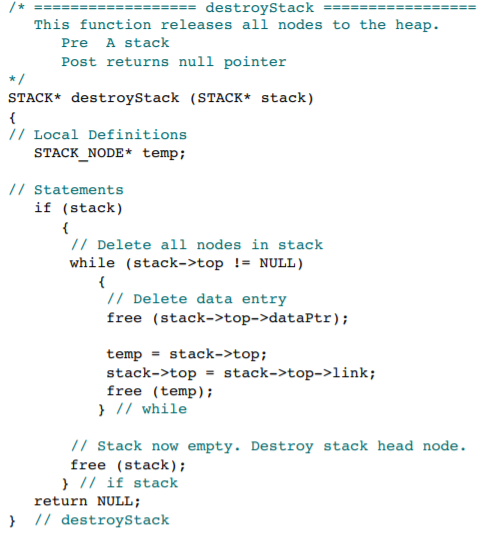
Stack Count

Stack count returns the number of items in the stack. Because this count is stored in the stack head node



Destroy Stack

Destroy stack deletes the nodes in a stack and returns a null pointer. It is the user’s responsibility to set the stack pointer in the calling area to NULL by assigning the return value to the local stack pointer.

****

**SOURCE CODE:**

**H**eader File:

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

void \*dataPtr;

struct node\*next;

}DNODE;

typedef struct {

int count;

struct node\*top;

}HNODE;

HNODE\* createStack(){

HNODE\* head=(HNODE \*)malloc(sizeof(HNODE));

if(head)

{

head->count=0;

head->top=NULL;

}

return head;

}

bool isEmpty(HNODE \*head)

{

if(head->count)

return false;

return true;

}

bool isFull()

{

DNODE \*temp=(DNODE \*)malloc(sizeof(DNODE));

if(temp){

return false;

}

else

return true;

}

bool push(HNODE\*\* head,void \*dataPtr)

{

if(!isFull())

{

DNODE \*temp=(DNODE \*)malloc(sizeof(DNODE));

//temp->dataPtr=(int\*) malloc(sizeof(int));

//\*((int\*)temp->dataPtr) = \*((int\*) dataPtr);

temp->dataPtr = dataPtr;

temp->next=(\*head)->top;

(\*head)->top=temp;

(\*head)->count++;

return true;

}

else

return false;

}

bool pop(HNODE \*\*head,void\*\*var){

if(!isEmpty(\*head))

{

DNODE \*temp;

temp=(\*head)->top;

printf("%d",\*((int\*)temp->dataPtr));

\*var=temp->dataPtr;

(\*head)->top=temp->next;

(\*head)->count--;

free(temp);

return true;

}

else

return false;

}

bool display(HNODE \*\*head,void \*\*var)

{

if((\*head)->top)

{

\*var=(\*head)->top->dataPtr;

(\*head)->top=(\*head)->top->next;

return true;

}

return false;

}

bool stackTop(HNODE \*head,void\*\*var){

if(!isEmpty(head)){

\*var=head->top->dataPtr;

return true;

}

return false;

}

HNODE\* destroyStack(HNODE\*head){

DNODE \*temp;

if(head->count){

while(!head->top){

free(head->top->dataPtr);

temp=head->top;

free(temp);

head->top=head->top->next;

}

}

free(head);

return NULL;

}

Driver Code:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include "stackdfn.h"

/\*typedef struct node{

void \*dataPtr;

struct node\*next;

}DNODE;

typedef struct {

int count;

struct node\*top;

}HNODE;\*/

int main()

{

HNODE \*head;

HNODE\*temp=(DNODE\*)malloc(sizeof(HNODE));

int ch,\*item;

void \*var;

while(1)

{

printf("\n1:create stack\t2:push\t3:pop \t4:display\t 5:display stack top\t6:destroy stack\t7:exit\n");

printf("enter the choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:if(!(head=createStack()))

printf("malloc failure\n");break;

case 2:if((item = (int\*)malloc(sizeof(int))))

{

printf("\n enter the element to be pushed\n");

scanf("%d",item);

push(&head,item);

}

// printf("%p\n",head->top);

break;

case 3:if(pop(&head,&var))

{

// printf("%p",head->top);

printf("\nthe popped item is %d\n",\*((int\*)var));

}

else

printf("\nstack underflow\n");

break;

case 4:

temp->top=head->top;

if(!temp->top)

{

printf("\nstack is empty");

}

else

{

while(display(&temp,&var)){

printf("%d\t",\*((int\*)var));

}

}

break;

case 5:if(stackTop(head,&var)){

printf("\nthe stack top is%d\n",\*((int\*)var));

}

else

printf("\nstack is empty\n");break;

case 6:head=destroyStack(head);break;

case 7:exit(0);

default:printf("\ninvalid choice\n");

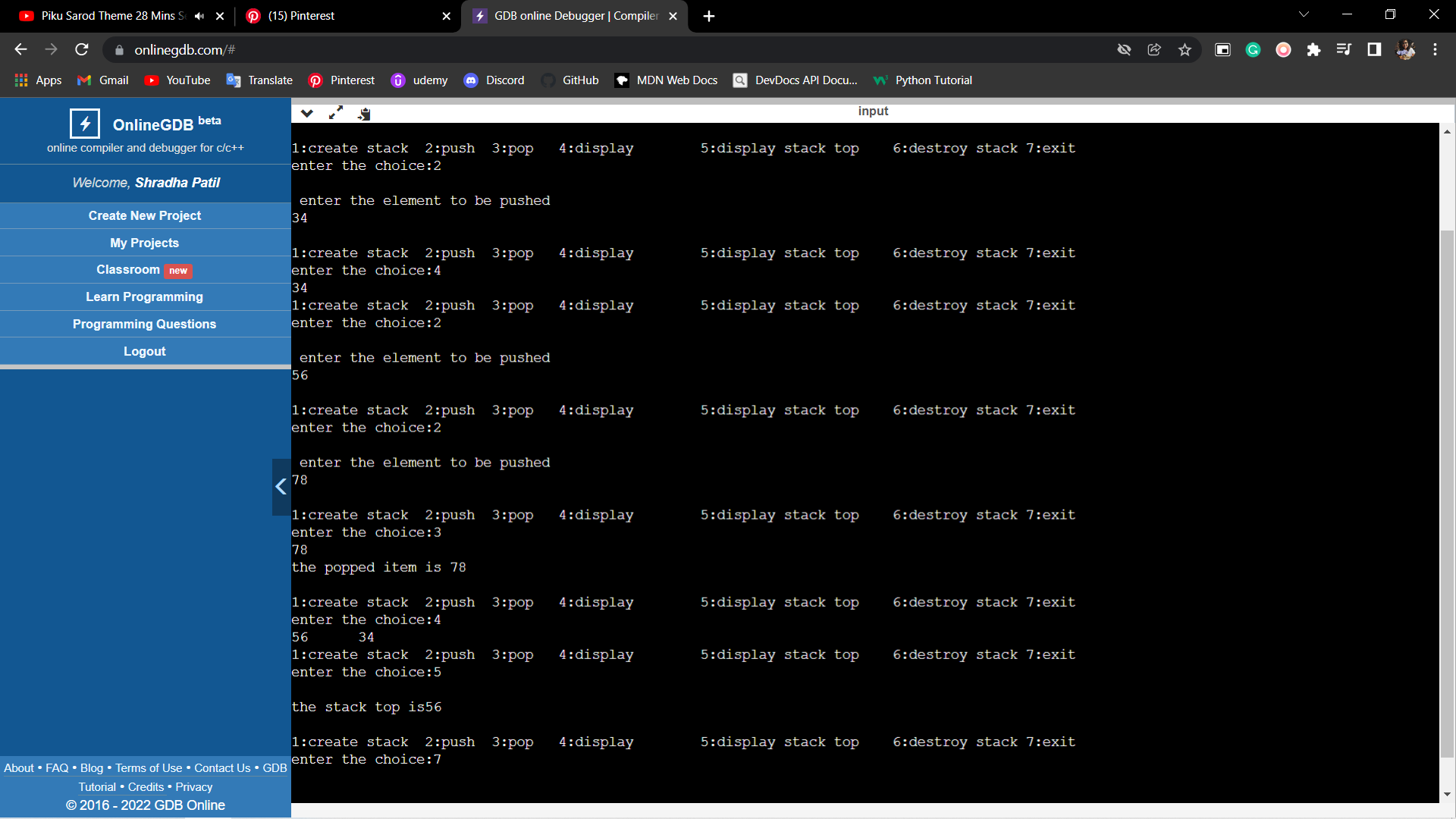
}

}

return 0;

}

**OUTPUT:**

****

**CONCLUSION:**

The stack is the simplest data structure and easier to implement as a program. It used the LIFO (last in, first out) approach which means the element entered last is the one that is removed first. This is because stack uses only one end to add (push) and remove (pop) elements.

The stack data structure has many uses in software programming. The prominent one among them is expression evaluations. Expression evaluation also includes converting the expression from infix to postfix or prefix. It also involves evaluating the expression to produce the final result.

**REFERENCES:**

Richard.F.Gilberg, Behrouz.A. Forouzan, Data Structures: A Pseudocode Approach with C,

Cengage Learning, 2nd edition 2007 and onwards

2. Horowitz, Sahni, Anderson-Freed, Fundamentals of Data Structures in C, Universities Press, 2nd

Edition, 2007 and onwards.

**-**