**Stack data structure(Introduction and Program)**

Stack is a linear data structure which follow a particular order in which the operation are performed .

The order may be LIFO or FILO.

Mainly the following three basic operation are performed in the stack:

Push: Adds an item in the stack. If the stack is full then it is said to be an overflow condition.

POP: Removes an item from the stack the items are popped in the reversed order in which they are pushed if the stack is empty, then it is said to be an Underflow condition.

Peek or Top: Returns top element of the stack.

isEmpty: Returns true if stack is empty, else false.

How to understand a stack practically?

There are many real-life example of a stack. Consider the simple example of plates stacked over one another in a canteen. The plate which is the top is the first one to be removed, i.e the plates which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order.

Time Complexities of the operation on stack:

Push(), pop(), isEmpty(), and peek() all take O(1) time. We do not run any loop in any of these operation.

Application of the stack:

Balancing of symbols

Infix to Postfix/Prefix conversion

Redo-Undo features at many places like editor, Photoshop.

Forward and backward feature in web browser

Used in many algorithm like Tower of Hanoi, tree traversals, stock span problem, histogram problem.

Backtracking is one of the algorithm designing technique. Some example of back tracking are knight-Tour problem, N-Queen problem, find your way through maze and game like chess or checker in all this problems we dive into someway if that way is not efficient we come back to the previous state for that purpose we need stack.

In Graph Algorithm like Topological Sorting Connected Components

In Memory management any modern computer uses stack as the primary-management for a running purpose. Each program that is running in a computer system has its own memory allocations.

String reversal is also a another application of stack. Here one by one each character get inserted into the stack. So the first character of the sting is on the bottom of the stack and the last element of string is on the top of stack. After Performing the pop operations on the stack we get string reverse order.

Implementation:

There are two ways to implement a stack:

1. Using array
2. Using linked list

Implementing stack using Arrays

// C program for array implementation of stack

#include<stdio.h>

#include<stdflib.h>

#include<limit.h>

//A structure to represent a stack

Struct Stack

{

int top;

unsigned capacity;

int \*array;

}

// function to create a stack of given capacity. It initializes size of stack as 0

struct Stack\* createStack(unsigned capacity)

{

struct Stack\* stack = (struct Stack\*) malloc(sizeof (struct Stack));

stack->capacity = capacity = capacity;

stack->top = -1;

stack->array = (int\* ) malloc(stack->capacity \* sizeof(int ));

return stack;

}

// Stack is full when top is equal to the last index

int isFull (struct Stack\* stack)

{ return stack->top == stack->capacity -1 }

// Stack is empty when top is equal -1

int isEmpty ( struct Stack \* stack)

{ return stack->top == -1; }

// Function to add an item to stack. It increases top by 1

Void push(struct Stack\* stack , int item)

{

if (isFull(stack)

return;

stack->array[++stack->top] = item;

printf(“%d pushed to stack\n”,item);

}

// Function to remove an item from stack. It decreases top by 1

int pop(struct Stack\* stack)

{

if(isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top--];

}

// Function to return the top from stack without removing it

Int peek(struct Stack\* stack)

{

If(isEmpty(stack))

Return INT\_MIN;

Return stack->array[stack->top];

}

//Driver program to test above functions

Int main()

{

Struct Stack\* stack = createStack(100);

Push(Stack, 10);

Push(Stack, 20);

Push(Stack, 30);

Printf(“%d popped from stack\n”, pop(stack));

Return 0;

}

Output:

10 pushed into stack

20 pushed into stack

30 pushed into stack

30 popped from stack

Top element is: 20

Elements present in stack : 20 10

Pros: Easy to implement. Memory is saved as pointer are not involved.

Cons: It is not dynamic. It doesn’t grow and shrink depending on the needs at runtime.

Implementation Stack using Linked List:

// C program for linked list implementation of stack

#include<limit.h>

#include<stdio.h>

#include<stdlib.h>

//A structure to represent a stack

Struct StackNode {

Int data;

Struct StackNode\* next;

};

Struct StackNode\* newNode(int data)

{

Struct StackNode\* stackNode = (struct StackNode\*) malloc(sizeof(struct StackNode));

stackNode->data = data;

stackNode->next = NULL;

return stackNode;

}

Int isEmpty(struct StackNode\* root)

{

Return !root;

}

Void push(struct StackNode\*\* root, int data)

{

Struct StackNode\* stackNode = newNode(data);

stackNode->next = \*root;

\*root = stackNode;

Printf(“%d pushed to stack\n”,data);

}

Int pop(struct StackNode\*\* root)

{

If(isEmpty (\*root))

Return INT\_MIN;

Struct StackNode \*temp = \*temp;

\*root = (\*root)->next;

Int popped = temp->data;

Free(temp);

Return popped;

}

Int peek(struct StackNode\* root)

{

If(isEmpty(root))

Return INT\_MIN;

Return root->data;

}

Int main()

{

Struct StackNode\* root = NULL;

Push(&root, 10);

Push(&root,20);

Push(&root, 30);

Printf(“%d popped from stack\n”, pop(&root));

Printf(“Top element is %d\n”, peek(root));

Return 0;

}

Output:

10 pushed to stack

20 pushed to stack

30 pushed to stack

30 popped from stack

Top element is 20

Element present in stack : 20 10

Pros: The linked list implementation of stack can grow and shrink according to the need at runtime.

Cons: Requires extra memory due to involvement of the pointers.

We will cover the implementation of application of stack in separate posts.

References:

<https://www.geeksforgeeks.org/stack-data-structure-introduction-program/>