Stack Implementation

The operation in implementation of stack through program as follows:

1. We have variables such as $top, size\ and\ *s$. The variable top point to top of the stack, $size$ is the stack size and *s is to create the stack in the heap memory.

```
typedef struct Stack
{
    int top;
    int size;
    int *s;

} Stack;
```

2. *Creation of Stack*

```
void create(Stack *st)
{
  cout << "Enter the size of the stack" << endl;
  cin >> st->size;
  st->top = -1;
  st->s = (int *)malloc(st->size * sizeof(int));
}
```

```
int main()
{


Stack stck;
create(&stck); //Call by reference

}
```

$st \rightarrow size$, here we take the size from user to create a dynamic array to create a stack.

$st \rightarrow top = -1$, as $top = -1$ repesents stack is empty.

$st \rightarrow s = (int\ *)malloc\left(st \rightarrow size * sizeof(int)\right)$ is done to create array in heap memory.

As we are going to show stack functionality through array.

3.***<u>Push Operation</u>***

```cpp
void push(Stack *st, int x)
{
    if (st->top == st->size - 1)
    {
        cout << "Stack Overflow" << endl;
    }
    else
    {
        st->top++;
        st->s[st->top] = x;
    }
}

int main()
{

   int x;
   cin >> x;
   push(&stck, x); //Call by reference

}
```

first we have to check that the stack is full or not.

if $top == size - 1$ *, i.e. if size is* 4 *then we will have*

$a[0], a[1], a[2], a[3]$ *, hence size will increment from* 0 *to* 3.

then $top = -1$ *, when stack is empty,*

1*st element push* $\rightarrow$ $top = top + 1 = 0$.

2*nd element push* $\rightarrow$ $top = top + 1 = 1$.

3*rd element push* $\rightarrow$ $top = top + 1 = 2$.

4*rth element push* $\rightarrow$ $top = top + 1 = 3$.

$\boldsymbol{if\,(top == size - 1)\ then\ Stack\ is\ Full.}$

$\boldsymbol{if\ not\ full\ ,then\ we\ increment\ top = top + 1.}$

$\boldsymbol{and\ push\ the\ element\ at\ top\ of\ the\ stack =}$

$\boldsymbol{s[top] = element.}$

4. ***Pop Operation***

```
int pop(Stack *st)
{

    if (st->top == -1)
    {
        cout << "Stack Underflow" << endl;
    }

    return st->s[st->top--];
}

int main()
{

    Stack stck;
    pop(&stck); // call by reference
}
```

if top = -1 hence stack is empty therefore Underflow else we decrement the top . As we decrement the top and set to the array say, earlier top =3 , now top is decremented to 2 and we return s[2] , hence element at s[3] gets popped out.

5. ***Peek Operation***

Peek operation returns the top of the stack i.e., the element where top is pointed at current state.

Here we have extra checking of emptiness of the stack.

```
int isEmpty(Stack st)
{
    if (st.top == -1)
    {
        return 1;
    }
    return 0;
}
```

If top = -1 then return 1 which true that Stack empty else it will return 0, that represents stack is not empty.

Now coming to Peek operation.

```
int peek(Stack st)
{
    if (isEmpty(st)==0)
    {
        return st.s[st.top];
    }
    return -1;
}
int main()
{

 Stack stck;

 peek(stck)

}
```

Or,

```
int peek(Stack st)
{
    if (!isEmpty(st))
    {
        return st.s[st.top];
    }
    return -1;
}
```

```
int main()
{

 Stack stck;

 peek(stck)

}
```

Note here peek(stck) and int peek (Stack st) {...} is call by value not call by reference where st is <u>Formal Parameter</u> (Formal Object of Structure) and stck is <u>Actual Parameter</u> (Actual Object of Structure).

$\boldsymbol{if(!\,isEmpty(st))\ i.e.\ isEmpty\ is\ not\ 1\ i.e.\ 0\ (false)}$

$\boldsymbol{or\ if(isEmpty == 0)\ then\ it\ will\ return\ top.}$

$\boldsymbol{else\ isEmpty(st)\ is\ 1\ (true) then\ it\ will\ return}$

$\boldsymbol{-1.}$

$\boldsymbol{And\ we\ know\ ,if\ Top = \ -1\ ,stack\ is\ Empty.}$

Another extra operation we have is to check, whether Stack is Full or not.

```
int isFull(Stack st)
{
    return st.top == st.size - 1;
}

int main()
{

    Stack stck;
   isFull(stck);
}
```

Note == is relational operator, checks if

1st Operand = 2nd Operand is true.

i.e. top = size-1 is true, then it returns 1 (True).

6. Is Empty Operation

```
int isEmpty(Stack st)
{
    if (st.top == -1)
    {
        return 1;
    }
    return 0;
}
int main()
{

    Stack stck;
     isEmpty(stck)

}
```

$\boldsymbol{if\ top\ =\ -1, then\ it\ returns\ 1\ (true) else\ return\ 0\ (False).}$

7.*Stack Traversal*

```
void Display(Stack st)
{

    for (int i = st.top; i >= 0; i--)
    {
        cout << st.s[i] << " ";
    }
    cout << endl;
}

int main()
{

    Stack stck;
     Display (stck)

}
```

Here we traverse and get all the element , that are being pushed into the stack.

7. *Deleting the Stack*

Now , when we exit from stack as

```
exit(0)
```

function stored in:

```
#include <stdlib.h>
```

header file.

```
        free(stck.s);
        stck.s = NULL;
        exit(0);
```

Therefore as we exit from Stack , first we free stack , then assign the pointer variable to NULL and then exit as here we are just describing the dynamic memory allocation of the program.

This is all about operation of stack.

-------------- **************** -----------------------