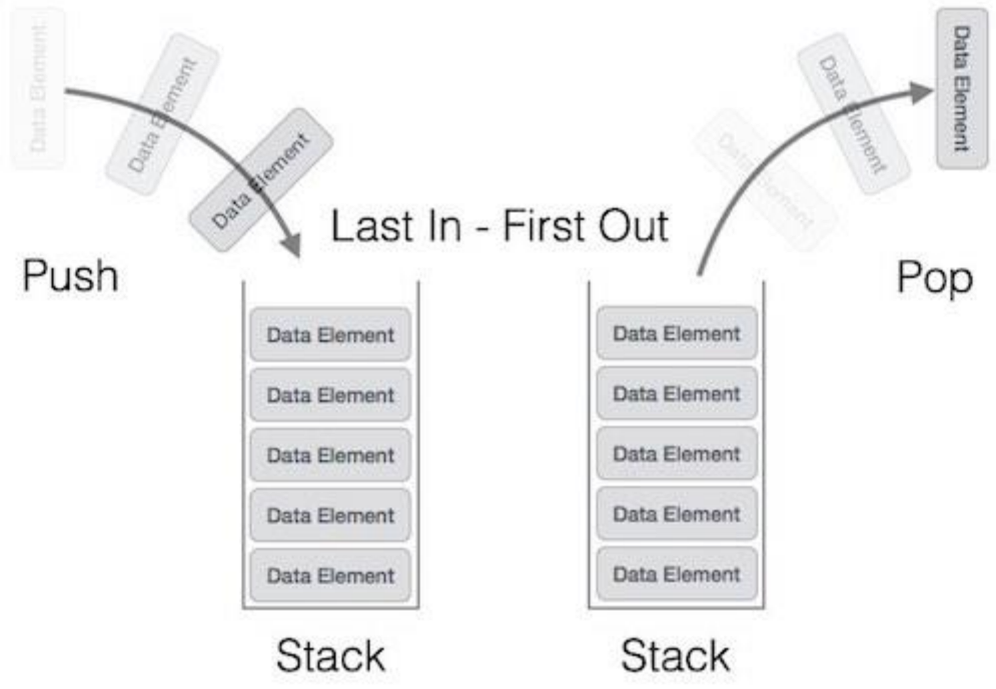
FACULTY NAME- PRAVIN KUMAR PANDEY
DEPARTMENT NAME- COMPUTER SCIENCE AND ENGINEERING
SUBJECT- DATA STRUCTURE USING- C
SEMESTER- 3
YEAR- 2

UNIT-2

Stack:

It is a sequence of items/data element that are accessible at only one end of the sequence. Think of a stack as a collection of items/data element that are piled one on top of the other, with access limited to the topmost items/data element.

A stack inserts items/data element on the top of the stack and removes item from the top of the stack. It has LIFO (last-in / first-out) ordering for the items/data element on the stack.



Stack (ADT) Data Structure:

Stack is an Abstract data structure (ADT) works on the principle of last in first out (LIFO). The last element add to the stack is the first element to be delete. Insertion and deletion can be takes place at one end called TOP. It looks like one side closed tube.

- The add operation of the stack is called **push** operation.
- The delete operation is called as **pop** operation.
- Push operation on a full stack causes stack overflow.
- Pop operation on an empty stack causes stack underflow.
- **TOP** is a pointer variable, which is used to access the top element of the stack.

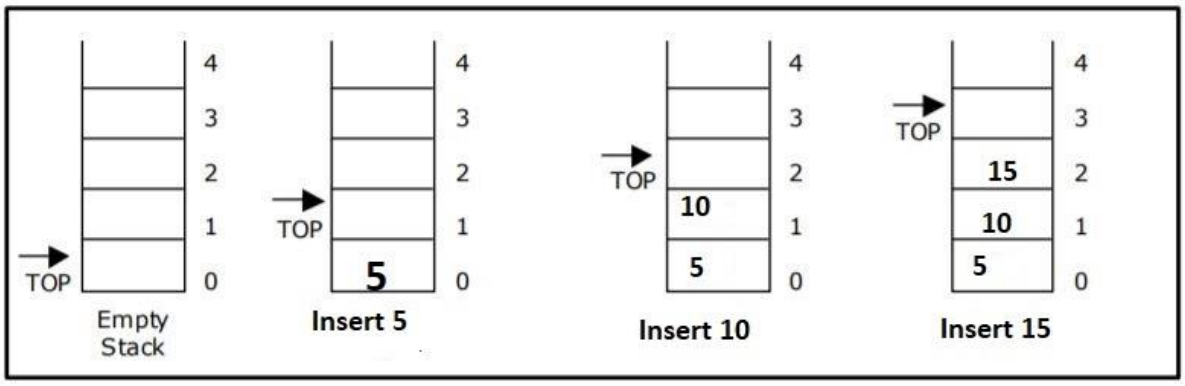
- If you push elements that are added at the top of the stack;

- In the same way when we pop the elements, the element at the top of the stack is deleted.

Representation of Stack:

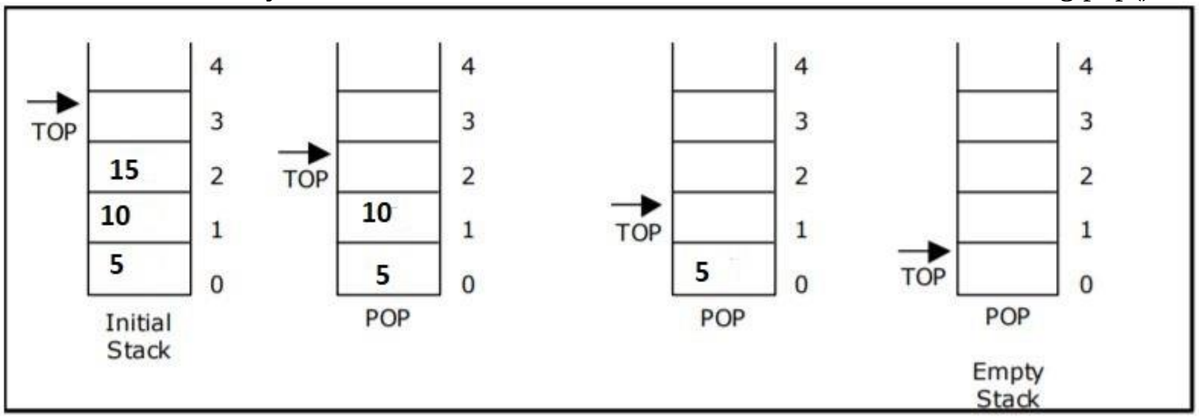
Let us consider a stack with 5 elements capacity. This is called as the size of the stack. The number of elements to be added should not exceed the maximum size of the stack. If we attempt to add new element beyond the maximum size, we will encounter a stack overflow condition. Similarly, you cannot remove elements beyond the base of the stack. If such is the case, we will reach a stack underflow condition.

When an element is added to a stack, the operation is performed by push(). Below figure shows the creation of a stack and addition of elements using push().



Push operations on Stack

When an element is taken off from the stack, the operation is performed by pop(). Below figure shows a stack initially with three elements and shows the deletion of elements using pop().



Pop operations on Stack

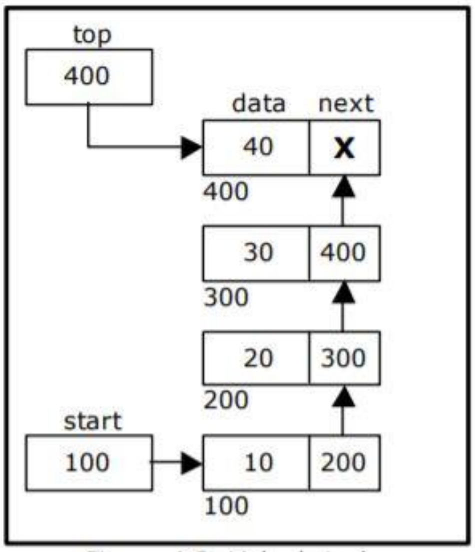
Source code for stack operations, using array

```
# include
# include
# include
# define MAX 6
int stack[MAX];
int top = 0;
int menu()
```

```
{
        int ch;
        clrscr();
        printf("\n … Stack operations using ARRAY... ");
        printf("\n -----------**********-------------\n");
        printf("\n 1. Push ");
        printf("\n 2. Pop ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter your choice: ");
        scanf("%d", &ch);
        return ch;
}
void display()
{
        int i;
        if(top == 0)
        {
                printf("\n\nStack empty..");
                return;
        }
        Else
        {
                printf("\n\nElements in stack:");
                for(i = 0; i < top; i++)
                        printf("\t%d", stack[i]);
        }
}
void pop()
{
        if(top == 0)
        {
                printf("\n\nStack Underflow..");
                return;
        }
        else
                printf("\n\npopped element is: %d ", stack[--top]);
}
void push()
{
        int data;
        if(top == MAX)
        {
                printf("\n\nStack Overflow..");
                return;
        }
        else
        {
                printf("\n\nEnter data: ");
                scanf("%d", &data);
                stack[top] = data;
                top = top + 1;
                printf("\n\nData Pushed into the stack");
        }
}
void main()
{
        int ch;
        do
        {
                ch = menu();
                switch(ch)
                {
                        case 1:
                                push();
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                exit(0);
                }
                getch();
        } while(1);
}
```

Linked List Implementation of Stack:

We can represent a stack as a linked list. In a stack push and pop operations are performed at one end called top. We can perform similar operations at one end of list using top pointer. The linked stack looks as shown in figure.



Linked stack representation

Source code for stack operations, using linked list:

```
# include
# include
# include
struct stack
{
        int data;
        struct stack *next;
};
void push();
void pop();
void display();
typedef struct stack node;
node *start=NULL;
node *top = NULL;

node* getnode()
{
        node *temp;
        temp=(node *) malloc( sizeof(node)) ;
        printf("\n Enter data ");
        scanf("%d", &temp -> data);
        temp -> next = NULL;
        return temp;
}
void push(node *newnode)
{
        node *temp;
        if(newnode == NULL)
        {
                printf("\n Stack Overflow..");
                return;
        }
        if(start == NULL)
        {
                start = newnode;
                top = newnode;
        }
        else
        {
                temp = start;
                while (temp -> next! = NULL)
                        temp = temp -> next;
                temp -> next = newnode;
                top = newnode;
        }
        printf("\n\n\t Data pushed into stack");
}
void pop()
{
        node *temp;
        if (top == NULL)
        {
                printf("\n\n\t Stack underflow");
                return;
        }
        temp = start;
        if( start -> next == NULL)
        {
                printf("\n\n\t Popped element is %d ", top -> data);
                start = NULL;
                free(top);
                top = NULL;
        }
        else
        {
                While (temp -> next! = top)
                {
                        temp = temp -> next;
                }
                temp -> next = NULL;
                printf("\n\n\t Popped element is %d ", top -> data);
                free(top);
                top = temp;
        }
}
void display()
{
        node *temp;
        if(top == NULL)
        {
                printf("\n\n\t\t Stack is empty ");
        }
        else
        {
                temp = start;
                printf("\n\n\n\t\t Elements in the stack: \n");
                printf("%5d ", temp -> data);
                while(temp != top)
                {
                        temp = temp -> next;
                        printf("%5d ", temp -> data);
                }
        }
}
char menu()
{
        char ch;
        clrscr();
        printf("\n \tStack operations using pointers.. ");
        printf("\n -----------**********-------------\n");
        printf("\n 1. Push ");
         printf("\n 2. Pop ");
        printf("\n 3. Display");
        printf("\n 4. Quit ");
        printf("\n Enter your choice: ");
        ch = getche();
        return ch;
}
void main()
{
        char ch;
        node *newnode;
        do
        {
                ch = menu();
                switch(ch)
                {
                        case '1' :
                                newnode = getnode();
                                push(newnode);
                                break;
                        case '2' :
                                pop();
                                break;
                        case '3' :
                                display();
                                break;
                        case '4':
                                return;
                }
                getch();
        } while(ch != '4' );
}
```

Stack Applications:

- Stack is used by compilers to check for balancing of parentheses, brackets and braces.
- Stack is used to evaluate a postfix expression.
- Stack is used to convert an infix expression into postfix/prefix form.
- In recursion, all intermediate arguments and return values are stored on the processor's stack.
- During a function call the return address and arguments are pushed onto a stack and on return they are popped off.
- Depth first search uses a stack data structure to find an element from a graph.

Arithmetic Expressions

The way to write arithmetic expression is known as a **notation**. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are-

- Infix Notation
- Prefix (Polish) Notation
- Postfix (Reverse-Polish) Notation

Infix Notation:
We write expression in infix notation, e.g. a - b + c, where operators are used in-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation:
In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

Postfix Notation:
This notation style is known as Reversed Polish Notation. In this notation style, the operator is postfixed to the operands i.e., the operator is written after the operands. For example, ab+. This is equivalent to its infix notation a + b.

Operator precedence:
We consider five binary operations: +, -, *, / and $ or ↑ (exponentiation). For these binary operations, the following in the order of precedence (highest to lowest)

OPERATOR	PRECEDENCE	VALUE
Exponentiation ($ or ↑ or ^)	Highest	**3**
*, /	Next highest	**2**
+, -	Lowest	**1**

In-fix- to Postfix Transformation with stack:
Procedure:

1. Scan the infix expression from left to right.
2. If the scanned symbol is left parenthesis, push it onto the stack.
3. If the scanned symbol is an operand, then place directly in the postfix expression (output).
4. If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
5. If the scanned symbol is an **operator, then go on removing all the operators from the stack and place them in the postfix expression**, if and only if the precedence of the operator which is on the top of the stack is greater than (or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

Example1:
Convert the infix expression A + B * C – D / E * H into its equivalent postfix expression

Symbol	Postfix expression	Stack
A	A	
+	A	+
B	AB	+
*	AB	+*
C	ABC	+*

-	ABC*+	-
D	ABC*+D	-
/	ABC*+D	-/
E	ABC*+DE	-/
*	ABC*+DE/	-*
H	ABC*+DE/H	-*
	ABC*+DE/H*-	The input is now empty. Pop the output symbols from the stack until it is empty.

Example2:

Convert ((A – (B + C)) * D) ↑ (E + F) infix expression to postfix form:

Symbol	Postfix expression	Stack
(		(
(		((
A	A	((
-	A	((-
(	A	((-(
B	AB	((-(
+	AB	((-(+
C	ABC	((-(+
)	ABC+	((-
)	ABC+-	(
*	ABC+-	(*
D	ABC+-D	(*
)	ABC+-D*	
↑	ABC+-D*	↑
(	ABC+-D*	↑ (
E	ABC+-D*E	↑ (
+	ABC+-D*E	↑ (+
F	ABC+-D*EF	↑ (+
)	ABC+-D*EF+	↑
	ABC+-D*EF+↑	The input is now empty. Pop the output symbols from the stack until it is empty.

In-fix- to Prefix Transformation with stack:

The precedence rules for converting an expression from infix to prefix are identical. The only change from postfix conversion is that traverse the expression from right to left and the operator is placed before the operands rather than after them. The prefix form of a complex expression is not the mirror image of the postfix form.

Example2:

Convert the infix expression A + B - C into prefix expression.

Symbol	Prefix expression	Stack
C	C	
-	C	-

B	BC	-
+	BC	-+
A	ABC	-+
	-+ABC	The input is now empty. Pop the output symbols from the stack until it is empty.

References

[1] https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

[2] https://www.edutechlearners.com/download/books/DS.pdf

[3]https://www.iare.ac.in/sites/default/files/DS_NOTES_BY_Dr_L_V_NARASIMHA_PRASAD_0.pdf

[4]http://cse.iitkgp.ac.in/~pds/notes/