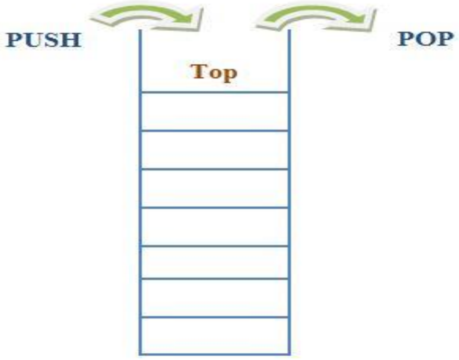
Stack

- Stack is a linear data structure.
- Stack is an ordered list of similar data type.
- Stack in which elements follow a specific order and operations will be performed.
- Stack follows LIFO(Last In First Out).
- Elements will be added from the end called TOP of stack and removes from the same end.



Stack ADT: Abstract Data Type specifies the operations can be performed on Stack

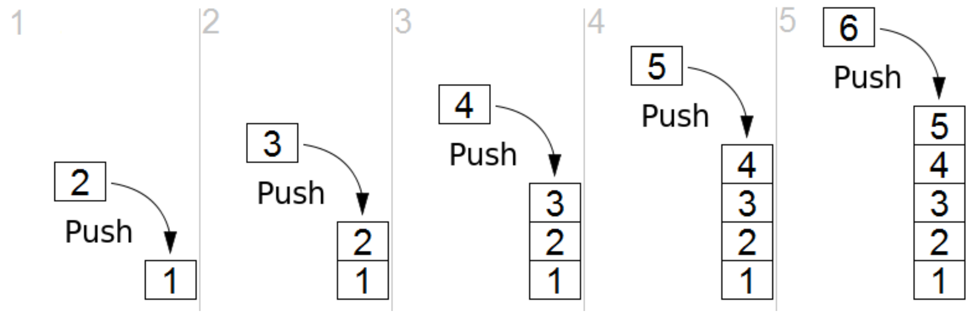
Push: Push an element on to the Stack. Returns "Overflow" if stack is full.

Pop: Deletes an item from Stack. Returns "Underflow" if Stack is empty.
Peek: Returns top element of stack but not removes.
isEmpty: Returns true if stack is empty, else false
isFull: Returns true if stack is full, else false

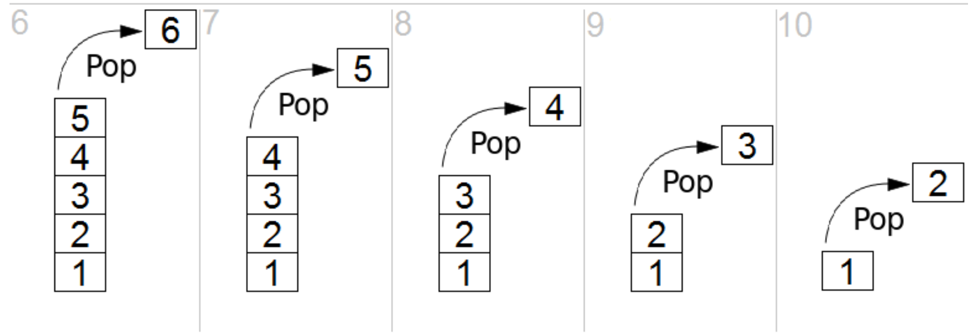
Stack workflow:

- A pointer called TOP is used to keep track of the top element in the stack.
- The initial value of TOP is -1 so that the stack is empty if TOP == -1.
- We increase the value of TOP and place the new element in the position pointed to by TOP on PUSH.
- TOP value modified by -1 on POP and returns the item which is deleted.
- Before push, we need to test if the Stack is Full or Not
- Before pop, we need to test if the Stack is Empty or Not

The flow of pushing elements on to the stack:



The flow of deleting items from Stack:



Applications of stack:

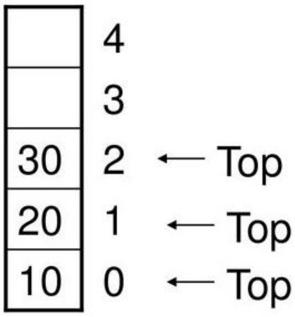
1. Function recursion.
2. Finding the Binary value of Decimal value.
3. Infix to Postfix or Prefix conversion
4. Forward and backward feature in web browsers
5. Tower of Hanoi problem
6. Tree traversals
7. Backtracking
8. DFS algorithm of Graph

Stack can be implemented in 2 ways:

- Using arrays
- Using linked lists

Implementing Stack using Arrays:

- In stack implementation using arrays, we use simple array to represent the stack as follows.
- We use primitive variable top to represent index and perform operations on Stack.



- Top = -1

Method1:

- In this implementation we use simple array to represent stack.
- In this code program, we perform all the stack operations such as push, pop, traverse without using functions.

```
#include<stdio.h>
int stack[5], top=-1, size=5 ;
int main()
{
        int ch,ele;
        while(1)
        {
                printf("1.Push \n");
                printf("2.Pop \n");
                printf("3.Peek \n");
                printf("4.Traverse \n");
                printf("5.Quit \n");

                printf("Enter choice : ");
                scanf("%d", &ch);

                if(ch==1)
                {
                        printf("Enter element : ");
                        scanf("%d", &ele);
                }
                switch(ch)
                {
                        case 1 :        if(top==size-1)
                                        printf("Stack is OverFlow\n\n");
                                        else
```

```
                         {
                              stack[++top] = ele;
                              printf("Element pushed onto stack\n\n");
                         }
                         break ;

               case 2 :  if(top==-1)
                              printf("Stack is Underflow\n\n") ;
                         else
                              printf("Popped item is : %d\n\n",
stack[top--]);
                              break ;

               case 3 :  if(top==-1)
                              printf("Stack is Underflow\n\n") ;
                         else
                              printf("Peek element is :
%d\n\n",stack[top]);
                              break ;

               case 4 :  if(top==-1)
                              printf("Stack has no elements to
display\n\n");
                         else
                         {
                              int i ;
                              printf("Stack elements :\n");
                              for(i=top ; i>=0 ; i--)
                                   printf("%d\n",stack[i]);
                         }
                         break;

               case 5 :  exit(1);
               default:  printf("Invalid choice\n\n");
          }
     }
     return 0 ;
}
```

Method-2:

- We use functions to implement the stack in this approach
- We call the functions from the main functions.
- We represent the prototypes of each function before definition

```
#include<stdio.h>
#define SIZE 5

void push();
void pop();
void peek();
void traverse();

int stack[SIZE], top=-1;
int main()
{
        int ch, ele;
        while(1)
        {
                printf("Stack operations :\n");
                printf("1. Push\n");
                printf("2. Pop\n");
                printf("3. Peek\n");
                printf("4. Display\n");
                printf("5. Quit\n");

                printf("Enter your choice : ");
                scanf("%d", &ch);

                switch(ch)
                {
                        case 1 :        push();
                                        break;

                        case 2 :        pop();
                                        break ;

                        case 3 :        peek();
                                        break ;

                        case 4 :        traverse();
                                        break;

                        case 5 :        exit(0);
                        default:        printf("Invalid choice \n");
                }
        }
        return 0;
}

void push()
```

```
{
	if(top==SIZE-1)
	{
		printf("Stack is Full \n");
	}
	else
	{
		int ele;
		printf("Enter ele to push : ");
		scanf("%d", &ele);
		stack[++top]=ele;
		printf("Element inserted...\n");
	}
}

void pop()
{
	if(top==-1)
	{
		printf("Stack is Empty\n");
	}
	else
	{
		printf("Popped : %d \n", stack[top--]);
	}
}

void peek()
{
	if(top==-1)
	{
		printf("Stack is Empty\n");
	}
	else
	{
		printf("Peek : %d \n", stack[top]);
	}
}

void traverse(void)
{
	if(top==-1)
	{
		printf("Stack is Empty \n");
	}
	else
```

```
        {
                int i;
                printf("Stack elements are :\n");
                for(i=top ; i>=0 ; i--)
                {
                        printf("%d \n", stack[i]);
                }
        }
}
```

Method-3:

- In this implementation we use static array to represent the stack.
- We use functions to perform all operations.
- Functionality programming advantage is code re-usability and easy debugging.
- The Program code as follows:

```
#include<stdio.h>
#define SIZE 5

void push(int);
int pop(void);
int peek(void);
void traverse(void);
int isFull(void);
int isEmpty(void);

int stack[SIZE], top=-1;
int main()
{
        int ch, ele;
        while(1)
        {
                printf("Stack operations :\n");
                printf("1. Push\n");
                printf("2. Pop\n");
                printf("3. Peek\n");
                printf("4. Display\n");
                printf("5. Quit\n");

                printf("Enter your choice : ");
                scanf("%d", &ch);

                switch(ch)
        {
            case 1 :    printf("Enter element to push : ");
                            scanf("%d", &ele);
                            push(ele);
                            break;

            case 2 :    ele = pop();
                            if(ele)
                                printf("Popped : %d \n", ele);
                            else
                                printf("Stack is Empty \n");
                            break ;

            case 3 :    ele = peek();
                            if(ele)
                                printf("Peek item : %d \n", ele);
                            else
                                printf("Stack is Empty \n");
                            break ;

            case 4 :    traverse();
                            break;

            case 5 :    exit(0);
            default:    printf("Invalid choice \n");
        }
    }
    return 0;
}

void push(int ele)
{
    if(isFull())
    {
        printf("Stack is Full \n");
    }
    else
    {
        ++top;
        stack[top]=ele;
        printf("Element inserted...\n");
    }
}

int pop(void)
{
	int ele;
	if(isEmpty())
	{
		return 0;
	}
	else
	{
		ele=stack[top];
		--top;
		return ele;
	}
}

int peek(void)
{
	if(isEmpty())
	{
		return 0;
	}
	else
	{
		return stack[top];
	}
}

void traverse(void)
{
	if(isEmpty())
	{
		printf("Stack is Empty \n");
	}
	else
	{
		int i;
		printf("Stack elements are :\n");
		for(i=top ; i>=0 ; i--)
		{
			printf("%d \n", stack[i]);
		}
	}
}

int isFull(void)
{
	if(top==SIZE-1)
		return 1;
```

```
        else
                return 0;
}

int isEmpty(void)
{
        if(top==-1)
                return 1;
        else
                return 0;
}
```

Method4:

- In this implementation we use dynamic functionality of arrays.
- Dynamic stack means, the Stack with specified initial capacity and the capacity shrinks and grows depends or insertion and deletion of elements.
- We use DMA functionality of stdlib.h header file such as calloc(), realloc() and free()
- Dynamic stack operations can be performed using Pointers.
- Stack and TOP are pointer type variables in Dynamic Stack implementation.
- The code program as follows:

```
#include<stdio.h>

int size;
int *stack, *top;

void createStack();
void push();
void pop();
void traverse();

int main()
{
        int ch;
        createStack();

        printf("Stack operations :\n");
        while(1)
        {
                printf("1.Push \n");
                printf("2.Pop \n");
                printf("3.Display \n");
                printf("4.Quit \n");

                printf("Enter choice : ");
                scanf("%d", &ch);

                switch(ch)
                {
                    case 1  :   push();
                                break;

                    case 2  :   pop();
                                break;

                    case 3  :   traverse();
                                break;

                    case 4  :   exit(1);
                    default :   printf("Invalid choice\n\n");
                }
        }
        return 0;
}

void createStack()
{
        printf("Enter initial size : ");
        scanf("%d", &size);

        stack = (int*)calloc(size, sizeof(int));
        if(stack==NULL)
        {
                printf("Stack creation failed\n");
                exit(0);
        }
        else
        {
                top = stack;
        }
}
void push()
{
        int ele;
        printf("Enter element to be pushed : ");
        scanf("%d" , &ele);
        if(top==stack+size)
        {
```

```
                ++size;
                stack=(int*)realloc(stack, size*sizeof(int));
                if(stack==NULL)
                {
                        printf("Unable to increase the size \n");
                }
                else
                {
                        *top = ele;
                        ++top;
                        printf("Element pushed...\n");
                }
        }
        else
        {
                *top = ele;
                ++top;
                printf("Element pushed...\n");
        }
}
void pop()
{
        if(stack==top)
        {
                printf("Stack is Empty\n");
        }
        else
        {
                --top;
                printf("Popped : %d \n", *top);
                if(size>5)
                {
                        --size;
                        stack = (int*)realloc(stack , size*sizeof(int));
                }
        }
}
void traverse()
{
        if(stack==top)
        {
                printf("Stack is Empty\n");
        }
        else
        {
                int* i;
                printf("Stack elements are : \n");
                for(i=top-1 ; i>=stack ; i--)
                {
                        printf("%d \n", *i);
                }
        }
}
```