1) • It is the foundation of generic programming which involves writing a code that is independent of particular type.
• It can be viewed as variable that can be instantiated to any data type, irrespective of whether this data type is fundamental or user defined.
• It enforces code reusability feature of C++

```
#include <iostream>
using namespace std;
template <class T>
void sort (T a[], int n)
  {
    for (int i=0; i<n-1; i++)
     { for(int j=0; j<n-1-i; j++)
        { if(a[j] > a[j+1])
          {
            T temp = a[j];
              a[j] = a[j+1];
              a[j+1] = temp;
          }
template <class T>
void Print (T a[], int n)
   { cout << "Array is \t";
     for(int i=0; i<n; i++)
       { cout << a[i] << "\t";
       } }
int main ()
{ int a;
```

```
cout << " enter the size of the int array " << endl;
cin >> a;
int ar [a];
cout << " enter the elements " << endl;
for(int i=0; i<a; i++)
    {
     cin >> ar[i];
    }
cout << " original \n";
  Print (ar, a);
  cout << " sorted " << endl;
  Print (ar, a);
cout << " enter the size for float array \n";
  cin >> a ; float arr [a];
  cout << " enter the elements " << endl;
  for(int i=0; i<a; i++)
    {
     cin >> arr[i];
    }
  cout << " original " << endl;
    printe (arr, a);
    cout << " sorted \n";
    Print << (arr, a);
}
```

```
cout << " enter the size of the int array " << endl;
cin >> a;
int ar [a];
cout << " enter the elements " << endl;
for( int i=0; i<a; i++)
    {
     cin >> ar[i];
    }
cout << " original \n";
  Print (ar ,a );
  cout << " sorted " << endl;
  Print (ar,a) ;
cout<< " enter the size for float array \n";
  cin>> a  ;  float arr [a];
  cout << " enter the elements " << endl;
  for( int i=0; i<a ; i++)
     {
      cin >> arr[i] ;
     }
  cout<< " original " << endl ;
     printe (arr,a);
     cout<< " sorted \n " ;
     Print << (arr, a);
}
```

```
template <class T>
  void secondlargest(T a[], int n)
   {
     for (int k=0; k<n; k++)
       {
         for(int l=0; l<n-i; l++)
          {
            if (a[l] > a[l+1]
              {
                T temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
              }
          }
       }
  cout << " the second largest element is" << a[n-2];
    }
int main ()
  {
    int s;
   cout << "enter size of array \n";
    cin>>s;
    int arr[s];
    cout << "enter the elements of integer array" <<endl;
    for(int i=0; i<s; i++)
     {
       cin>> arr[i];
     }
      float a;
   secondlargest(arr, s);
  cout<<"enter the size for float array \n";
  cin>> a;
   float ar[a];
  cout<<" enter the elements for float array" << endl;
   float for (int j=0; j<a; j++) {
```

```
cin >> arr[a];
}
secondlargest(arr,a);
}
```

3) The steps are –

1) start
2) input a prefix expression in the 'prefix' string.
3) for each last character 'ch' of 'prefix' string

```
if('ch' is an operand)
  push 'ch' in 'operandstk'
else
 op2 = pop an operand from 'operandstk'
op1 = pop an operand from 'operandstk'
value = result of applying 'ch' in 'op2' & 'op1'
push 'value' to 'operandstk'
```

4) Pop the top element of the stake 'operandstk' which is the required result.
5) Exit.

4) Given expression −*+ABCD where
A = 4, B = 3, C = 2, D = 5

Symbol	op1	op2	value	opnd stack
5				5
2				5,2
3				5,2,3
4				5,2,3,4
+	4	3	7	5,2,7
*	7	2	14	5,14
−	14	5	9	9

result

The steps are

1) if ch is an operand, immediately output.
2) close parenthesis : pop stack symbol until an open parenthesis appears.
3) operator, pop all stack symbols until a symbol of lower precedence or a right-associative symbol of equal precedence appears. Then push the operator.
4) End of input, pop all remaining stack symbols.

convert the infix exp ((a+b)*c-(d-e))^(f+g) to postfix using stack.

Symbol	stack ~~Postfix String~~	string ~~operstack~~.
(	(	
(	((	
a	((	a
+	((	a
b	((+	ab
*	(*	ab+
c	(*	ab+c
-	(* -	ab+c
(	(-(	ab+c*
d	(-(	ab+c* d
-	(-(-	ab+c* d
e	-(-	ab+c*de
)	-	ab+c*de -
)	-	ab+c*de--
^	^	ab+c* de--
(	^(	ab+c* de--
f	^ (	ab+c* de-f
+	^(+	ab+c* de-f
g	^(+	ab+c*de-fg
)	^ +	ab+c*de-fg+

```
void addq(int item)
{
  // add an item to the queue
  int k = (rear+1) % max_size;
  if(front == k)
  {
    cout << "queue full");
    return;
  }
  rear = k;
  queue [rear] = item;
}
```

Steps:

1) pass a value to the function,
2) assign [(rear+1) % maxsize] to some variable
3) check if k equal to front.
4) if true, show message queue is full.
5) else assign the value to queue [rear]

The steps are

1) Read in the expression
2) Process each character of the expression to a single digit number (0 to 9) then push it into the stack.
   If the character corresponds to one of the arithmetic operations (+, -, *, /), then.
   - pop a number off the stack, call it op2.
   - pop a number off the stack, call it op1.
   - combine the operand using the arithmetic operator as follows

     Result = op1 operator op2
   - push result onto the stack.
3) When the end is reached, pop the remaining number the, stack, This number is the value of the expressions.

) Postfix expression ab+c*de--fg+^

12+3*45--67+^

Scanned Symbol	op1	op2	result	stack
1				1
2				1,2
+	2	1	3	3
3				3,3
*	3	3	9	9
4				4,9
5	-	-	-	5,4,9
-	4	5	-1	-1,9
-	9	-1	10	10
6	-	-	-	6,10
7	-	-	-	7,6,10
+	6	7	13	13,10
^	10	13	$10^{\wedge}13$	$10^{\wedge}13$

The problems associated with linear queue are By the defination of the queue, when we add an element in queue, rear pointer is increased by 1 element in the queue. when we remove front pointer is increased by 1.

Consider the operations performed on queue with size 5 as follows.

i) Initially empty queue is there so, front = 0, & rear = -1

ii) when we add 5 elements to the queue, the data of the queue becomes as follows front = 0 & rear = 4.

iii) Now we have deleted 2 elements from queue so there should be space for another 2 elements, but as rear pointer is pointing at last position so queue overflow condition is true, we cant insert any new element in the queue even if it has empty space. To overcome this problem there is another variation of queue known as circular queue.

Big oh notation is a mathematical notation that describes the limiting behaviour of function when argument tends towards a particular value or infinity In computer science, big o notation is used to classify algorithm by how they respond to changes in input size such as how the processing time of an algorithm changes as the problem size becomes extremely large.

$f(n) = O(g(n))$ if there exists positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n > n_0$

Proof:

By the big oh notation $f(n)$ is $O(n^3)$ if $f(n) \leq n_0$ for some $n > n_0$ let us check this condition.

if $n^3 + 20n + 1 \leq cn^3$

then $1 + \frac{20}{n^2} + \frac{1}{n^3} \leq c$

then Big oh notation holds for $n \geq n_0 = 1$

and $c \geq 22$.

Performance of an algorithm can be measured in two ways

i) space complexity

ii) time complexity.

Space complexity refers to the amount of space needed by a program to run to completion.
It has 2 parts.
i) Fixed Part - its independent of input and outputs, constants, fixed size component variable, instruction spaces, simple variable.
ii) Variable part is space needed by component variable whose size is dependent on the particular problem instance being solved and recursion stack space.
$S(P)$ space requirement of any program is
$S(P) = C + Sp(I)$ where C is a constant and I is the instance characteristics.

```
float sum(float a[], const int n)
  {
    float sum = 0;
    for(int i = 0; i < n; i++)
      {
        sum += a[i];
      }
  }.
```

Since n is passed by value, one word must be allocated to it. a is the base address, one word for it space needed by the fun is independent of n, hence
$Sp(n) = 0$.

Time complexity of any program $T(P)$ is given by compile time and run time, compile time doesn't depend upon instance characteristics. We focus on the run time of the program denoted by $T(P)$.
Program step is syntactically and symmantically meaningful segment of the program that has run time and independent of the instance characteristics.

```
float sum (float * a, const int n)
{
    int s=0   // one time
    for (int i =0; i<n; i++) // for n+1 times
    { s+=a[i];   // n times
    }
    return 0; // one time
}
```

Total $2n+3$.

13) Recursion - It is the name given for expressing anything in terms of itself.

Recursion function is a function which calls itself until a particular condition is met.

Properties.

i) Recursive algorithm should terminate at some point otherwise recursion will never end.

ii) hence recursive algorithm should have stopping condition to terminate along with recursive calls.

Advantages -

1) Clearer and simpler versions of algorithm can be created using recursion

2) Recursive defination of a problem can be easily translated into recursive functions.

3) Lot of book keeping activities such as initialisation etc required in iterative solutions is avoided.

Disadvantage -

1) When a function is called, the function saves form parameters, local variables and return addresses and hence consumes a lot of memory.

2) Lot of time in pushing and popping and hence consumes more time to compute the result.

```
#include <iostream>
#include <string.h>
using namespace std;
bool isPal( const string &str, int start, int end)
  {
    if(start >= end)
       return true;
    if(str[start] != str[end])
       return false;
    return isPal(str, ++start, --end);
  }

int main()
  {
    char str[50];
    cout<<" enter string \n";
    cin>> str;
    int i =0;
    int 0 = strlen(str);
    0=5-1
   if(int check = isPal(str,i,s))
     cout <<" is Palindrome";
    else
     cout <<" not Palindrome";
    }
```

```
# include <iostream>
using namespace std;
int fib(int n)
{
  if(n==0)
  return 0;
  if(n==1)
   return 1;
else
  { return fib(n-1) + fib(n-2); } }
```

```
int main
{ int f;
  f = fib(4);
  cout<< f;
}
```

Fibonacci sequence

0, 1, 1, 2, 3, 5, 8 ....

Each element is the sum of two preceeding elements and fibonacci of a number is the value at that position in sequence.

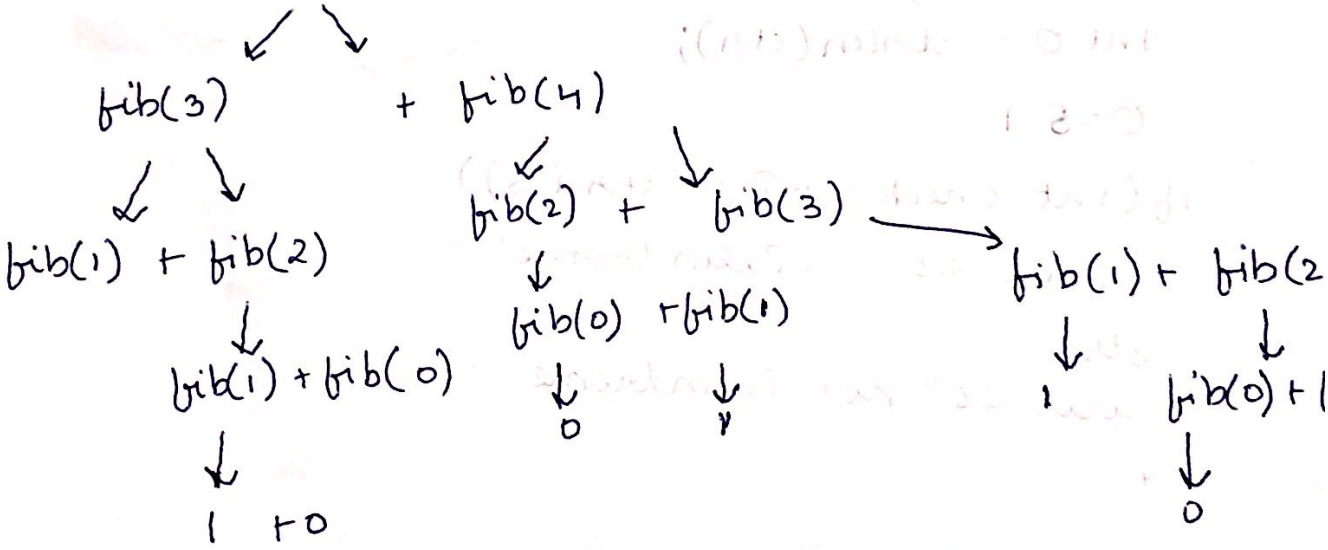
ie fib(0) = 0

fib(1) = 1

fib(2) = 1

fib(3) = 2

For fib(5)



16) A stack is an ordered list in which insertions and deletions are made at one end called the top. It also called first in first out list

```
template <class T, int Maxsize>
 class stack
{ T arr [maxsize];
  int top;
```

```
public:
stack() { top = -1; }
bool isEmpty() { return top == -1; }
bool isFull() { return top == maxsize -1; }
void push(T item)
 { if (Isfull()) {
      cout << " stack overflow "; }
        arr[++top] = item;
       }
T pop {
  if (ifEmpty()) {
   cout << " stack underflow ";
       return -99; }
   return arr[top--];
    }
};
```

```
#include <iostream>
using namespace std;
int counter =0;
void tower (int n, char source, char temp, char dest)
{ if(n==1)
   {
cout<< " move disk 1 from " << source << " to " << dest <<
       endl;
      counter++;
      return;
    }
 tower(n-1, source, dest, temp);
cout<< " move disk " << n << " from " << source <<
 << dest << endl;
```

```
    counter++;
  tower(n-1, temp, source, dest);
  }
int main()
  {
    char a, b, c;
     int n;
  cout<<" enter the no of disk"<<endl;
     cin>> n;
   tower(n,'a','b','c');
   cout<<" the steps are \n";
   }
```

```
#include<iostream.h>
#include<stdlib.h>
using namespace std;
class twostack
{
  int *arr;
  int size;
  int top1, top2;
 public:
   twostack(int n)
   { size = n;
     arr = new int[n];
     top1 = -1;
     top2 = size;
   }
 void push(int x)
  { if (top1< top2-1)
      { top1++;
        arr[top1] = x;
          }
```

```
else
{
 cout<<" stack overflow "<<endl;
  exit(1);
  }
 }
void push2 (int x)
 { if (top1 < top2-1)
    { top2--;
    arr[top2] = x;
  }
 else
  { cout<<" stack overflow \n";
     exit (0);
   }
 }
int pop1()
 { if(top1 >=0)
    { int x = arr[top1];
       top1--;
       return x;
     }
   else
   {
     cout<<" stack underflow "<< endl;
      exit (0);
     }
   }
 int pop2()
  { if(top2 < size)
      { int x = arr[top2];
        top2++;
        return x;
        }
```

```
    else
    { cout << " stack underflow" << endl;
      exit (1);
    }
  }
};
int main ()
{
  twostacks ts(5);
  ts.push1 (5);
  ts.push2 (10);
  ts.push2 (15);
  ts.push1 (11);
  ts.push (7);
 cout<<" popped element from stack1 is " << ts.pop1
     << endl;
  ts.push(40);
 cout << " popped element from stack2 is " << ts.pop2() << en
  return 0;
}
```

19) There is a special type of ADT called the priority queue. This is a special type of table whose items a assigned priority values as they are added into the structure. This priority value is used to determi the order in which the items in the table are to b processed.

Methods:

create

This is the priority queue's class constructo It will create a new empty priority queue.

isEmpty()

returns true if the priority queue is empty

insert ()

Adds a new item to the priority queue.

delete ()

Removes and returns the item in the priority queue with the highest priority.

Priority queue can be implemented in 4 ways

a) Using arrays

The sort order will be from smallest to largest (ie the item with highest priority will always be the last item in the array)

Limitations :

- Only a limited number of items may be added to the priority queue. If their limit is set to a large number, then this structure may waste a lot of memory.
- Although the delet() method is very efficient, the insert () method is very inefficient.

b) using linked list

The sort order will be from largest to smallest (ie the item with highest priority is always the first item in the list).

Limitations :

- Although the delet() method is very efficient, the insert () method is inefficient.

c) using a sorting algorithm to make a priority queue

A sorting algorithm can be used to implement a priority queue.

Limitations.

- Although the insert () method is efficient, the delet() method is inefficient.

20)
```
# include <iostream>
using namespace std;
# include "stack.h";
char choice;
void menu()
{
  cout << "\n\t Menu";
  cout << "\n 1. Arrival (A)";
  cout << "\n 2. Departure (D)";
  cout << "\n 3. Exit (E)";
  cout << "\n enter your choice";
  cin >> choice;
}
int main()
{
  stack <int> cars(10);
  int cno;
  int pos;
  int c;
  int temp[10] = 0;
  int count[10] = 0;
  menu();
  while (choice != 'E')
  {
    switch (choice)
    {
      case 'A':
        cout << "Enter the car no:";
        cin >> cno;
        if(cars.push(cno))
          { cout << "\n sorry garage is full";
            break;
          }
      else
        { cout << "\n car
```

```
        break;
    }
case 'D':
    cout << "\n Enter the car number to depart : ";
     cin >> cno;
     for(int i=9; i>=0; --i)
        {
          c = cars.pop();
          if(c == cno)
          {
     cout<<" \n Car Successfully departed.";
     cout<<" \n It was moved "<< count[i] << " times to
                let other cars depart.";
          break;
         }
     else
     {
       temp[i] = c;
      cout << [i]++;
       }
   }
   for (int i=0 ; i<10; ++i)
      { if (temp[i] ==0)
        { continue;
          }
        else
          { cars.push (temp[i]);
          }
      }
    }
  }
}
```