One Dimensional Array (C++ Implementation)

```
In all functions shown below assume
ARR   Array of integer
L     Number of occupied element in the array
Max   Maximum size of the array
```

Function to traverse the array ARR

```
void Traverse(int ARR[], int L)
{
  for (int C=0;C<L;C++)
    cout<<ARR[C]<<endl;
}
```

Function to Read elements of the array ARR

```
void Read(int ARR[], int L)
{
  for (int C=0;C<L;C++)
    cin>>ARR[C];
}
```

Function to reverse the content of a one dim array ARR

```
void Read(int ARR[], int L)
{
  for (int C=0;C<L/2;C++)
  {
    int T=ARR[C];
    ARR[C]=ARR[L-C-1];
    ARR[L-C-1]=T;
  }
}
```

Function to Search for an element from ARR by Linear Search

```
void Lsearch(int ARR[], int L)
{
  int Data,Found=0,C=0;
  cout<<"Enter Data to be searched:";cin>>Data;
  while (C<L && !Found)
  {
    if (ARR[C]==Data)
      Found++;
    else
      C++;
  }
  if (Found)
    cout<<"Data Found at :"<<C<<endl;
  else
    cout<<"Data Not Found in the array"<<endl;
}
```

Function to Sort the array ARR by Bubble Sort

```
void BubbleSort(int ARR[], int L)
{
  for (int I=0;I<L-1;I++)
    for (int J=0;J<L-I-1;J++)
      if (ARR[J]>ARR[J+1])
      {
        int Temp=ARR[J];
        ARR[J]=ARR[J+1];
        ARR[J+1]=Temp;
      }
}
```

Function to Sort the array ARR by Insertion Sort

```
void InsertionSort(int ARR[], int L)
{
  for (int I=1;I<L;I++)
  {
    int Temp=ARR[I], J=I-1;
    while (Temp<ARR[J] && J>=0)
    {
      ARR[J+1]=ARR[J])
      J--;
    }
    ARR[J+1]=Temp;
  }
}
```

Function to Sort the array ARR by Selection Sort

```
void SelectionSort(int ARR[], int L)
{
  for (int I=0;I<L-1;I++)
  {
    int Small=I;
    for (int J=I+1;J<L;J++)
     if (ARR[Small]>ARR[J])
       Small=J;
    if (Small!=I)
    {
      int Temp=ARR[I];
      ARR[I]=ARR[Small];
      ARR[Small]=Temp;
    }//if
  }  //for
}
```

Function to Search for an element from ARR by Binary Search

```
void BinarySearch(int ARR[], int L)
{
  int Data,LB=0,UB=L-1,Mid,Found=0;
  cout<<"Enter Data to be searched:";cin>>Data;
  while (LB<=UB && !Found)
  {
    Mid=(LB+UB)/2;
    if (ARR[Mid]<Data)
      LB=Mid+1;
    else
      if (ARR[Mid]>Data)
        UB=Mid-1;
      else Found++;
  }
  if (Found)
    cout<<"Found at:"<<Mid<<endl;
  else
    cout<<"Not Found!!"<<endl;
}
```

Function to merge X and Y arrays (already sorted) of lengths N and M

```
void Merge(int X[],int Y[],int ARR[],int N,int M,int &L)
{
  int I=0,J=0;
  L=0;                //Initialisation of counters for X, Y, and ARR
  while (I<N && J<M)
    if (X[I]<Y[J])
      ARR[L++]=X[I++];
    else
      if (X[I]>Y[J])
        ARR[L++]=Y[J++];
      else
      {
        ARR[L++]=X[I++];
        J++;
      }
  while (I<N) ARR[L++]=X[I++];
  while (J<M) ARR[L++]=Y[J++];
}
```

Two Dimensional Array (C++ Implementation)

Function to read the array A

```
In functions shown below assume
A,B,C   Two Dimensional Arrays of integers
N,L,M   Number of Rows/Columns
```

```
void Read(int A[][20], int N, int M)
{
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
    {
      cout<<”(“<<R<<’,’<<C<<”)?”;cin>>A[R][C];
    }
}
```

Function to find the sum of two dimensional arrays A and B

```
void  Addition(int A[][20],int B[][20],int C[][20],int N,int M)
{
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
      C[R][C]=A[R][C]+B[R][C];
}
```

Function to multiply matrices A and B of order NxL and LxM

```
void  Multiply(int A[][20],int B[][20],int C[][20],int N,int L,int M)
{
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
    {
      C[R][C]=0;
      for (int T=0;T<L;T++)
        C[R][C]+=A[R][T]*B[T][C];
    }
}
```

Function to find & display sum of rows & sum of cols. of a 2 dim. array A

```
void  SumRowCol(int A[][20],int N,int M)
{
  for (int R=0;R<N;R++)
  {
    int SumR=0;
    for (int C=0;C<M;C++)
      SumR+=A[R][C];
    cout<<”Row(“<<R<<”)=”<<SumR<<endl;
  }
  for (int C=0;C<M;C++)
  {
    int SumC=0;
    for (int R=0;R<N;R++)
      SumC+=A[R][C];
    cout<<”Column(“<<C<<”)=”<<SumC<<endl;
  }
}
```

Function to find sum of diagonal elements of a square matrix A

```
void  Diagonal(int A[][20],int N,int &Rdiag,int &LDiag)
{ for (int I=0,Rdiag=0;I<N;I++) Rdiag+=A[I][I];
  for (I=0,Ldiag=0;I<N;I++) Ldiag+=A[N-I-1][I];
}
```

Function to find out transpose of a two dimensional array A

```
void  Transpose(int A[][20],int B[][20],int N, int M)
{
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
      B[R][C]=A[C][R];
}
```

Function to display content of a two dimensional array A

```
void Display(int A[][20], int N, int M)
{
  for (int R=0;R<N;R++)
  {
    for (int C=0;C<M;C++)
      cout<<setw(10)<<A[R][C];
    cout<<endl;
  }
}
```

Function to swap the content of the first and third row of 4x4 matrix A

```
void  Swap1N3(int A[][4])      //     1       2       3       4
{                              //     5       6       7       8
                               //     9       10      11      12
  for (int C=0;C<4;C++)        //     13      14      15      16
  {
    int T=A[0][C]
    A[0][C]=A[2][C];           //     9       10      11      12
    A[2][C]=T;                 //     5       6       7       8
  }                            //     1       2       3       4
}                              //     13      14      15      16
```

Function to add alternate elements in two-dimensional array A of any order

(Note: Access the elements row-wise and start adding elements from A[0][0] onwards)

```
int (int A[][20], int N, int M)
{
  int Sum=0,Alt=0;
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
    {
      if (Alt%2==0)
        Sum+=A[R][C];
      Alt++;
    }
  return Sum;
}
```

```
If the content is

 1  10   6   7
 2   3  12   4
 8  11   5   9

Sum will be:34
(i.e.1+6+2+12+8+5)
```

```
If the content is

 1  10   6
 2   3  12
 8  11   5

Sum will be:23
(i.e.1+6+3+8+5)
```

Function to transfer content from a two dim array to one dim array

```
void (int A[][10],int B[], int N, int M)
{
  int I=0;
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
      B[I++]=A[R][C];
}
```

Function to transfer content from a one dim array to two dim array

```
void (int B[],int A[][10], int N, int M)
{
  int I=0;
  for (int R=0;R<N;R++)
    for (int C=0;C<M;C++)
      A[R][C]=B[I++];
}
```

Function to copy diagonal elements of a square matrix to one dim array

```
void (int B[],int A[][10], int N)
{ int I=0;                     //Matrix A            Array B
  for (int R=0;R<N;R++)        //1  2  3  4          1 6 11 16 4 7 10 13
    B[I++]=A[R][R];            //5  6  7  8
  for (R=0;R<N;R++)            //9  10 11 12
    B[I++]=A[R][N-R-1];        //13 14 15 16
}
```

Stack

It is a non-primitive linear data structure in which insertion and deletion of elements takes place from only one end, known as top. It is also known as LIFO (Last In First Out) data structure.

//Static Stack (Stack implemented using Array)

```
const int Max=5;

void Push(float S[],int &T)
{
  if (T<Max-1)        //Check for Stack not Full
  {
    T++;
    cout<<"Data:";cin>>S[T];
  }
  else
    cout<<"Stack is Full!"<<endl;
}

void Pop(float S[],int &T)
{
  if (T!=-1)          //Check for Stack not Empty
  {
    cout<<S[T]<<" deleted!"<<endl;
    T--;
  }
  else
    cout<<"Stack is Empty!"<<endl;
}

void StackDisp(float S[],int T)
{
  for (int I=T;I>=0;I--)
    cout<<S[I]<<endl;
}

void main()
{
  //Initialisation Steps
  float Stack[Max];
  int Top=-1;
  char Ch;
  do
  {
     cout<<"P:Push O:Pop S:Show Q:Quit ";cin>>Ch;
     switch(Ch)
     {
          case 'P':Push(Stack,Top);break;
          case 'O':Pop(Stack,Top);break;
          case 'S':StackDisp(Stack,Top);break;
     }
  }
  while (Ch!='Q');
}
```

Queue

It is a non-primitive linear data structure in which insertion and deletion of elements take place from two opposite ends rear and front respectively. It is also known as FIFO (First In First Out) data structure.

//Static Circular Queue (Queue implemented using Array)

```
const int Max=10;
struct Passenger
{
   int Pno;char Name[20];
};

void Qinsert(Passenger Q[],int &R,int F)
{
  if ((R+1)%Max!=F)
  {
    R=(R+1)%Max;
    cout<<"Pno :";cin>>Q[R].Pno;
    cout<<"Name:";gets(Q[R].Name);
  }
  else
    cout<<"Passenger Queue is Full!"<<endl;
}
void Qdelete(Passenger Q[],int R,int &F)
{
  if (R!=F)
  {
    F=(F+1)%Max;
    cout<<Q[F].Pno<<":"<<Q[F].Name<<" removed..."<<endl;
  }
  else
    cout<<"Passenger Queue is empty!"<<endl;
}
void Qdisplay(Passenger Q[],int R,int F)
{
  int Cn=(F+1)%MAX;
  while (Cn!=R)
  {
    cout<<Q[Cn].Pno<<":"<<Q[Cn].Name<<endl;
    Cn=(Cn+1)%Max;
  }
}
void main()
{ //Initialisation Steps
  Passenger Que[Max]; int Rear=0,Front=0;
  char Ch;
  do
  {
     cout<<"[I]Insert [D]Delete [S]Show [Q]Quit ";cin>>Ch;
     switch(Ch)
     {
           case 'I':Qinsert(Que,Rear,Front);break;
           case 'D':Qdelete(Que,Rear,Front);break;
           case 'S':Qdisplay(Que,Rear,Front);break;
     }
  }
  while (Ch!='Q');
}
```

INFIX, POSTFIX and PREFIX notations

INFIX notation: An expression is said to be in INFIX notation if the operator is in between the operands. For example: A + B is in INFIX notation.

POSTFIX notation: An expression is said to be in POSTFIX notation if the operator is after the operands. For example: A B + is in POSTFIX notation.

PREFIX notation: An expression is said to be in PREFIX notation if the operator is before the operands. For example: + A B is in PREFIX notation.

INFIX to POSTFIX conversion

The following conversion logic will work only for the INFIX expression, which is fully parenthesized according to **BEDMAS** (**B**rackets, **E**xponents, **D**ivide, **M**ultiply, **A**dd, **S**ubtract) rule.

Order of operations

Order	Operator	Remarks
1	()	
2	^	
3	* or /	Whichever occurs first
4	+ or -	Whichever occurs first

Conversion Logic

1. If Operator, PUSH to stack
2. If Operand, Output as POSTFIX
3. If), POP from stack and output as POSTFIX

Example1 `A+B*C=(A+(B*C))`

INFIX	STACK	POSTFIX
(		
A		A
+	+	A
(	+	A
B	+	A B
*	+ *	A B
C	+ *	A B C
)	+	A B C *
)		A B C * +

Example3 `P*Q-R/S=((P*Q)-(R/S))`

INFIX	STACK	POSTFIX
(		
(		
P		P
*	*	P
Q	*	P Q
)		P Q *
-	-	P Q *
(	-	P Q *
R	-	P Q * R
/	- /	P Q * R
S	- /	P Q * R S
)	-	P Q * R S /
)		P Q * R S / -

Example2 `A-B+C/D=((A-B)+(C/D))`

INFIX	STACK	POSTFIX
(		
(		
A		A
-	-	A
B	-	A B
)		A B -
+	+	A B -
(	+	A B -
C	+	A B - C
/	+ /	A B - C
D	+ /	A B - C D
)	+	A B - C D /
)		A B - C D / +

Example4 `S-T+U=((S-T)+U)`

INFIX	STACK	POSTFIX
(		
(		
S		S
-	-	S
T	-	S T
)		S T -
+	+	S T -
U	+	S T - U
)		S T - U +

Evaluation of expression in POSTFIX notation

Evaluation Logic

1. If Operand, PUSH to stack
2. If Operator,
   (a) Op2=POP from Stack
   (b) Op1=POP from Stack
   (c) Operate Op1 and Op2
   (d) PUSH the result back in Stack
3. At the end of expression, POP the final result from the stack

Example 1 `2 10 + 5 2 - *`

POSTFIX	Steps	STACK
2	PUSH	2
10	PUSH	2 10
+	POP,POP,Operate,PUSH	12
5	PUSH	12 5
2	PUSH	12 5 2
-	POP,POP,Operate,PUSH	12 3
*	POP,POP,Operate,PUSH	36

Final Result 36

Example 2 **T F AND T F NOT AND OR**

POSTFIX	Steps	STACK
T	PUSH	**T**
F	PUSH	**T F**
AND	POP,POP,Operate,PUSH	**F**
T	PUSH	**F T**
F	PUSH	**F T F**
NOT	POP,Operate,PUSH	**F T T**
AND	POP,POP,Operate,PUSH	**F T**
OR	POP,POP,Operate,PUSH	**T**

Final Result **T**

Example 3 20,5,/,5,2,3,^,*,-

POSTFIX	Steps	STACK
20	PUSH	20
5	PUSH	20 5
/	POP,POP,Operate,PUSH	4
5	PUSH	4 5
2	PUSH	4 5 2
3	PUSH	4 5 2 3
^	POP,POP,Operate,PUSH	4 5 8
*	POP,POP,Operate,PUSH	4 40
-	POP,POP,Operate,PUSH	-36

Final Result -36

Example 4 50,16,2,4,*,/,-

POSTFIX	Steps	STACK
50	PUSH	50
16	PUSH	50 16
2	PUSH	50 16 2
4	PUSH	50 16 2 4
*	POP,POP,Operate,PUSH	50 16 8
/	POP,POP,Operate,PUSH	50 2
-	POP,POP,Operate,PUSH	48

Final Result 48