Example 1	Example 2

//Declaration of variables
int A, B, C, S;
//Inputting content in variables
cin>>A>>B>>C;
//To find sum of all variables
S=A+B+C;
cout<<S<<endl;
	//Declaration of variables
	int A1,A2,A3;
	//Inputting content in variables
	cin>>A1>>A2>>A3;
	//To find and display biggest out of 3 variables
	if (A1>A2 && A1>A3)
	cout<<A1<<endl;
	else if (A2>A1 && A2>A3)
	cout<<A2<<endl;
	else
	cout<<A3<<endl;

Arrays

Look at the above Examples, Example 1 is for accepting three values from the user & displaying sum of them whereas Example 2 is for accepting three values & displaying biggest value out of them. Think if this requirement grows to large number of values, it will require an additional effort of coding each variable for each operation explicitly. That is the reason, we require arrays, which allows to store multiple values in a single variable name and its each value is identified by its position (known as a subscript). Array is a **homogeneous collection of data.** Array is also known as a **subscripted variable** (or a variable with a subscript, in C++ subscripts range from 0 to N-1). It allows us to write generic code, which does not change with the variation in size of the array.

Declaration of arrays:

Syntax:

```
<Data Type> <Array Name>[<Size>];
```

Examples:

```
int A[10],N;      //Valid
float B[25];      //Valid
double C[5];      //Valid
cin>>N;
int D[N];         //Invalid 1
int E[];          //Invalid 2
```

Note: Invalid 1 - While writing size of the array, a variable is not allowed in the declaration.
Invalid 2 - Writing size of the array is a must in the declaration.

Initialization of array:

Syntax:

```
<Data Type> <Array Name>[<Size>]={<Value1>,<Value2>,..,<ValueN>};
```

Examples:

```
int A[4]   ={12, 45, 23, 13};                  //Valid 1
float B[8] ={1.25, 23, 65.8, 999.12};          //Valid 2
double C[] ={23000.50, 98100, 120050.20};      //Valid 3
int D[3]   ={10, 20, 40, 80};                  //Invalid 1
int E[4]   ={,, 2, 5};                         //Invalid 2
```

Note: Valid 1 - Assigns values 12 to A[0], 45 to A[1], 23 to A[2] and 13 to A[3]
Valid 2 - Assigns values 1.25 to B[0], 23 to B[1], 65.8 to B[2], 999.12 to B[3] and the remaining elements B[4], B[5], B[6] and B[7] will be assigned by zero.
Valid 3 - Assigns values 23000.50 to C[0], 98100 to C[1] and 120050.20 to C[2]. In this case size of the array will be automatically allocated as 3

Invalid 1 - Too many initializers i.e. Values are more than the size of array.
Invalid 2 - Expression Syntax error i.e. Null Values are not allowed in initialization.

Now, let us rewrite the Examples 1 and 2 with the help of arrays.

Example 1a

```
//Declaration of Array and Sum
int A[3], Sum=0;

//Inputting content in Array elements
for (int I=0;I<3;I++)
  cin>>A[I];

//To find sum of all the values of Array
for (I=0;I<3;I++)
  Sum+=A[I];
cout<<Sum<<endl;
```

In the above example, if it is required to add values stored in 200 size array, we need to simply replace the size in declaration and for loops.

Example 2a

```
//Declaration of variables
int A[5],Big;
//Inputting content in variables
for (int I=0;I<5;I++)
  cin>>A[I];
//To find Biggest Value
Big=A[0];    //Assume very first element as Big
for (I=1;I<5;I++)
  if (Big<A[I])
    Big=A[I];//Reassign Big, if required
cout<<"Biggest Value:"<<Big<<endl;
```

In the above example, if it is required to find biggest value stored in 500 size array, we need to simply replace the size in declaration and for loops.

Passing Array to function

An Array can be passed to a function as a single component. We do not require to pass individual elements explicitly. Moreover, array is always passed to a function as a reference parameter and so the changes done in array identifier acting as a formal parameter is reflected in the array identifier acting as a actual parameter.

Example 3a

```
const int Max=5;
void Enter(int A[Max])
{ //Function to enter content of an array
  for (int I=0;I<Max;I++) cin>>A[I];
}
void Display(int A[Max])
{ //Function to display content of an array
  for (int I=0;I<Max;I++)
    cout<<A[I]<<":";
  cout<<endl;
}
void Reverse(int A[Max])
{ //Function to display reverse of an array
  for (int I=Max-1;I>=0;I--)
    cout<<A[I]<<":";
  cout<<endl;
}
void main()
{
  int P[Max];
  //Inputting content in Array elements
  Enter(P);
  //To display the content of Array
  Display(P);
  //To display the content of Array
  Reverse(P);
}
```

Example 3b(Same as 3 a with Function Prototype)

```
const int Max=5;
void Enter(int []);
void Display(int []);
void Reverse(int []);
void main()
{ int P[Max];
  //Inputting content in Array elements
  Enter(P);
  //To display the content of Array
  Display(P);
  //To display the content of Array
  Reverse(P);
}
void Enter(int A[Max])
{
  for (int I=0;I<Max;I++) cin>>A[I];
}
void Display(int A[Max])
{
  for (int I=0;I<Max;I++)
    cout<<A[I]<<":";
  cout<<endl;
}
void Reverse(int A[Max])
{
  for (int I=Max-1;I>=0;I--)
    cout<<A[I]<<":";
  cout<<endl;
}
```

Note: While writing actual parameters for array, it will be invalid to mention square braces [ ], write only name of the array. In the Examples 3a and 3b, function calls `Enter(P); Display(P); and Reverse(P);` have P as the actual parameter of type integer array.	**Note:** While writing function prototype containing array as formal parameter, we can simply mention Data type followed by a set of square braces [ ]. As in the Example 3b, we used `int [ ]`

The functions used in the above programs are capable of accepting an array of size 5 only, unless we change the value of Max in const declaration. As a programmer, we should try to write codes as generic as possible. The following examples illustrate functions with flexible size array. As an exercise, you can rewrite Example 3 also in generic way.

Example 4: To find avg. marks of sections

```
void Enter(float [],int );
float Average(float [],int );
void main()
{
  float A[6],B[8],C[5],AVG;
  cout<<"Marks of Sec-A"<<endl;
  Enter(A,6);
  AVG=Average(A,6);
  cout<<"Sec-A Average"<<AVG<<endl;
  cout<<"Marks of Sec-B"<<endl;
  Enter(B,8);
  AVG=Average(B,8);
  cout<<"Sec-B Average"<<AVG<<endl;
  cout<<"Marks of Sec-C"<<endl;
  Enter(C,5);
  AVG=Average(C,5);
  cout<<"Sec-C Average"<<AVG<<endl;
}
//Function to enter content of an array
void Enter(float M[],int N)
{
  for (int I=0;I<N;I++)
  {
    cout<<"Roll No ["<<I<<"]";
    cin>>M[I];
  }
}

//Function to get Avg. of values in the array
float Average(float M[],int N)
{
  float Sum=0;
  for (int I=0;I<N;I++)
    Sum+=M[I];
  return Sum/N;
}
```

Note: In the above example, all three arrays A, B and C are of different size and passed to same Enter() and Average() functions.

Example 5: To count and display all odd values

```
void Enter(int [],int );
int CountOdds(int [],int );
void main()
{
  int P[6],Q[8],R[5],ODD;
  cout<<"Team P Scores:"<<endl;
  Enter(P,6);
  ODD=CountOdd(P,6);
  cout<<"Team P Odds:"<<ODD<<endl;
  cout<<"Team Q Scores:"<<endl;
  Enter(Q,8);
  ODD=CountOdd(Q,8);
  cout<<"Team Q Odds:"<<ODD<<endl;
  cout<<"Team R Scores:"<<endl;
  Enter(R,5);
  ODD=CountOdd(R,5);
  cout<<"Team R Odds:"<<ODD<<endl;
}
//Function to enter content of an array
void Enter(int P[],int N)
{
  for (int I=0;I<N;I++)
  {
    cout<<"Player ["<<I<<"]";
    cin>>P[I];
  }
}
//Function to get Avg. of values in the array
int CountOdds(int P[],int N)
{
  int Count=0;
  for (int I=0;I<N;I++)
    if (P[I]%2!=0)
    {
      cout<<P[I]<<":";
      Count++;
    }
  return Count;
}
```

Note: In the above example, all three arrays P, Q and R are of different size and passed to same Enter() and CountOdd() functions.

Linear Search: It is a logic to search for a value from an array by checking the elements one after the other starting from the very first element of the array.

Example 6: To search for a particular employee code from an array containing employee codes of all the employees working in a company.

```
void Register(long [],int );
void Search(long [],int ,int );

void main()
{
  long Eno[6],Enos;
  cout<<"Enter all Emp.Nos:"<<endl;
  Register(Eno,6);
  cout<<"Eno to be searched:";
  cin>>Enos;
  Search(Eno,6,Enos);
}

//Function to enter the content of an array
void Register(long E[],int N)
{
  for (int I=0;I<N;I++)
  {
    cout<<"["<<I<<"] Eno?" ;
    cin>>E[I];
  }
}
//Function to search for a
//particular eno (employee number)
void Search(long E[],int N,int Eno)
{
  int Found=0;
  for (int I=0;I<N && Found==0;I++)
    if (E[I]==Eno)
      Found=1;
  if (Found==1)
    cout<<Eno<<" is Found"<<endl;
  else
    cout<<Eno<<" is not Found"<<endl;
}
```

Note:
In the above example, Search() function is to find whether a particular Eno is present in the array or not and so as soon as it is found, value of Found variable is changed to 1, and the loop terminates.
Look at the condition in the loop `I<N && Found==0` the loop terminates either the value of `I` becomes `N` or `Found` becomes `1`

Example 7: To count the presence of a particular citycode from an array containing citycodes in an array.

```
void Register(int [],int );
void Search(int [],int ,int );

void main()
{
  int CCode[6],CCS;
  cout<<"Enter all CityCodes:"<<endl;
  Register(CCode,6);
  cout<<"CityCode to be searched:";
  cin>>CCS;
  Search(CCode,6,CCS);
}

//Function to enter the content of an array
void Register(int C[],int N)
{
  for (int I=0;I<N;I++)
  {
    cout<<"["<<I<<"] CCode?" ;
    cin>>C[I];
  }
}

//Function to count the presence of a Ccode
void Search(int C[],int N,int CS)
{
  int Found=0;
  for (int I=0;I<N;I++)
    if (C[I]==Ens)
      Found++;

  cout<<CS<<" is present ";
  cout<<Found<<" Times"<<endl;
}
```

Note:
In the above example, Search() function is to count the number of times a particular City Code (i.e. CS) is present in the array and so the loop continues till the last elements. Every time, when the element matches with the City Code (i.e. CS) to be searched, Found gets incremented.
Look at the condition in the loop `I<N`, the loop terminates only when the value of `I` becomes `N`