Al-Albayt University
Computer Science Department

C++ Programming 1 (901131)

Coordinator:

Dr. Ashraf Al-ou'n

الفصل الدراسي الأول 2018 -2019

جامعة آل البيت
Al albayt University

3 - Functions

Modular programming with Functions

- Large program can be constructed from smaller pieces (modules) which are more manageable than the original program.a
- Modules in C++ are functions and classes.
- Function is a group of statements that is executed when it is called from some point of the program.
- Function can be called multiple times in a program.

Advantages of modular programming

- Functions in C++, make programs
  - Easier to understand
  - Easier to change
  - Easier to write
  - Easier to test
  - Easier to debug
  - Easier for teams to develop

Pre-packaged and Programmer defined functions

- Pre- packaged functions are provided as part of the C++ programming environment
- available in standard C++ library which provides rich collection of functions for performing:
  - Common mathematical calculations
  - String manipulations
  - Character manipulations
  - Input/output, error checking and many other useful operations

- Programmer defined functions
  - Programmer can write functions to define specific tasks
  - Could be used at many points in a program
  - Actual statements defining the function are written only once
  - These statements are hidden from other functions

Function Libraries

- Predefined functions are found in libraries
- The library must be "included" in a program to make the functions available
- An include directive tells the compiler which library to include.
- To include the math library containing **`sqrt()`**:

```
#include <cmath>
```

Math library functions

- Allow the programmer to perform common mathematical calculations
- Example: `cout << sqrt( 49.0);`
  - Calls the sqrt (square root) function and print 7
  - The sqrt function takes an argument of type double and returns a result of type double, as do all functions in the math library
- Function arguments can be
  - Constants: `sqrt(4);`
  - Variables: `sqrt(x);`
  - Expressions: `sqrt(3y + 6);`

Commonly used math library functions

Function	Description	Example
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` returns `10.0` `ceil(-9.8)` returns `-9.0`
`cos(x)`	trigonometric cosine of x (x in radians)	`cos(0.0)` returns `1.0`
`exp(x)`	exponential function ex	`exp(1.0)` returns `2.71828` `exp(2.0)` returns `7.38906`
`fabs(x)`	absolute value of x	`fabs(5.1)` returns `5.1` `fabs(-8.76)` returns `8.76`
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` returns `9.0` `floor(-9.8)` returns `-10.0`
`fmod(x,y)`	remainder of x/y as a floating-point number	`fmod(13.657,2.333)` returns `1.992`
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` returns `1.0` `log(7.389056)` returns `2.0`
`log10(x)`	logarithm of x (base 10)	`log10(10.0)` returns `1.0` `log10(100.0)` returns `2.0`
`pow(x,y)`	x raised to power y (xy)	`pow(2, 7)` returns `128` `pow(9,0.5)` returns `3`
`sin(x)`	trigonometric sine of x (x in radians)	`sin(0.0)` returns `0`
`sqrt(x)`	square root of x	`sqrt(900.0)` returns `30.0` `sqrt(9.0)` returns `3.0`
`tan(x)`	trigonometric tangent of x (x in radians)	`tan(0.0)` returns `0`

Example1


```
#include <iostream>
#include <cmath>  using
namespace std;

int main(){

      double x;
      cout<<"enter a number to find its square root: ";
      cin>>x;
      cout<<"square root = "<< sqrt(x);
}
```

enter a number to find its square root: 25
square root = 5

Example 2


```
#include <iostream>
#include <cmath>
using namespace std;

int main(){

  double A = 6.1,B = 2.3,C = -8.14;

  cout<<"fmod("<<A<<","<<B<<")="<<fmod(A,B)<<endl;

  cout<<"pow(ceil("<<C<<"),2)="<<pow(ceil(C),2)<<endl;
}
```

fmod(6.1,2.3) = 1.5
pow(ceil(-8.14),2) = 64

Random number generator

- **`rand()`**: returns a random **integer** between **0** and **32767**

```
int main()
{
   int x;
   for(int i=1; i<=100; i++)
   {
      x = rand();
      cout << x << endl;
   }
}
```

Generate 12 random numbers between 0 and 9


```
int main()
{
   int  x;
  for(int i=1;i<=12;i++)
  {
      x = rand()%10;
      cout << x << endl;
  }
}
```

```
1
7
4
0
9
4
8
8
2
4
5
5
```

Generate 15 random numbers between 1 - 10


```
int main()
{
  int x;
  for( int i=1 ; i <= 15 ; i++)
  {
     x = rand()%10 + 1;
     cout << x << endl;
  }
}
```

```
2
8
5
1
10
5
9
9
3
5
6
6
2
8
2
```

Generate 10 random numbers between 5 - 7


```
int main()
{
  int x;
  for( int i=1 ; i<=10 ; i++)
  {
     x = rand()%3 + 5;
     cout<<x<<endl;
  }
}
```

```
7
7
6
6
7
6
5
5
6
7
```

Using **srand** function


```
int main(){

   int x,seed;
   cout <<"Enter a seed:";
   cin >>seed;
   srand(seed);

   for(int i=1 ; i<=5 ; i++)
   {
       x = rand();
       cout <<x<< endl;
    }
}
```

```
Enter a seed:5
54
28693
12255
24449
27660
```

```
Enter a seed:11
74
27648
21136
4989
24011
```

Generate 1200 random numbers between 1-6, then count the occurrence of each number


```
int main(){
   int x, c1=0, c2=0, c3=0, c4=0, c5=0, c6=0;
   for( int i = 1; i <= 1200 ; i++){
       x = rand()%6 + 1;
       if (x == 1) c1++;
       if (x == 2) c2++;
       if (x == 3) c3++;
       if (x == 4) c4++;
       if (x == 5) c5++;
       if (x == 6) c6++;
       }
       cout<< 1 <<":"<< c1<<endl;
       cout<< 2 <<":"<< c2<<endl;
       cout<< 3 <<":"<< c3<<endl;
       cout<< 4 <<":"<< c4<<endl;
       cout<< 5 <<":"<< c5<<endl;
       cout<< 6 <<":"<< c6<<endl;
}
```

1 : 190
2 : 188
3 : 214
4 : 207
5 : 200
6 : 201

Generate 1200 random numbers between 1 and 6, then print the count of even and odd numbers

```
int main(){
   int x , even = 0, odd = 0;
   for( int i = 1; i <= 1200 ; i++){
       x = rand() % 6 + 1;
       switch( x ) {
             case 1: case 3: case 5:
                   odd++;
                   break;
             case 2: case 4: case 6:
                   even++;
       }
   }
   cout<<"Even count is "<<even<<endl;
   cout<<"Odd count is "<<odd<<endl;
}
```

Programmer defined functions

- Functions modularize a program
- Function can be called multiple times
  - Software reusability
- Local variables
  - Known only in the function in which they are defined
  - All variables declared in function definitions are local variables
- Parameters
  - Local variables passed to function when called
  - Provide outside information

Programmer defined functions

- Each program consists of a function called **main**
- programmer can write there own customized functions
- Programmer defined functions have 2 important components:
  - Function definition
    - Describes how function does its task
    - Can appear before or after the function is called
  - Function prototype

Function Definition


```
return-value-type  function-name( parameter-list )
   {
       declarations and statements
   }
```

- **Function-name:** any valid identifier
- **Return-value-type:**
  - The data type of the result returned from the function.
  - Return value type **void** indicated that the function does not return a value

- **Parameter-list**
  - comma-separated list of the arguments
  - contains the declarations of the parameters received by the function when it is called
  - If the function does not receive any values, **parameter-list** is **void** or simply left **empty**

Example


```
int square(int y){
    return y * y;
}
```

- **`return`** keyword
  - Format **`return`** ***`expression;`***
  - Returns the value calculated by the afunction
  - Returns data, and control goes to function's caller
    - If no data to return, use **`return;`**
  - Function ends when reaches right brace
    - Control goes to caller

- Functions cannot be defined inside other functions

Function Prototype

- Must appear in the code before the function can be called
- The compiler use function prototypes to validate function call

- Format:

```
Return-type Function_Name(Parameter_List);
```

- Function prototype tells the compiler
  - The function's name
  - Type of data returned by the function (void if return nothing)
  - Number of parameters the function accepts to receive
  - Types of parameters
  - The order in which these parameters are expected

Function prototype

- Function prototype is not required if the function definition appears before the first use function's first use in the program

- Prototype must match function definition
  - Function prototype

```
double max(double, double, double);
```

  - Definition

```
double max(double x, double y, double z)
{
        …
}
```

Forms of functions

- Functions that take inputs and returns output:
  - `double square(double)`
  - `int max( int a, int b, int c)`
- Functions that take inputs but don't return output
  - `void Printsum( int x, int y)`
- Functions that don't take any inputs but returns an output:
  - `int Read_number(void);`
- Functions that don't take and inputs and don't return any input
  - `void Print_hello(void);`

Function call

- Function is invoked by function call
- A function call specifies the function name and provides information (as arguments) that the called function needs
- Functions called by writing
  - `functionName(argument);`

  or
  - `functionName(argument1, argument2, …);`
- The argument can be a constant, variable or expression

Example program


```
int square( int );          //function prototype

int main()
{
      for ( int x = 1; x <= 7; x++ )
            cout <<square(x)<<"  ";       //function call
      cout << endl;
}
int square(int y) {         //function definition
      return y * y;
}
```

1	4	9	16	25	36	49

Example Program


```
double maximum( double, double, double ); //function prototype

int main() {
   double n1;
   double n2;
   double n3;
   cout << "Enter three floating-point numbers:";
   cin >> n1 >> n2 >> n3;
   cout << "Maximum is: "<< maximum( n1,n2,n3 )<<endl;
}

double maximum( double x, double y, double z )   {
   double max = x;
   if ( y > max )   max =
       y;
   if ( z > max )   max =
       z;
   return max;
}
```

```
Enter three floating-point numbers:7.1
2.9
5.5
Maximum is: 7.1
```

Function signature

- Function signature is also called simply signature
- It is the portion of a function prototype that includes
  - name of function
  - parameters (types of its argument)
- Example

```
double maximum( double, double, double );
```

Function signature

Argument Correction

- Argument values that do not correspond precisely to the parameter types are converted to proper type before the function is called

- **`cout<<sqrt(4)`**
  - the compiler converts **`int`** value **`4`** to the **`double`** value **`4.0`** before the value is passed to **`sqrt`**

- Changing from **`double`** to **`int`** can truncate data e.g. 3.5 to 3

- Some of the conversions may lead to incorrect results if proper promotion rules are not followed

Promotion rules

- How types can be converted to other types without losing data
- Applies to expressions containing values of two or more data types in which, type of each value is promoted to the highest type in the expression
- Also used when the type of an argument to a function does not match the parameter type specified in the function definition
- Converting values to lower type can result in incorrect values

Promotion hierarchy for built in data types

Data types
long double
double
float
unsigned long
long
unsigned
int
unsigned short
Short
unsigned char
char
bool

Type casting

- A value can be converted to a lower type only by explicitly assigning the value to a variable of lower type by using the cast operator

- Example: `static_cast<double>(total)` // total is of type integer

  - produces a **`double`** representing the integer value of the variable total (operand in parenthesis)
  - **`double`** is higher type and **`int`** is lower type

- Converting values to lower type can result in incorrect values

Header Files

- Library header file
  - Contain function prototypes for library functions
  - Definition of various data type and constants needed by library functions
  - Examples: **<cstdlib>** , **<cmath>** **,<iostream>**.
  - Load with **#include<filename>: #include<cmath>**
- Custom header files
  - Defined by the programmer
  - Save as **filename.h**
  - Included in a program using **#include** preprocessor directive
  - Example: **#include "square.h"** //programmer defined header file

- The programmer defined header file should end with .h

Write a function that takes the length and width of a rectangle and returns the area


```
double area(double, double);

void main( ){
   double L,W,A;
   cout<<"Enter the length:";
   cin>> L;
   cout<<"Enter the width:";
   cin>> W;
   A = area(L,W);
   cout<<"The area of the rectangle is "<<A<<endl;
}
double area(double X,double Y){
   double Z;
   Z = X*Y;
   return Z;
}
```

Write a function that takes three integers and returns the maximum one


```
int max(int, int, int);

int main(){
   int x, y, z;
   cout<<"Enter 3 numbers please :";
   cin>> x >> y >> z;
   cout<<"The Maximum is "<< max(x,y,z) <<endl;
}

int max(int a,int b,int c){
   int m = a;
   if (m < b) m=b;
   if (m < c) m=c;
   return m;
}
```

```
Enter 3 numbers please :5
9
6
The Maximum is 9
```

Write a function that takes an integer and returns **true** if it is a prime number and **false** otherwise

```
bool prime(int);
int main(){
   int x;
   cout << "Enter a number please :";
   cin >> x;
   if (prime(x))
       cout<< x <<" is a prime number\n";
  else
       cout << x <<" is not a prime number\n";
}
bool prime(int a){
bool p = true;
   for(int i=2 ;i < a); i++)
       if (a%i == 0){ p = false;  break; }

   return p;}
```

Write a program that uses a function to calculate and print the sum of two numbers.


```
void print_sum( int,int );
int main() {
  int x,y;
  cout<<"Enter two numbers please :";
  cin>>x>>y;
  print_sum(x,y);
  print_sum(y,x);
  print_sum(5,7);
}

void print_sum( int x , int y){
  int s;
  s = x + y;
  cout<<x<<"+"<<y<<" = "<<s<<endl;
}
```

Write a function that reads two numbers and another function to prints their sum


```
void read_number();
void print_sum(int,int);
int main(){
   read_number();
}
void read_number(){
   int A , B;
   cout<<"Enter two numbers please ";
   cin>>A>>B;
   print_sum(A , B);
}

void print_sum(int x,int y){
   int s;
   s = x + y;
   cout<<x<<"+"<<y<<" = "<<s<<endl;
}
```

Write a function that prints one of three messages randomly


```
void print_message();

int main(){
  for(int i = 1 ; i <= 10 ; i++)
     print_message();
}

void print_message() {
  int i = rand()%3 + 1;
  if(i == 1) cout<<"Hello\n";
  if(i == 2) cout<<"Hi\n";
  if(i == 3) cout<<"Welcome\n";
}
```

Scope Rules

- Scope
  - Portion of program where identifier can be used

- Scopes for an identifier are
  - Function scope
  - File scope
  - Function prototype scope
  - Block scope
  - Class scope
  - namespace scope

Scope Rules

- Function scope
  - Can only be referenced inside function in which they appear, can not be referenced outside function body
- File scope
  - Defined outside a function, known in all functions
  - Global variables, function definitions and prototypes all have file scope
- Function-prototype scope
  - Names in function prototype are optional, only type required
  - Compiler ignores the name if used in parameter list of function-prototype
  - In a single prototype, name can be used once
  - Identifier used in function prototype can be reused elsewhere in the program

Scope Rules

- Block scope
  - Identifiers declared inside the block
  - Begins at declaration, ends at right brace **}** of the block
    - Can only be referenced in this range
  - Local variables, function parameters
  - In case of nested blocks and identifiers in inner and outer block have same name
    - Identifier in the outer block is hidden until the inner block terminates

Unary Scope Resolution Operator

- Unary scope resolution operator **(::)**
  - Access global variable if local variable of same name is in scope
  - Not needed if names are different
  - Use **::variable**

```
y = ::x + 3;
```

  - Can not be used to access a local variable of same name in an outer block
  - Good to avoid using same names for local and global variables

Example Program

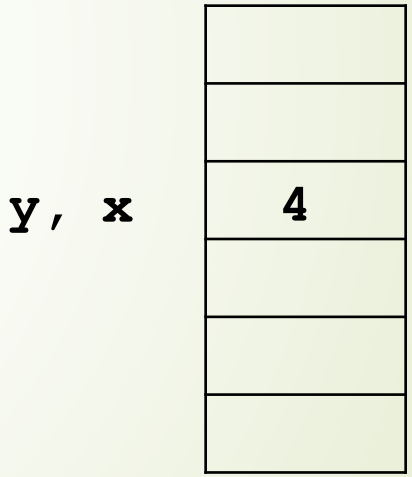
```
int x = 10;
int f1(int);
int main(){
    cout << x << endl;
    int x = 15;
    cout << x << endl;
    Cout << ::x << endl;
    if (true){ int x = 5; cout << x << endl; }
    cout << x << endl;
    cout << f1(x) << endl;
    cout << ::x << endl;
}
int f1(int a){
        cout << x << endl;
        x = x - a;
        int x = 13;
        cout << x << endl;
        return x + a;
}
```

```
10
15
10
5
15
10
13
28
-5
```

Reference Variables (Alias)

- A reference variable is an alias (another name) for an already existing variable
- Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
- Example

```
int x = 4;
int &y = x;
```



- **y** is an alias for variable **x**
- **x** and **y** both refer to same variable (same memory location)

Example


```
int main()
{

   int x;
   int &y = x;          //y is an alias for x

   x = 5;
   cout<<"x = "<< x <<"\t"<<"y = "<< y << endl;

   y = 7;
   cout<<"x = "<< x <<"\t"<<"y = "<< y << endl;
}
```

```
x = 5   y = 5
x = 7   y = 7
```

Call By Value and Call By Reference

- When passing a parameter by value, the function receives a ***copy*** of the variable, so it can't modify the variable

```
void function(int var);
```

- When passing a parameter by reference, the function receives a ***reference*** to the variable (**not a copy of it)**, so it can modify the variable

```
void function(int &var);
```

Example


```
void square1(int);
void square2(int &);
int main() {
  int x,y;
  cout<<"Enter a number:";
  cin>>x;
  square1(x);
  cout<< "calling x by value, x = " << x <<endl;
  square2(x);
  cout<< "calling x by ref, x = " << x <<endl;
}
void square1(int a) { // call by value, int a = x
  a = a * a;}
  void square2(int  &a){ //call by reference, int &a = x
  a = a * a;
}
```

```
Enter a number: 10
calling x by value, x = 10
calling x by ref, x = 100
```

Example


```
void read(int &, int);
int main()
{
      int x = 0, y = 0;
      read(x , y);
      cout <<"x = " << x << endl;
      cout <<"y = " << y << endl;
}

void read(int &a , int b){
      cout <<"Enter two numbers:\n";
      cin >> a >> b;
}
```

Enter two numbers:
6
4
x = 6
y = 0

Static Variables

- Local variables in function declared with keyword **`static`**
- Keeps value between function calls
- Only known in own function
- Static local variables retain their values when function is exited

Example


```
void f( );

int main()
{
      f();
      f();
      f();
}
void f()
{
      static int a = 1;
      int b = 1;
      cout <<"a = "<< a++ <<"\tb = "<< b++ <<endl;
}
```

```
a = 1     b = 1
a = 2     b = 1
a = 3     b = 1
```

Default arguments

- default argument provide a value to be used instead of omitted parameters in function call
- Default value is automatically used by the compiler and passed in to the function
- If not enough parameters, rightmost go to their defaults
- Default argument should be specified in the first occurrence of the function name, typically prototype

Eample


```
int f(int x=1 , int y=2 , int z=4 );

int main()
{
   cout << f() <<endl;
   cout << f(3) <<endl;
   cout << f(3,4) <<endl;
   cout << f(6,8,1) <<endl;
}

int f(int x, int y , int z)
{
   return x + y + z;
}
```

```
7
9
11
15
```

Recursive functions

- *A recursive function* is a function that makes a call to itself
- May result in a infinite recursion
- Typical problems solved using recursion
  - Factorial
  - Fibonacci series
  - Sum between X, Y

Recursive Factorial


```
int fact(int);

int main(){
   int x;
   cout << "Enter a number:";
   cin >> x;
   cout << x <<"! = " << fact(x) <<endl;
}

int fact(int n){
   if (n == 1)
      return 1;
   else
      return n*fact(n-1);
}
```

```
Enter a number:4
4! = 24
```

Fibonacci Series

- 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , …
- $fib\ (0)\ = 0$
- $fib\ (1)\ = 1$
- $fib\ (n) = fib\ (n - 1\ ) + fib(n - 2)$

Fibonacci Series


```
int fib(int);

int main(){
  int x;
  cout << "Enter a number:";
  cin >> x;
  cout << "Fibonacci("<<x<<") = "<< fib(x) << endl;
}

int fib( int n){
  if(n == 0 || n == 1)
     return n;
  else
     return fib(n-1) + fib(n-2);
  }
```

```
Enter a number:7
Fibonacci(7) = 13
```

Sum of numbers from 1 to n


```
int sum(int);

void main() {
  int x;
  cout<<"Enter end number: ";
  cin>>x;
  cout<<"Sum of the series = "<<sum(x)<<endl;
}

int sum(int n) {
  if (n == 1)
     return 1;  else
     return n + sum(n - 1);
}
```

```
Enter end number: 3
Sum of the series = 6
```

Sum of numbers from X to Y


```
int sum(int , int);

int main(){
   int x, y;
   cout <<"Enter first number:";
   cin >> x;
   cout <<"Enter second number:";
   cin >> y;
   cout <<"Sum of the series = "<< sum(x,y) <<endl;
}

int sum (int a ,int b){
   if (a == b)
       return b;
   else
       return b + sum(a , b-1);
}
```

```
Enter first number:5
Enter second number: 7
Sum of the series = 18
```

Function Overloading

- C++ enables several functions of the same name to be defined
  - as long as they have different signatures
- The C++ compiler selects the proper function to call by examining the number, types and order of the arguments in the call
- Overloaded functions are normally used to perform similar operations that involve different program logic on different data types

Example


```
int square(int x) {
      cout << "square of integer " << x << " is ";
      return x * x;
}
double square(double y) {
      cout << "square of double " << y << " is ";
      return y * y;
}
int main(){
      cout << square(5)<< endl;
      cout << square(5.0)<<endl;
}
```

```
square of integer 5 is 25
square of double 5 is 25
```

Inline Functions

- Keyword **`inline`** before function return type
  - Asks the compiler to copy code into program instead of making function call
    - Reduce function-call overhead
    - Compiler can ignore **`inline`**
  - Good for small, often-used functions
- Example

```
inline double cube(double s){ return s*s*s; }
```

Example

```
inline double cube(double s) { return s* s* s }

int main()
{

      double side;
      cout<<"Enter the side length of your cube: ";
      cin >> side;
      cout << "Volume of cube = "<<cube(side)<<endl;
}
```

04 - Arrays

Array

- Set of adjacent memory locations of the same data type.
- All of them have one name, which is the array name.
- each location has subscript number starts at zero.
- One Dimensional array

```
Data_Type array_name[size]
```

```
int a[10]
```

- Two Dimensional array

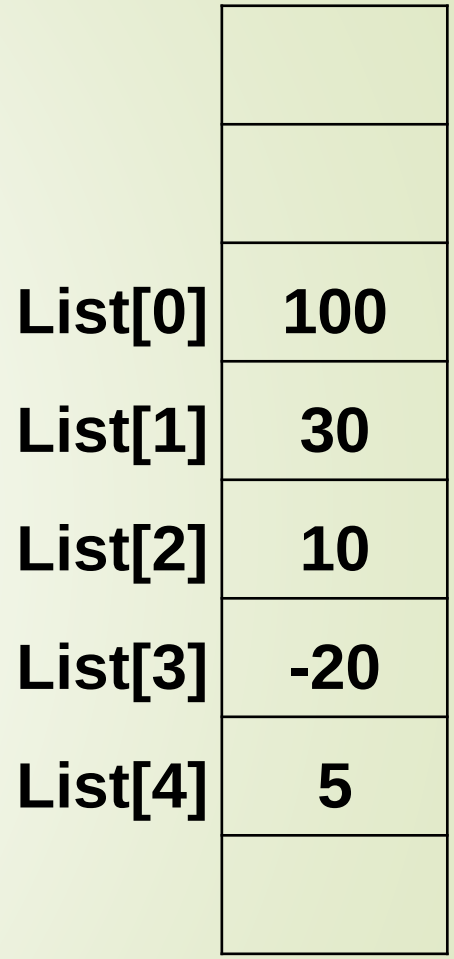
```
Data_Type array_name[Rows][Columns]
```

```
int a[3][4]
```

One Dimensional array Example


```
int List[5];
   List[0] = 100;
   List[1] = 30;
   List[2] = 10;
   List[3] = -20;
   List[4] = 5;

for(i=0; i < 5; i++)
      cout<<List[i]<<"\t";
```



100 30 10 -20 5

Definitions and initial values


```
int X[4]={4,2,3,1};
int A[10]={0};
int B[100]={1};
int C[7]={1,2,6};
int d[5];
int E[ ]={4,6,2};
```

Not accepted Statements

```
int A[4]={4,2,4,5,2};
int E[ ];
```

Read and print an Array


```
int main() {
        int A[3] = {10,20,15};
        cout<<"Print Array:";
        for (int i=0 ; i<3 ; i++)
              cout << A[i] << '\t';
        cout<<endl;

        for (int i=0 ; i<3 ; i++) {
              cout << "Enter A["<<i<<"]:";
              cin >> A[i];
        }
        cout<<"Print Array:";
        for (int i=0 ; i<3 ; i++)
           cout << A[i] << '\t';
}
```

```
Print Array: 10   20   15
Enter A[0]: 3
Enter A[1]: 7
Enter A[2]: 1
Print Array: 3    7    1
```

Write a program that inputs an array elements then prints the count of elements greater than 10


```
const int S = 10;
int main() {
        int A[S];
        for(int i=0 ; i<S ; i++) {
                cout << "Enter A[" << i << "]:";
                cin >> A[i];
        }
        int count = 0;
        for (int i = 0; i < S; i++) {
                if (A[i] > 10)
                        count++;
        }
        cout<< "No of element > 10 = "<<count <<endl;
}
```

Multiplying elements in even positions


```
const int S = 5;
int main() {
  int A[S];
  for(int i=0; i < S; i++) {
    cout<<"Enter A["<<i<<"]:";
    cin>>A[i];
  }

  long even = 1;
  for (int i = 0; i < S; i++)
    if(i % 2 == 0)
     even *= A[i];

  cout<<"Multiplying elements in Even positions = "<<even;
}
```

```
Enter A[0]:2
Enter A[1]:6
Enter A[2]:1
Enter A[3]:11
Enter A[4]:3
Multiplying elements in Even positions = 6
```

Find the Maximum element


```
const int Size = 5;  int
main() {

        int V[Size];
        cout<<"Enter 5 numbers to find the maximum\n";

        for(int i = 0; i < Size; i++)
              cin >> V[i];

        int Max = V[0];
        for(int i = 1; i < Size; i++)
              if (Max < V[i])
                    Max = V[i];
        cout<<"Max = "<< Max <<endl;
}
```

```
Enter 5 numbers to find the maximum
14
11
23
7
19
Max = 23
```

Find the Maximum element


```
const int Size = 5;

int main() {
   int V[Size];
   cout<<"Enter 5 numbers to find the maximum\n";
   for(int i = 0; i < Size; i++)
       cin >> V[i];
    int Max = V[0];
    int Pos = 0;

    for(int i=1; i< Size; i++)
       if(Max < V[i]){
          Max = V[i];
          Pos = I;
       }
    cout<<"Max= "<<Max<<" at position "<<pos<<endl;
}
```

```
Enter 5 numbers to find the maximum
14
11
23
7
19
Max = 23 at position 2
```

Array Search using Linear Search


```
int main() {
        int V[5]={7,12,5,31,4}, Element, i;

        cout << "Enter the element to search for:";
        cin >> Element;

        bool Found = false;
        for(i = 0; i < 5;
        i++)
                if(Element == V[i]){
                        Found = true;
                        break;
                }
        if (Found)
                cout<<Element<<" Found at position
                "<<i<<endl;
        else
                cout<<Element<<" is not in the array \n";
}
```

Enter the element to search for:18
18 is not in the array

Enter the element to search for:5
5 Found at position 2

Swap Function


```
void swap(int &,int &);
int main(){
      int a = 2, b = 9;
      cout<<"a = "<<a<<'\t'<<"b = "<<b<<endl;
      swap(a,b);
      cout<<"\nAfter Swap\n";
      cout<<"a = "<<a<<'\t'<<"b = "<<b<<endl;
}
void swap(int &x , int &y) {
      int temp = x;
      x = y;
      y = temp;
}
```

a = 2 b = 9

After Swap

a = 9 b = 2

Array Sort with Bubble Sort


```
void swap(int &, int &);
const int Size = 5;
int main() {
  int V[Size] = {5,9,7,4,3};

  for(int i = 1 ; i < Size; i++)
    for(int j = 0 ; j < Size-i; j++)
      if(V[j] > V[j+1])
        swap(V[j], V[j+1]);
cout<<"Array after Sort:\n";
for(int i = 0; i < Size; i++)  cout<< V[i]<<'\t';
}
void swap(int &x , int &y) {
      int temp = x;
      x = y;
      y = temp;
}
```

Array after Sort:

3 4 5 7 9

5	9	7	4	3

- i = 1:
  - j = 0

5	9	7	4	3

  - j = 1

5	7	9	4	3

  - j = 2

5	7	4	9	3

  - j = 3

5	7	4	3	9

- i = 2:
  - j = 0

5	7	4	3	9

  - j = 1

5	4	7	3	9

  - j = 2

5	4	3	7	9

- i = 3:
  - j = 0

4	5	3	7	9

  - j = 1

4	3	5	7	9

- i = 4:
  - j = 0

3	4	5	7	9

```
if(V[j] > V[j+1])
      swap( V[j],V[j+1] );
```

5	4	3	2	1

- i = 1:
  - j = 0

4	5	3	2	1

  - j = 1

4	3	5	2	1

  - j = 2

4	3	2	5	1

  - j = 3

4	3	2	1	5

- i = 2:
  - j = 0

3	4	2	1	5

  - j = 1

3	2	4	1	5

  - j = 2

3	2	1	4	5

- i = 3:
  - j = 0

2	3	1	4	5

  - j = 1

2	1	3	4	5

- i = 4 :
  - j = 0

1	2	3	4	5

```
if(V[j] > V[j+1])
        swap( V[j],V[j+1] );
```

Passing Arrays to functions

- When passing an array to a function, it is passed by reference.
- No need to declare add the **&** operator

Sum of Array elements


```
int Sum(int [],int);
int main( )
{
   int A[5]={2,3,4,5,6};
   int B[4]={5,3,1,7};
   cout<<"The sum of A is "<<Sum(A,5)<<endl;
   cout<<"The sum of B is "<<Sum(B,4)<<endl;
}

int Sum(int x[],int size){
   int S = 0;
   for(int i = 0 ; i < size ; i++)
       S = S + x[i];
   return S;
}
```

```
The sum of A is 20
The sum of B is 16
```

Read and print Arrays


```
void read(int [], int);
void print(int[], int);
int main(){
   int A[5] , B[4];
   read(A,5);
   read(B,4);
   print(A,5);
   print(B,4);
}
void read(int x[], int size){
   for(int i = 0 ; i < size ; i++){
       cout<<"Enter a number please:";
       cin >> x[i];
   }
}
void print(int x[], int size){
   for(int i = 0 ; i < size ; i++)
       cout << x[i]<<" ";
}
```

Sum of two Arrays


```
void sum (int [], int [], int [], int);
void print (int [], int);
void main(){
   int a[4]={4,3,2,1};
   int b[4]={2,2,4,4};
   int c[4];
   sum(a,b,c,4);
   print(c,4);
}
void sum(int x[], int y[], int z[], int size){
   for (int i = 0 ; i < size ; i++)
      z[i] = x[i]+y[i];
}
void print(int x[] , int size){
   for(int i = 0 ; i < size ; i++)
      cout << x[i]<<" ";}
```

Two Dimensional array

- Declare a 3 by 4 array

```
int Matrix[3][4];
```

- Assign values to array Elements

```
Matrix[0][1]= 30;
Matrix[1][0]= 100;
Matrix[1][2]= 10;
Matrix[2][3]= -20;
Matrix[2][0]= Matrix[1][2];
```

	0	1	2	3
0		100		
1	30		10	
2	10			-20

- Print Elements

```
for( int i=0 ; i < 3 ; i++){
    for( int j = 0 ; j < 4 ; j++)
        cout << Matrix[i][j] << "\t";
    cout << endl;
}
```

2D Array Initialization


```
int A[2][3] = {1,2,3,10,20};
```

	0	1	2
0	1	2	3
1	10	20	0

```
int M[3][4]= {{4,30},{100,5,10},{1,7,2,-20}};
```

	0	1	2	3
0	4	30	0	0
1	100	5	10	0
2	1	7	2	-20

Example 1


```
int main() {
  int A[2][3]={ {1,2}, {4,5,6} }, B[2][3]={1,2,4,5,6};
  for(int i=0; i<2; i++){
    for(int j=0; j<3; j++)
      cout<<A[i][j]<<'\t';
    cout<<endl;
  }
  cout<<"\n\n";
  for(int i=0; i<2; i++){
    for(int j=0; j<3; j++)
      cout<<B[i][j]<<'\t';
    cout<<endl;
  }
}
```

```
1	2	0
4	5	6

1	2	4
5	6	0
```

Example 2


```
const int R = 2, C = 3;

int main() {

  int A[R][C];

  for (int i=0; i < R ; i++)
    for (int j=0; j < C ; j++) {
      cout<<"Enter
      A["<<i<<"]["<<j<<"]:";
      cin>>A[i][j];
    }
  for (int i=0; i < R; i++){
   for(int j=0; j < C ; j++)
      cout<<A[i][j]<<'\t';
   cout<<endl;
  }
}
```

```
Enter A[0][0]:1
Enter A[0][1]:6
Enter A[0][2]:7
Enter A[1][0]:3
Enter A[1][1]:5
Enter A[1][2]:2
1       6       7
3       5       2
```

Sum of 2D Array elements


```
int sum(int [][2], int);

int main(){
   int x[3][2]={6,5,4,3,2,1};
   int y[4][2]={5,4,3,2,3,4,1};
   cout<<"The sum of array x is " <<sum(x,3)<<endl;
   cout<<"The sum of array y is " <<sum(y,4)<<endl;
}
int sum(int a[][2],int r){
   int s = 0;
   for(int i = 0 ;i < r ; i++)
       for(int j = 0 ; j < 2 ; j++)
             s += a[i][j];
   return s;
}
```

```
The sum of array x is 21
The sum of array y is 22
```

Diagonal Sum


```
int Dsum(int [][3]);

int main() {

   int A[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};

   cout<<"Diagonal Sum = "<<Dsum(A)<<endl;
}

int Dsum(int a[][3]) {
   int S=0;
   for (int i=0 ; i<s ; i++)
       S += a[i][i];

   return S;
}
```

Diagonal Sum = 15

Inverse Diagonal Summation


```
const int R = 3, C= 3;

int main (){
   int A[R][C]= {{1,2,3}, {6,11,7}, {7,3,9}};
   int Dsum =0;
   for (int i=0 ; i < R ; i++)
       for (int j=0; j < C ; j++)
            if ( i+j == 2)
                 DSum += A[i][j];

   cout<<"Inverse Diagonal Sum = "<<DSum<<endl;
}
```

Inverse Diagonal Sum = 21

Lower Triangular Multiplication


```
const int R = 3, C = 3;

int main()
{
    int A[R][C] = {{1,2,3},{4,5,6},{7,8,9}};
    long m = 1;

   for (int i=0; i < R; i++)
     for(int j=0; j<i; j++)
         m *= A[i][j];

   cout<<"Lower Triangular Multiplication =  "<<m<<endl;
}
```

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

Lower Triangular Multiplication = 224

Lower Triangular Multiplication


```
const int R = 4, C = 4;

int main() {
   int A[R][C] = {{6,2,3,4},
                  {4,2,3,4},
                  {1,2,3,4},
                  {1,2,3,4}};
  long m = 1;
  for (int i=0; i < R; i++)
        for(int j=0; j < C; j++)
        if (i > j)
              m *= A[i][j];
  cout<<"Lower Triangular Mul. = "<<m<<endl;
}
```

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Lower Triangular **Mul.** = 48

Copy one array to another


```
const int R = 4, C = 3;

int main()
{
   int A[R][C] ={{1,1,1},{2,2,2}, {3,3,3}, {4,4,4} };
   int B[R][C];
   for(int i = 0; i < R ;i++)
      for(int j= 0 ; j < C ; j++)
            B[i][j] = A[i][j];
   for(int i=0; i < R; i++){
      for(int j = 0 ; j < C ; j++)
            cout<< B[i][j] <<'\t';
   cout<<endl;
  }
}
```

1	1	1
2	2	2
3	3	3
4	4	4

Store the following symbols in an array


```
const int R = 4, C = 4;

int main() {
   char Symbol[R][C];
   for (int i=0 ; i < R ; i++)
       for (int j=0 ; j < C ; j++)
            if (i == j)
                 Symbol[i][j] = '*';
            else
                 if (i > j)
                      Symbol[i][j] = '#';
                 else
                      Symbol[i][j] = '$';
}
```

*	$	$	$
#	*	$	$
#	#	*	$
#	#	#	*

Matrix Summation


```
const int R = 3, C = 4;  int
main() {
      int A[R][C] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
      int B[R][C] = {{4,5,6,7}, {3,6,7,8}, {9,1,4,2}};
      int C[R][C];
      for(int i = 0 ; i < R ; i++)
            for(int j = 0; j < C ; j++)
                  C[i][j] = A[i][j] + B[i][j];

      for(int i = 0 ; i < R ; i++){
            for(int j = 0 ; j < C ; j++)
                  cout<<C[i][j]<<'\t';
            cout<<endl;
           }
}
```

5	7	9	11
8	12	14	16
18	11	15	14

Compute the average for 2 marks for 3 students


```
int main() {
  int marks[3][2];
  for(int i = 0 ; i < 3 ; i++){
    for(int j = 0 ; j < 2 ; j++){
      cout <<"Enter mark "<<j+1
           <<" for student "<<i+1<<" : ";
      cin>>marks[i][j];
       }
   cout<<endl;
   }
   float avg[3];
   for(int i = 0 ; i < 3; i++){
      float sum = 0;
         for(int j = 0 ; j < 2 ; j++)
           sum = sum + marks[i][j];
      avg[i] = sum/2;
   }
   for(int i = 0 ; i < 3 ; i++)
      cout <<"Average of Student "<<i+1
           <<" = "<<avg[i]<<endl;
}
```

```
Enter mark 1 for student 1 : 5
Enter mark 2 for student 1 : 7

Enter mark 1 for student 2 : 3
Enter mark 2 for student 2 : 4

Enter mark 1 for student 3 : 6
Enter mark 2 for student 3 :
11

Average of Student 1 = 6
Average of Student 2 = 3.5
Average of Student 3 = 8.5
```

Transpose


```
int main(){
   int A[3][4]={{1,2,3,4} ,{5,6,7,8} ,{9,10,11,12}};
   cout<<"Original Array:\n";
   for(int i = 0 ; i < 3 ; i++){
      for(int j = 0 ; j < 4 ; j++)
         cout<<A[i][j]<<"\t";
    cout<<endl;
  }
   int B[4][3];
  for(int i = 0 ; i < 3 ; i++)
      for(int j = 0 ; j < 4 ; j++)
         B[j][i]=A[i][j];
  cout<<"\nTrnspose:\n";
  for(int i = 0 ; i < 4 ; i++){
    for(int j = 0 ; j < 3 ; j++)
         cout<<B[i][j]<<"\t";
   cout<<endl;
   }
}
```

Original Array:

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

Transpose:

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