```
struct tm {
    int     tm_sec;  // 0 -59
    int     tm_min;  // 0 -59
    int     tm_hour;  // 0-23
    int     tm_mday;  // 1-31
    int     tm_mon;   // 0 -11
    int     tm_year;  // years since 1900
    int     tm_wday;  // 0 - Sunday 1 - Monday... to 6
    int     tm_yday;  // julian day 0 - Jan 1
    int     tm_isdst;  // 1 if daylight savings - 0 if not
};
```

```
//time1
#include <iostream>

using namespace std;

#include <ctime>

int main()
{
        time_t time1;
        time_t time2;
        struct tm *tptr;
        clock_t ticks;
        char *currentDate;



        if ( ( time1 = time(( time_t * ) 0 )) != ( time_t )-1 )
         {
                cout << time1 << endl;
                currentDate = ctime( &time1 );
                cout << "currentdate is " <<  currentDate << endl ;

                tptr = localtime( &time1 );


        cout << "Tomorrow is  " << "month " <<
                                (tptr->tm_mon+1) << " day "<<  (++(tptr->tm_mday)) <<
                                " year " << (tptr->tm_year) << endl  <<
                                "Current time is "  << " hour " << tptr->tm_hour <<
                                " minute " << tptr->tm_min <<
```

```
                                " second " << tptr->tm_sec << endl;
        cout << "currentdate is " <<  currentDate << endl ;
        }
        else
         {
                cout <<  "Error with the time() function\n";
         }

                  tptr = gmtime( &time1 );

                cout << "Tomorrow is  " << "month " <<
                                (tptr->tm_mon+1) << " day "<<  (++(tptr->tm_mday)) <<
                                " year " << (tptr->tm_year) << endl  <<
                                "Current time is "  << " hour " << tptr->tm_hour <<
                                " minute " << tptr->tm_min << endl;


                time2 = mktime(tptr);   // opposite of localtime
                cout << time2 << endl;

        if (( ticks = clock() ) != ( clock_t )-1 )
                {
                ticks= double ((ticks)/ double(CLOCKS_PER_SEC) *1000000000.0);  //
CLK_TCK  CLOCKS_PER_SEC
                cout <<  ticks << " nanoseconds used by the processor"  << endl;
                }
        else
                cout << "Error with the clock() function\n" ;

        return 0;
}
    output
1586830561
currentdate is Mon Apr 13 19:16:01 2020

Tomorrow is  month 4 day 14 year 120
Current time is  hour 19 minute 16 second 1
currentdate is Mon Apr 13 19:16:01 2020

Tomorrow is  month 4 day 15 year 120
Current time is  hour 2 minute 16
1586942161
3000000 nanoseconds used by the processor

//math1
/* Include Files */
```

```
#include <cmath>
#include <iostream>
#include <cerrno>
#include <climits>
#include <cstring>
using namespace std;
 // can check is errno is 0

int main()
{
		double n, x, y;
		long double longD;
		int myInt;
		long double PI =
3.14159265358979323846264338327950288419716939937510;
		errno = 0;
		x = -1;
		y = sqrt( x );
		cout << "sqrt error number is " << strerror(errno) << endl;
		cout <<  "sqrt of " << x  << " is " << y << endl;

		errno = 0;
		y = tan (90 * PI / 180.0 );
		cout << "tan error number is " << strerror(errno) << endl;
		cout <<  "tan of 90 degrees " << " is " << y << endl;

		errno = 0;
		n = 300;
		x = 100;

		y = pow( x, n );
		cout << "pow error number is " << strerror(errno) << endl;
		cout <<  x << " raised to the " << n << " power is " << y << endl;

		n = -1;
		x = 0;
		errno = 0;
		y = pow( x, n );
		cout << "pow error number #2 is " << strerror(errno) << endl;
		cout <<  x << " raised to the " << n << " power is " << y << endl;

		errno = 0;
		myInt = INT_MAX;
		myInt++;
		cout << "int error number is " << strerror(errno) << endl;
		cout <<  "myInt " << myInt << endl;
```

```
    errno = 0;
    x=0.0;
    y=1.0;
    n=y/x;
    longD=y/x;
    cout << "division by 0 is " << strerror(errno) << endl;
    cout << " division by 0 " << n << endl;
    cout << " division by 0 long double  " << longD << endl;

    //Overflow/Underflow

    cout << pow(10.0,18.0) + 25.0 - pow(10.0,18.0)  << endl;
   cout << pow(10.0,18.0) - pow(10.0,18.0) + 25.0  << endl;
  longD=pow(10.0,18.0);
  cout << longD + 25.0 - longD  << endl;

    return 0;
}
```

}output

```
sqrt error number is Domain error
sqrt of -1 is nan
tan error number is No error
tan of 90 degrees  is 1.63318e+016
pow error number is Result too large
100 raised to the 300 power is inf
pow error number #2 is Result too large
0 raised to the -1 power is inf
int error number is No error
myInt -2147483648
division by 0 is No error
 division by 0 inf
 division by 0 long double  inf
0
25
25
```

//math 2

```
// Program to depict how to handle divide by zero exception
//The runtime_error class is a derived class of Standard Library class exception, defined
//in exception header file for representing runtime errors.
//This Exception is caught by the catch block which prints the message “Exception
//occurred” and then calls the what function with runtime_error object e. The what()
//function {used in the code given below} is a virtual function of the class Standard
//exception defined in stdexcept header file, it is used to identify the exception. This
```

```
//prints the message “Math error: Attempted to divide by Zero”, after which the program
//resumes the ordinary sequence of instructions*/
#include <iostream>
#include <stdexcept> // To use runtime_error
using namespace std;


// Defining function Division
double Division(double num, double den)
{
    // If denominator is Zero
    // throw runtime_error
    if (den == 0) {
        throw runtime_error("Math error: Attempted to divide by Zero\n");
    }

    // Otherwise return the result of division
    return (num / den);
} // end Division

int main()
{
    double numerator, denominator, result;
    numerator = 12.5;
    denominator = 0;

    // try block calls the Division function
    try {
        result = Division(numerator, denominator);

        // this will not print in this example
        cout << "The quotient is "
             << result << endl;
    }

    // catch block catches exception thrown
    // by the Division function
    catch (runtime_error& e) {

        // prints that exception has occurred
        // calls the what function
        // using runtime_error object
        cout << "Exception occurred" << endl
                << e.what();
          }
```

```
} // end main
```

Output
Exception occurred
Math error: Attempted to divide by Zero

```
//          fpointer1.cpp
//   Objective -  Tabulate Trig functions
// Note deferencing a pointer to a function invokes the function
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
void tabulate (double(*function)(double), double first, double last,
                                        double incr);
double mySqr (double);
int main()
{
  double final, increment, initial;
  cout <<"Enter initial value: "<< endl;
  cin >> initial;

  cout <<"Enter final value: " << endl;
  cin >> final;

  cout <<"Enter increment value: " << endl;
  cin >> increment;

  cout <<"\n\t x\tcos(x)" << endl;
  tabulate(cos,initial,final,increment);
  cout <<"\n\t x\tsin(x)" << endl;
  tabulate(sin,initial,final,increment);
  cout <<"\n\t x\ttan(x)" << endl;
  tabulate(tan,initial,final,increment);
  cout <<"\n\t x\tmySqr(x)" << endl;
  tabulate(mySqr,initial,final,increment);
  }

  void tabulate (double(*function)(double), double first, double last,
```

```
                                        double incr)
 {
 double x;
 int i, numIntervals;

 // ceil is a function that given a double returns the
 // smallest integer that's greater than or equal to x
 numIntervals=ceil((last-first) / incr);
 for (i=0;i<=numIntervals;i++)
   {
   x=first+i*incr ;
   cout <<setw(10) << x << setw(12)<< (*function)(x) << endl;// dereferencing the
pointer

   //(like an array name)
   }
 }

 double mySqr(double x)
 {
   return x*x;
 }
```

output

```
Enter initial value:
0
Enter final value:
0.5
Enter increment value:
0.1
```

```
            x       cos(x)
   0          1
 0.1    0.995004
 0.2    0.980067
 0.3    0.955336
```

```
 0.4    0.921061
 0.5    0.877583

           x        sin(x)
  0          0
 0.1   0.0998334
 0.2    0.198669
 0.3     0.29552
 0.4    0.389418
 0.5    0.479426

           x        tan(x)
  0          0
 0.1    0.100335
 0.2     0.20271
 0.3    0.309336
 0.4    0.422793
 0.5    0.546302

           x       mySqr(x)
  0          0
 0.1       0.01
 0.2       0.04
 0.3       0.09
 0.4       0.16
   0.5       0.25
```

```
// fpointer2.cpp
// illustrates pointers to functions

#include <iostream>
using namespace std;

// function prototypes
    float addOne(      //  adds 1 to a number
            int number) ;    // the number to be incremented
    float addTwo(
     //  adds 2 to a number
            int number) ;    // the number to be incremented twice
```

```
int main()
{
  // pf is a pointer to a function
  // the function must have an integer as an argument
  // and returns a float

  float (*pf)(int); // declares a pointer to a function that
                                          // returns a float and has an int as an
                                          // argument
  pf=addOne;
  cout <<       (*pf)(4)<< endl; //  invoke function
  pf=addTwo;
  cout << (*pf)(4)<< endl;  // invoke function
  cout <<       pf(4)<< endl; // alternate way to call function
  }                   // smiliar to the address of an array

  float addOne(     //  adds 1 to a number
        int number)     // the number to be incremented
        {
              float returnfloat ; // floating point number to be returned
              returnfloat=++number;
              return returnfloat;
        }

  float addTwo(     //  adds 2 to a number
        int number)     // the number to be incremented twice
        {
              float returnfloat ; // floating point number to be returned
              number++;
              returnfloat=++number;
              return returnfloat;
        }
output
5
6
6
```

```
/*          void1.cpp
 *   Objective -  Illustrates the using void pointers */
/* Include Files */
#include <iostream>
using namespace std;


/* Function Declaration */
void funcVoidPtr(      // function that uses a void pointer
    void * voidPointer,   // void pointer that must be cast
    int pointerType); // identifies type of pointer that was passed

 enum parameterType
 {
    integerParameter,
    floatParameter
 };

int main()
{
    enum parameterType myParameterType;  // parameter type
    int myInt=1;
    float myFloat=88.0f;

    // passing an integer pointer
    myParameterType=integerParameter;
    funcVoidPtr(&myInt,myParameterType);
    cout <<  "The value of myInt after the function call is " << myInt << endl;

    // passing an float pointer
    myParameterType=floatParameter;
    funcVoidPtr(&myFloat,myParameterType);
    cout <<  "The value of myFloat after the function call is " << myFloat << endl;

}

/*******************************************/
//   must cast pointer
```

```
    void funcVoidPtr(      // function that uses a void pointer
       void * voidPointer,   // void pointer that must be cast
       int pointerType) // identifies type of pointer that was passed
   {
       if (pointerType==integerParameter)
              {

              (*(int *)voidPointer)++;

              }
       else
              {
              (*(float *)voidPointer)++;
              }
   }
   output
   The value of myInt after the function call is 2
   The value of myFloat after the function call is 89


//              const1.cpp
//   Objective -  Illustrates a const variable with a pointer
#include <iostream>
Using namespace std;

int main( void )
{
       const int firstInt=88;
       int secondInt=77;
       const int * myPtr = &firstInt;  // myPtr is a pointer to a const int and the
                                        // initial value is &firstInt
       //  firstInt++;                  // cannot modify firstInt
 //    cout << ++(*myPtr)<< endl;      //  Compiler error cannot modify a const object

       cout << "the value of firstInt is " << *myPtr++ << endl; // changes the pointer
       myPtr=&secondInt; // the pointer can be changed
       cout << *myPtr << endl;
```

```
//     cout << ++(*myPtr) << endl  //  still cannot modify what is pointed to
       return 0;
   }
   output
   the value of firstInt is 88
   77
   //        const2.cpp
   //   Objective -  Illustrates a const variable with a pointer
   #include <iostream>
   using namespace std;

   int main()
   {
       int myInt=88;
       int secondInt=66;
       int * const myPtr = &myInt; // pointer is constant not what is points to
       *myPtr=77;
       cout << myInt << endl;
       // myPtr=&secondInt;   will produce a compliation error
   }
   output
   77
```

```
//          const3.cpp
//   Objective -  Illustrates a const variable with a pointer
#include <iostream>
using namespace std;

void constFunction (const int * const  myPtr);
void anotherConstFunction  (const int * myPtr);
void yetAnotherConstFunction (int const * myPtr);

int main()
{
          int first=88;
          const int * const myPtr1 = &first;
        //  *myPtr1=77;  will produce a compilation error
        //   myPtr1++; // will produce a compilation error
          constFunction(myPtr1);

          int second = 77;
          const int * myPtr2 = &second;
         // *myPtr2 = 77;  will produce a compilation error
          myPtr2 = &first;
          anotherConstFunction (myPtr2);
```

```
          int third = 66;
          int * const myPtr3 = &third;
         *myPtr3 = 55;
      //     myPtr3 = &first;
         yetAnotherConstFunction(myPtr3);

}

void constFunction (const int * const  myPtr)
{
          int third=77;
          cout << "output from constFunction"<< endl;
       //   *myPtr=55;
        //  myPtr=&third; // cannot change const object
          cout << "third is " << third << endl;

}

void anotherConstFunction(const int * myPtr)
{
          int third=77;
          cout << "output from anotherConstFunction"<< endl;
         // *myPtr=55;  // cannot change const object
          cout << "first is " << *myPtr<< endl;
             myPtr=&third; // can change the pointer
}

void yetAnotherConstFunction (int const * myPtr)
{
          // can change first and Myptr
          int third=77;
          cout << "output from yetAnotherConstFunction"<< endl;
          // *myPtr=55;  // cannot change const object (first)
          cout << "first is " << *myPtr<< endl;
         // myPtr=&third; // cannot change the pointer
           //*myPtr=44; // cannot change const object (thirdInt)
          cout << "third is " << third<< endl;
}
```

output

```
output from constFunction
third is 77
output from anotherConstFunction
first is 88
output from yetAnotherConstFunction
first is 55
third is 77
```

**Bit operations**

```
#include <iostream>
```

```
using namespace std;

int main()
{
        /*
         *
         *   Synopsis  -  Displays the results of bit operations on
         *                variables of type unsigned int.
         *
         *   Objective -  Illustrates operations on bits.
         */


                unsigned int myInt=13;


                 cout << "(~myInt) " << hex << (~myInt) << endl;
                 cout << "myInt << 3 " << hex << (myInt << 3) << " in decimal " << dec <<
(myInt << 3)<< endl;
                 cout << "myInt >> 2 " <<hex << (myInt >> 2) << " in decimal " << dec <<
(myInt >> 2) << endl;
                 cout << "'a' & '5' is " <<('a' & '5') << endl;
                 cout << "'a' | '5' is " <<('a' | '5') << endl;
                 cout << "'a' ^ '5' is " <<('a' ^ '5') << endl;
                 myInt=11 ;
                 cout << (myInt|4) + (myInt^7) << endl;

                // set the third bit
                myInt=2;
                myInt=myInt|4 ;
                cout << "myInt " << myInt << endl;

                // test the third bit
                if (myInt&4)
                        cout << "the third bit is on " << endl;
                else
                        cout << "the third bit is off " << endl;
                return 0;
```

```
}

    Output
(~myInt) ffffffd2
myInt << 3 68 in decimal 104
myInt >> 2 3 in decimal 3
'a' & '5' is 33
'a' | '5' is 117
'a' ^ '5' is 84
27
myInt 6
the third bit is on

#include <iostream>

using namespace std;

int main()
{
        /*
        *  Objective -  Determine if a number is divisible by 4 using
        *  bit operations
        */
                unsigned int inputInteger;

                cout << "Enter an integer (or EOF to quit): " << endl;
                cin >> inputInteger;
                while (cin)
                {
                 if ((inputInteger&1) == 0)  // test the first bit - if the number is odd
                 {
                        if ((inputInteger&2) == 0) // see if divisible by 2
                        cout << inputInteger << " is divisible by 4"
                                        "" << endl;
                        else
                                cout << inputInteger <<  " is not divisible by 4"  << endl;
                        }
                        else
```

```
                    cout << inputInteger <<  " is not divisible by 4"  << endl;
        cin >> inputInteger;
        }
```

Output

```
Enter an integer (or EOF to quit):
5
5 is not divisible by 4
3
3 is not divisible by 4
6
6 is not divisible by 4
8
8 is divisible by 4
12
12 is divisible by 4
```