```
    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;
        }
        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

```
1478746039
currentdate is Wed Nov 09 18:47:19 2016

Tomorrow is  month 11 day 10 year 116
Current time is  hour 18 minute 47 second 19
Tomorrow is  month 11 day 11 year 116
Current time is  hour 2 minute 47
1478861239
0 nanoseconds used by the processor
```

```
/*                          math1.cpp
 *
 *   Synopsis  - Accepts input of values for x and n and displays
 *               the calculated values from the pow(), sqrt() and
 *               tan() functions.
 *
 *   Objective - To illustrate error handling with the
 *                mathematical functions in the ANSI C library.
 */

/* Include Files */
#include <cmath>
#include <iostream>
#include <cerrno>
#include <climits>
using namespace std;
 // can check is errno is 0

int main()
{
                double n, x, y;
                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;


        //Overflow/Underflow

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

        return 0;
}
    }output
sqrt error number is Domain error
sqrt of -1 is -1.#IND
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 1.#INF
pow error number #2 is Result too large
0 raised to the -1 power is 1.#INF
int error number is No error
myInt -2147483648
    0
```

```
//          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 * myPtr);

void anotherConstFunction  (int const * myPtr);

int main()
{
        int myInt=88;
        const int * const myPtr = &myInt;
        //*myPtr=77;  will produce a compliation error
        //myPtr++; // will produce a compliation error
        myInt++; // can change myInt
        constFunction (myPtr);
        anotherConstFunction (myPtr);
```

```
}

void constFunction (const int * myPtr)
{
        // can change myInt and Myptr
        int thirdInt=77;
        cout << "output from constFunction"<< endl;
         //*myPtr=55;  // cannot change const object (myInt)
        cout << "myInt is " << *myPtr<< endl;
        myPtr=&thirdInt;
         //*myPtr=44; // cannot change const object (thirdInt)
        cout << "thirdInt is " << thirdInt<< endl;
}

void anotherConstFunction(int const * myPtr)
{
        int thirdInt=77;
        cout << "output from anotherConstFunction"<< endl;
        //*myPtr=55;  // cannot change const object
        cout << "myInt is " << *myPtr<< endl;
        myPtr=&thirdInt; // cannot change const object
        cout << "thirdInt is " << *myPtr<< endl;
    }
    output
output from constFunction
myInt is 89
thirdInt is 77
output from anotherConstFunction
myInt is 89
thirdInt 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
```