```
// Problem 1a --- Answers will vary, but your function must use a
// do-while loop.

double H( int n )
{
    double sum = 0.0;
    int i = 1;
    do {
        sum += 1.0 / double( i++ );
    } while ( i <= n );
    return( sum );
}


// Problem 1b --- Answers will vary, but your function must use
// recursion.

double H( int n )
{
    double sum = 1.0;
    if ( n > 1 ) {
        sum = H( n - 1 ) + 1.0 / double( n );
    }
    return( sum );
}


// Problem 1c --- Answers will vary, but your function must use
// a for loop.

#include <cmath>

double GH( int n, int p = 1 )
{
    double sum = 0.0;
    for ( int i = 1; i <= n; i++ ) {
        sum += 1.0 / pow( double( i ), p );
    }
    return( sum );
}



// Problem 2a

#include <cmath>

double f( double x )
{
    double result = 0.0;
    if ( x >= -1.0 && x < 0.0 ) {
        result = cos( x * x );
    }
    else if ( x >= 0.0 && x < 1.0 ) {
        result = exp( x * x );
    }
    else if ( x >= 1.0 && x <= 2.0 ) {
        result = sqrt( x ) + log( x ) / x;
    }
    return( result );
}

// Problem 2b --- Monte Carlo integration using the function above with a = -1.0,
// b = 2.0, N = 10000, and max_f = 3.0.  I got 3.8277.
//
```

```
// (The exact value of the integral is about 3.826353907264014.)


// Problem 3 with TOLERANCE = 1.0e-09 and MAX_ITERS = 150...
//
//  Using x1 = -1.0 and x2 = 0.0, result is -0.57577347
//  Using x1 = 0.0 and x2 = 2.0, result is 1.1872837
//  Using x1 = 2.0 and x2 = 5.0, result is 4.3884897



// Problem 4a
#include <iostream>
#include <cstdlib>
#include <iomanip>
using namespace std;

int bitrev( float x[], int n );

int main()
{
    float x[8] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};

    bitrev( x, 8 );

    int i = 0;
    while( i < 8 ) {
        cout << "x[" << i << "] = " << setw(4) << x[i++] << endl;
    }

    cout << "\n\n"; system( "PAUSE" );
    return( EXIT_SUCCESS );
}

//  Bit reversal permutation
//  Usage:  bitrev( x, n )
//
int bitrev( float x[], int n )
{
    float xt;
    int k, j = 0, n2 = n / 2;

    for ( int i = 0; i < n-1; i++ ) {
        if ( i < j ) {
            xt = x[j];
            x[j] = x[i];
            x[i] = xt;
        }
        k = n2;
        while ( k < j+1 ) {
            j -= k;
            k /= 2;
        }
        j += k;
    }
    return( 0 );
}

// Problem 4b
//
// The bitrev() function applies a bit-reversal permutation.  It reorders
// the elements of an array in bit-reversed order.  For example, in an
// array of size 32, the elements are indexed from 0 to 31.  In base-two
// 0 is written 00000 and 31 is written 11111.  Let's focus on the element in
```

```
// location 24.  In base-two, 24 is written 11000.  Reversing the bits of 24
// gives 00011. So the element stored at index 24 is moved to index 3 (i.e. 00011).


// Problem 5
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    int n = 47;
    const int max_iters = 2000;
    int count = 0;

    cout << n << " ";
    do {
        if ( n % 2 == 0 )
            n = n / 2;
        else
            n = 3 * n + 1;
        cout << n << " ";
        if ( ++count > max_iters )
            break;
    } while ( n != 1 );

    cout << "\n\n"; system( "PAUSE" );
    return( EXIT_SUCCESS );
}



// Problem 6
//
// Such a program would be terribly inefficient because harmonic numbers would
// be continually recomputed.  For example, in order to compute H(5), H(1) would
// computed 5 times, H(2) would be computed 4 times, H(3) would be computed 3
// times, H(4) would be computed 2 times, and H(5) would be computed 1 time.
// A more efficient program would involve reusing old harmonic numbers to
// get new ones.
```