

# 15th Annual Computer Science Programming Contest Solutions - C++

```
#include <iostream.h>

// Final Jeopardy Programming Problem
// Input: Our current winnings, followed by the winnings of each
//        of our two competitors (contestants 2 & 3)
// Output: The amount we should wager, according the rules of the
//        problem statement.

// function header:
int prompt_amount(int contestant);

int main()
{
    // get input from user
    cout << "Welcome to the Final Jeopardy round!" << endl;
    int dollars[3]; // current winnings of the 3 contestants
    for( int i = 0; i < 3; i++ )
        dollars[i] = prompt_amount( i+1 );

    // which of the other contestants have more money?
    int larger = dollars[1];
    if( dollars[2] > larger ) larger = dollars[2];

    // decide and output what our wager should be
    if( larger >= dollars[0] )
        cout << "You're not in the lead, and should wager your entire $"
              << dollars[0] << "." << endl;
    else if( dollars[0] > 2*larger )
        cout << "You can comfortably wager $" << dollars[0] - 2*larger - 1
              << "." << endl;
    else
        cout << "You should wager $" << 2*larger - dollars[0] + 1
              << "." << endl;

    return 0;
}

int prompt_amount(int contestant)
{
    int amount;
    while( 1 )
    {
        if( contestant == 1 )
            cout << "Enter your current winnings: ";
        else
            cout << "Enter the current winnings of contestant #"
                  << contestant << ": ";
        cin >> amount;
        if( amount >= 0 ) return amount;
        cout << "Please enter a nonnegative integer..." << endl;
    }
}
```

```

#include <iostream.h>

// Array Rank Programming Problem
// Input: An array size, followed by the specified number of integers
// Output: The corresponding rank array.

// The rank of an element is the number of smaller elements in the
// array plus the number of equal elements that appear to its left.
// The rank array is an array of element ranks. For example, if the
// input array is [4, 3, 9, 3, 7], the corresponding rank array is
// [2, 0, 4, 1, 3].

// function headers:
int prompt_size();

int main()
{
    // get input from user:
    const int array_size = prompt_size();
    int the_ints[array_size];
    int ranks[array_size];

    cout << "Enter the ints, each on a separate line:" << endl;
    for( int i = 0; i < array_size; i++ )
    {
        cin >> the_ints[i];
        ranks[i] = 0;    // simultaneously set all ranks to 0 as we get input
    }                    // (saves having another for loop)

    // compare all pairs of elements, increment the proper ranks
    for( int i = 1; i < array_size; i++ )
        for( int j = 0; j < i; j++ )
            if( the_ints[j] <= the_ints[i] )
                ranks[i]++;
            else
                ranks[j]++;

    // output the rank array
    cout << "The input array was: " << endl;
    for( int i = 0; i < array_size; i++ )
        cout << the_ints[i] << " ";
    cout << endl << "The corresponding rank array is: " << endl;
    for( int i = 0; i < array_size; i++ )
        cout << ranks[i] << " ";
    cout << endl;

    return 0;
}

int prompt_size()
{
    int s = 0;
    cout << "Enter desired size of array : ";
    cin >> s;
    while( s < 1 )
    {
        cout << "Please enter a size of at least 1: ";
        cin >> s;
    }
    return s;
}

```

```

#include <iostream.h>

// Perfect Numbers Programming Problem
// Input: A positive integer
// Output: A message stating whether the number is perfect, abundant,
// or deficient.

// The program continues to prompt for input until a nonpositive integer
// is provided as input.

// function header:
int sumdiv( int n );

int main()
{
    int n, s;
    while(1)
    {
        cout << endl << "Enter an integer (0 to quit): ";
        cin >> n;
        if( n < 1 ) break;

        s = sumdiv(n);
        if( s < n )
            cout << n << " is deficient." << endl;
        else if( s > n )
            cout << n << " is abundant." << endl;
        else
            cout << n << " is perfect." << endl;
    }
    cout << "Goodbye!" << endl;

    return 0;
}

// computes and returns the sum of the proper divisors of n
// (assumes n is positive)
// for example, if n=12, the return value is 1+2+3+4+6=16
int sumdiv( int n )
{
    int sum = 0;
    for( int i = 1; i <= n/2; i++ )
        if( n%i == 0 )
            sum += i;

    return sum;
}

```

```

#include <iostream.h>

// Permutations Programming Problem
// Input: An array size, followed by the specified number of integers
// Output: Each possible permutation of the input array of ints

// Assumption:
// The input array contains no duplicate ints

// Implementation note:
// This solution generates permutations recursively.

// A nonrecursive solution is available at the contest website:
// http://cs.wcu.edu/cscontest

// function headers:
int prompt_size();
void swap( int a[], int i, int j );
void perm( int list[], int k, int m );

int main()
{
    // get input from user:
    const int perm_len = prompt_size();
    int the_ints[perm_len];

    cout << "Enter the ints, each on a separate line:" << endl;
    for(int i = 0 ; i < perm_len; i++)
        cin >> the_ints[i];

    // find & display the permutations
    cout << "The permutations: " << endl;
    perm( the_ints, 0, perm_len-1 );

    return 0;
}

// recursively find and display permutations
void perm( int list[], int k, int m )
{
    if ( k==m )
    {
        // list has one permutation, output it
        for( int i = 0; i <= m; i++ )
            cout << list[i] << " ";
        cout << endl;
    }
    else
    {
        // list[k:m] has more than one permutation
        // generate these recursively
        for( int i = k; i <= m; i++ )
        {
            swap( list, i, k );
            perm( list, k+1, m );
            swap( list, i, k );
        }
    }
}

// prompt user for permutation size; ensure it's at least 1

```

```
int prompt_size()
{
    int p = 0;
    cout << "Enter number of integers in original array : ";
    cin >> p;
    while( p < 1 )
    {
        cout << "Please enter an array size of at least 1: ";
        cin >> p;
    }
    return p;
}

// swap a[i] and a[j]
void swap( int a[], int i, int j )
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

```

#include <iostream.h>

// Permutations Programming Problem
// Input: An array size, followed by the specified number of integers
// Output: Each possible permutation of the input array of ints

// Assumption:
// Unlike the previous version of this problem, this version assumes
// that the input array may contain duplicates.

// Solution overview:
// This solution generates permutations recursively. We find all
// permutations, check to see if the new one we've found is a duplicate,
// and add it to a list of permutations if not. We wait to output all the
// permutations until this process is complete.

// A nonrecursive solution is available at the contest website:
// http://cs.wcu.edu/cscontest

// Disclaimer:
// Using two-dimensional arrays in C++ as this solution does is not
// generally advisable; using a vector or some other pre-defined 2-D
// array class (e.g. apmatrix), or writing our own data structure
// (e.g. a struct or class) to do this would be a better idea...
// The problem is that when passing a 2-D array to a function, we
// actually only get to pass by reference the first element of the
// array; it's our job to know how many columns the array has, so
// we can figure the indexing correctly ourselves...
// Another disadvantage of this implementation is that it declares a 2-D
// array that is potentially much larger than necessary to hold all the
// permutations (if there are lots of duplicates).

// function headers:
int prompt_size();
int numperms( int n );
void swap( int a[], int i, int j );
int check_perm_list( int perm[], int i, int* perm_array, int n );
int perm_add( int list[], int k, int m,
              int* perm_array, int first_empty_row, int cols );

int main()
{
    // get input from user:
    const int perm_len = prompt_size();
    int the_ints[perm_len];
    const int nump = numperms( perm_len );

    cout << "Enter the ints, each on a separate line:" << endl;
    for(int i = 0 ; i < perm_len; i++)
        cin >> the_ints[i];

    // find the permutations
    int array_of_perms[nump][perm_len];
    int total_num_of_perms = perm_add( the_ints, 0, perm_len-1,
                                       &array_of_perms[0][0], 0, perm_len );

    // display the permutations
    cout << "The permutations: " << endl;
    for( int i = 0; i < total_num_of_perms; i++ )
    {
        for( int j = 0; j < perm_len; j++ )
            cout << array_of_perms[i][j] << " ";
    }
}

```

```

    cout << endl;
}

return 0;
}

// recursively add permutations to perm_array (a 2-D array that
// is our collection of permutations thus far)
// finds all permutations of list (only regarding elements k through
// m, inclusive); adds this permutation to perm_array if not already
// there; first_empty_row is where in perm_array to add the new perm.

int perm_add( int list[], int k, int m,
              int* perm_array, int first_empty_row, int cols )
{
    int i;
    if ( k==m )
    {
        // list has one permutation, conditionally add it to the list
        int temp_perm[cols];
        for( i = 0; i <= m; i++ )
            temp_perm[i] = list[i];

        if( check_perm_list( temp_perm, first_empty_row,
                             perm_array, cols ) )
        {
            for( int k = 0; k < cols; k++ )
                perm_array[first_empty_row*cols+k] = temp_perm[k];
            return first_empty_row + 1;
        }
        return first_empty_row;
    }
    else
    {
        // list[k:m] has more than one permutation
        // generate these recursively
        for( i = k; i <= m; i++ )
        {
            swap( list, i, k );
            first_empty_row =
                perm_add( list, k+1, m,
                         perm_array, first_empty_row, cols );
            swap( list, i, k );
        }
        return first_empty_row;
    }
}

// examines the list of permutations thus far (perm_array) to see if
// our current permutation candidate (perm) is really a duplicate

// inputs:
// perm is the permutation to look for in perm_array;
// i is how many permutations in perm_array to check (i.e. how many
// rows through which to iterate)
// perm_size is the number of elements in a permutation (and thus
// also the number of columns in perm_array)

// output:
// returns 0 if perm was found in perm_array, 1 if not

```

```

int check_perm_list( int perm[], int i,
                    int* perm_array, int perm_size )
{
    for( int j = 0; j < i; j++ )
    {
        int duplicate_found = 1;

        for( int k = 0; k < perm_size; k++ )
            if( perm[k] != perm_array[j*perm_size+k] ) // jth row, kth col
                duplicate_found = 0; // not a match, check next perm

        if( duplicate_found )
            return 0; // found a duplicate, return false
    }
    return 1; // went all way through list, didn't find duplicate
}

// basically just computes and returns n!
// assumes n is nonnegative (would just return 1 if not...)
int numperms( int n )
{
    int ans = 1;
    for( int i = n; i > 0; i-- )
        ans *= i;
    return ans;
}

// prompt user for permutation size; ensure it's at least 1
int prompt_size()
{
    int p = 0;
    cout << "Enter number of integers in original array : ";
    cin >> p;
    while( p < 1 )
    {
        cout << "Please enter an array size of at least 1: ";
        cin >> p;
    }
    return p;
}

// swap a[i] and a[j]
void swap( int a[], int i, int j )
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

```