Software Engineering
Lab 1 Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

11 October 2013

1 Hello World on different platforms

Once upon a time there was a C++, where we used to write our codes cross-platform. So basic ”Hello, World!” should be working on all the platforms.

Listing 1: Hello World!

```

#include <iostream>

int main(int argc, char **argv, char **env)
{
  std::cout<<"Hello, World!"<<std::endl;
  return EXIT_SUCCESS;
}
```

2 Variables

2.1 Global and other variables

Local variable is variable declared within scope like main() or any function or anything that using . Their life-cycle is limited with the scope. Whereas global variable is declared globaly which means not in any scope maybe except namespaces, where their life-cycle is limited with the life of the program/executable/thread its declared in.

2.1.1 Elementary functions: Mean, Min, and Max

Since there is no clue on the data type of the variables, instead of overloading the functions for the types int, float, double etc, I wanted to use templates for flexibility.

1. Maximum of two variables

Listing 2: Max Of Two!

```
/*
 * Function: myMaxFunction
 * ————————————————————
 * Calculating the maximum of the two given numbers
 *
 * T1: Any data type of number as first parameter
 * T2: Any data type of number as second parameter
 *
 * returns: nothing.
 * prints: Prints the maximum of the two numbers to console.
 *
 */
template<class T1, class T2>
void myMaxFunction(T1 firstNumber, T2 secondNumber)
{
  cout<<”The maximum of your inputs is: ”<<(firstNumber > secondNumber ? firstNumber : secondNumber)<<endl;
}
```

2. Minimum of two variables

Listing 3: Min Of Two!

```
/*
 * Function: myMinFunction
 * ————————————————————
 * Calculating the minumum of the two given numbers
 *
 * T1: Any data type of number as first parameter
 * T2: Any data type of number as second parameter
 *
 * returns: nothing.
 * prints: Prints the minimum of the two numbers to console.
 *
 */
template<class T1, class T2>
void myMinFunction(T1 firstNumber, T2 secondNumber)
{
    cout<<”The minumum of your inputs is: ”<<(firstNumber < secondNumber ? firstNumber : secondNumber)<<endl;
}
```

3. Mean of two variables

Listing 4: Mean Of Two!

```
/*
 * Function: myMeanFunction
 * ————————————————————
 * Calculating the mean of the two given numbers
 *
 * T1: Any data type of number as first parameter
 * T2: Any data type of number as second parameter
 *
 * returns: nothing.
 * prints: Prints the mean value of the two numbers to console.
 *
 */
template<class T1, class T2>
void myMeanFunction(T1 firstNumber, T2 secondNumber)
{
    cout<<”The mean of your inputs is: ”<<((firstNumber + secondNumber) / 2.)<<endl;
}
```

3 Combination

3.1 Factorial

Here is the iterative factorial function. It returns very large positive integer type because of computational needs. In source code, I'm defined my MY_DEBUG preprocessor to debugging and printing to console.

Listing 5: Factorial

```
/*
 * Function: myFactorialFunction
 * ------------------------------
 * http://en.wikipedia.org/wiki/Factorial
 * "Factorial of a non-negative integer n, denoted by n!"
 *
 * n: positive integer for factorial degree
 *
 * returns: the result of factorial calculation of type "unsigned long long int"
 */
unsigned long long int myFactorialFunction(unsigned int n)
{
#if MY_DEBUG
    cout<<"myFactorialFunction param n: "<<n<<endl;
#endif
    if(n <= 0) return 1;

    // stores the factorial calculation
    unsigned long long int factorial = 1;

    while(n > 1)
    {
        factorial *= n;
#if MY_DEBUG
        cout<<"myFactorialFunction current factorial is "<<factorial<<endl;
#endif
        --n;
    }

    return factorial;
}
```

3.2 Number of combinations from a set

Given set S, here is the function to find k-combination of the set. After having problems with big number calculations, instead of calculation factorials individually by function listed in Listing 5, I try to simplify calculations.

Listing 6: Set Combination

```
/*
 * Function: setCombination
 * ------------------------------
 * http://en.wikipedia.org/wiki/Combination
 * "K-combination of a set S"
 *
 * n: positive integer for set number
 * k: positive integer for k-combination
 *
 * returns: the result of combination of type "unsigned long long int"
 */
unsigned long long int setCombination(unsigned int n, unsigned int k)
{
#if MY_DEBUG
```

```
    cout<<"setCombination param n: "<<n<<" k: "<<k<<endl;
#endif

    // Calculation the denominator part with only k.
    // Instead of calculating all the factorials, below we go n to (n-k) to avoid n! and (n-k)!
    unsigned long long int kFactorial = myFactorialFunction(k);
    unsigned long long int setFactorial = 1; // Storing the result of numerator part, but just n to (n-k)


    // Instead of calculating n! / (n-k)!, we calculate n * (n - 1) * ... * (n - k + 1)
    for(unsigned int i = n; i > (n - k); i--)
        setFactorial *= i;

#if MY_DEBUG
    cout<<"setCombination setFactorial: "<<setFactorial<<" kFactorial: "<<kFactorial<<endl;
#endif

    return setFactorial / kFactorial;
}
```

3.3 Number of combinations with repetitions

Given set S, here is the function to find k-combination of the set with repetitions similar to Listing 6.

Listing 7: Combination w/Repetitions

```
/*
 * Function: repetitionCombination
 * ------------------------------
 * http://en.wikipedia.org/wiki/Combination#Number_of_combinations_with_repetition
 * "K-combination of a set S"
 *
 * n: positive integer for set number
 * k: positive integer for k-combination
 *
 * returns: the result of number of combination with repetitions of type "unsigned long long int"
 */
unsigned long long int repetitionCombination(unsigned int n, unsigned int k)
{
#if MY_DEBUG
    cout<<"repetitionCombination param n: "<<n<<" k: "<<k<<endl;
#endif
    unsigned long long int topResult = myFactorialFunction(n + k -1); // Calculating enumarator part of the formula
    unsigned long long int bottomResult = myFactorialFunction(k) * myFactorialFunction(n -1); // Calculating denominator part of the formula

#if MY_DEBUG
    cout<<"repetitionCombination topResult: "<<topResult<<" bottomResult: "<<bottomResult<<endl;
#endif

    return topResult / bottomResult;
}
```

3.4 Permutations

Given set S, finding k-permutation of the S. After having problems, similar simplification in Listing 6 is used on calculations.

Listing 8: Permutations

```
/*
 * Function: myPermutationFunction
 * ---------------------------
 * http://en.wikipedia.org/wiki/Permutation
 * "The k-permutations, or partial permutations, are the sequences of k distinct elements selected from a set."
 *
 * n: positive integer for set number
 * k: positive integer for k-combination
 *
 * returns: the result of number of combination with repetitions of type "unsigned long long int"
 */
unsigned long long int myPermutationFunction(unsigned int n, unsigned int k)
{
#if MY_DEBUG
    cout<<"myPermutationFunction param n: "<<n<<" k: "<<k<<endl;
#endif

    // return myFactorialFunction(n) / myFactorialFunction(n-k);

    // storing result of the permutation
    unsigned long long int result = 1;

    for(unsigned int i = n; i > n - k; i--) // instead of calculating factorials individually, we multiply from n to n-k( n * (n - 1) * ... (n - k + 1))
        result = result * i;

    return result;
}
```

4 List of Fibonacci numbers and its relation with the golden ratio

4.1 List of Fibonacci

Here is the recursive fibonacci function implemented according to the given formula. On the next example, I had problems with recursive fibonacci, and declared iterative one for being faster.

Listing 9: Fibonacci

```
/*
 * Function: fibonacci
 * ---------------------------
 * http://en.wikipedia.org/wiki/Fibonacci_number
 * "By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two."
 *
 * Calculating the Nth fibonacci number
 *
 * n: positive integer for fibonacci sequence
 *
 * returns: the result of Nth fibonacci number of type "unsigned long long int"
 */
```

```
unsigned long long int fibonacci(unsigned int n)
{
    if(n == 0) return 0; // base case for fib(2-2)
    if(n == 1) return 1; // base case for fib(2-1)
    return fibonacci(n - 1) + fibonacci(n - 2); // recursive call with fibonacci definition
}
```

4.2 Approximation of the golden ratio

Approximating the golden ratio by 0.000001 and with higher precisions. FIB_APPROX_POW is to define power of the 10, where -6, -9 and -12 is tested. Data types like float was not able to keep up with enought precisions, so double is used everywhere.

Listing 10: Fibonacci Approximation

```
unsigned long long int fibonacci2(unsigned int n)
{
    unsigned long long int a = 1, b = 1;
    for (unsigned int i = 3; i <= n; i++) {
        unsigned long long int c = a + b;
        a = b;
        b = c;
    }
    return b;
}

/*
 * Function: fibonacciApproximation
 * ----------------------------
 *
 * Getting ratio of 2 consecutive fibonacci number and substract from golden ratio
 * We try to approximate 10^ -6 to -12 with getting absolute of this difference
 *
 * returns: nothing
 */
void fibonacciApproximation(void)
{
    // register keywords probably will not work,
    // but if it forces program somehow to use CPU registers, it may become faster.

    // stores min difference between ratio and fibonacci
    register long double diff = pow(10.0L, FIB_APPROX_POW);
    // ratio of 2 consecutive fibonacci number
    register long double myFibonacciRatio = .0L;
    // keep track of the index of fibonacci numbers
    register unsigned int i = 1;
    // stores the fibonacci(i + 1) number
    register unsigned long long int fibonacciN1 = 0;
    // stores the fibonacci(i)
    register unsigned long long int fibonacciN = 0;
    // stores absolute value of difference between fibonacci ratio and golden ratio
    register long double approx = .0L;

    do
    {
        fibonacciN = fibonacci2(i); //fibonacci(i)
        fibonacciN1 = fibonacci2(i + 1); //fibonacci(i + 1)

        myFibonacciRatio = double(fibonacciN1) / double(fibonacciN);

        approx = abs(myFibonacciRatio - goldenRatio);

        i++;

    }while(approx > diff);

    cout<<"Latest fibonacciN: "<<fibonacciN<<endl;
    cout<<"Latest fibonacciN1: "<<fibonacciN1<<endl;
    cout<<"Fibonacci index: "<<i<<endl;
    cout<<"Result with 10^("<<FIB_APPROX_POW<<") approximation is: myFibonacciRatio: "<<myFibonacciRatio<<" myDifference: "<<approx<<endl;
}
```

5 Pascal's triangle

Calculating and showing Pascal's Triangle using set combination function in Listing 6.

Listing 11: Pascal Triangle

```
/*
 * Function: printPascalTriangle
 * ------------------------------
 * http://en.wikipedia.org/wiki/Pascal's_triangle
 * "Pascal's triangle is a triangular array of the binomial coefficients"
 *
 * Inspired by: http://www.cplusplus.com/forum/general/56615/#msg304724
 *
 * nPascal: positive integer for Nth first row of pascal
 *
 * returns: nothing
 * prints: pascal triangle of nPascal th first row
 */
void printPascalTriangle(unsigned int nPascal)
{
    unsigned long long int pascalNumber = 1; // storing first number of pascal triangle

    cout<<"==Pascal Triangle=="<<endl;

    for(unsigned int i = 0; i < nPascal; i++) // looping to print pascal lines
    {
        for(unsigned int j = 0; j <= i; j++) // looping to print pascal numbers in line i
        {
            pascalNumber = setCombination(i, j); // Calculation pascal numbers for ith row ex : (5 0) (5 1) (5 2) (5 3) (5 4) (5 5)
            cout<<pascalNumber<<" ";
        }
        cout<<endl;
    }

    return;
}
```