COMP 2011 Midterm Exam - Fall 2016 - HKUST

Date: October 22, 2016 (Saturday)
Time Allowed: 2 hours, 2–4pm
Instructions:
1. This is a closed-book, closed-notes examination.
2. There are **9** questions on **20** pages (including this cover page).
3. Write your answers in the space provided in black/blue ink. *NO pencil please, otherwise you are not allowed to appeal for any grading disagreements.*
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, you are **NOT** allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also **cannot** use any library functions not mentioned in the questions.

Student Name	Solution
Student ID	
Email Address	
Lecture & Lab Section	

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 6
3	/ 10
4	/ 10
5	/ 10
6	/ 12
7	/ 12
8	/ 10
9	/ 20
Total	/100

Problem 1[10 points] True or false

Indicate whether the following statements are ***true*** or ***false*** by circling **T** or **F**. You get 1.0 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

TF (a) A function cannot return a constant. For example, the following function will cause a compilation error.

```
const int confused(int x) {return x; }
```

TF (b) The following `enum` definition will cause a compilation error.

```
enum values { A = 2.3, B, C };
```

TF (c) A value will overflow a floating-point variable if its positive exponent is too large for the exponent field of the floating-point data type of the variable.

TF (d) All of the following special symbols are allowed in a variable name: * (asterisk), | (pipeline), - (hyphen), _ (underscore).

TF (e) If the last executable statement of a function is a `switch` statement, then the `break` statement in each `case` can be replaced by a `return` statement.

TF (f) The following code will output all even integers between 2 to 100 inclusive.

```
counter = 2;
do{
    cout << counter << endl;
    counter += 2;
}while(counter < 100);
```

TF (g) The order of calling functions in the following statement may vary from compiler to compiler.

```
a = f1(23, 14) * f2(12/4) + f3();
```

TF (h) If you assign a value to an array element whose subscript exceeds the size of array, the program will always crash during its execution.

TF (i) The following code will output "-1 is false".

```
if(-1)
    cout <<"-1 is true";
else
    cout <<"-1 is  false";
```

TF (j) The following code is legal, that is, it is syntactically correct and can be compiled without errors.

```
#include<iostream>
 using namespace std;

void fn (int &a)
{
    cout  <<  a  <<  endl;
}

int main()
{
    const int x = 10;
    fn(x);
}
```

Problem 2[6 points] break and continue

What is the output of the following program?

```
#include<iostream>

using namespace std;

int main()
{
    for (int i = 0; i < 4; ++i)
    {
        switch(i)
        {
            case0  : cout <<"0";
            case1  : cout <<"1";continue;
            case2  : cout <<"2";break;
            default: cout <<"D";break;
        }
        cout <<".";
    }
    return0;
}
```

Answer: _____

Solution:

Grading scheme:
case 0 : 01 - 1.5 point
case 1: 1 - 1.5 points
case 2: 2. - 1.5 points
case 3: D. - 1.5 points

Problem 3[10 points] Function Parameter Passing Methods

```
1  #include<iostream>
2  using namespace std;
3
4  float confusion (float x,float & y)
5  {
6       x += 5;
7       float temp = x; x = y; y =  temp;
8       return y;
9  }
10
11 int main()
12 {
13      int a = 10;float b =  1.1;
14
15      /* This comment is to be replaced by the statement in the question  */
16      return0;
17 }
```

If line #15 of the above program is replaced by each of the following statements, give the output if the resulting program can be compiled; otherwise, explain the compilation error.

(a) `cout << confusion(a, b) << endl;`

Answer: _____

(b) `cout << confusion(a, a) << endl;`

Answer: _____

(c) `cout << confusion(b, b) << endl;`

Answer: _____

(d) `cout << confusion(confusion(a,b), b) << endl;`

Answer: _____

(e) `cout << confusion(a, confusion(a,b)) << endl;`

Answer: _____

Solution:

2 points each

0.5 for compilation errors without explanation

Problem 4[10 points] enum and switch

The function `weekdayIfElse()` tells whether a given day is a weekend day, a weekday or not a legal day. Implement function `weekdaySwitchCase()` which gives an alternative implementation of `weekdayIfElse()` by replacing the if-else statements by a single switch statement.

```
enum DAYS {MON = 1, TUE, WED, THUR, FRI, SAT, SUN};

void weekdayIfElse(DAYS day)
{
    if(MON <= day && day <= SUN)
    {
        if(day == SUN || day == SAT)
            cout <<"This is a weekend day"<< endl;
        else
            cout <<"This is a weekday"<< endl;
    }
    else
        cout <<"Not a legal day"<< endl;
}
```

Answer:

Problem 5[10 points] Loops

An integer number is said to be a perfect number if the sum of its factors, including 1 (but not the number itself), is equal to the number. For example, 6 is a perfect number, because 1, 2, and 3 are the factors of 6 and their sum is 6; however, 12 is not a perfect number because the sum of the factors 1, 2, 3, 4 and 6 is greater than 12. Implement a program that finds and prints all the perfect numbers between 1 and 1000 inclusive.

Answer:

Problem 6[12 points] Array

The following program uses integer arrays of size n to store n-digit positive decimal integers. Implement function `arrayAdd()` which adds up two n-digit positive decimal integers, and reports whether an overflow happens.

```
#include<iostream>
usingnamespacestd;

/*
 * Helper function to print the positive decimal integer
 */
voidarrayPrint(constintarr[],intn)
{
      for(inti=0; i<n; i++)
            cout  << arr[i];
}

/*
 * Function definition of arrayAdd is presumed to be    here
 */

 int main()
{
      int op1[4] = {1,5,8,1};// op1 = 1581
       int op2[4]  =  {2,9,3,0};//  op2  = 2930

      /* more examples:
        *  op1[4] = {0,1,3,5} represents the integer    135
        *  op1[4] = {0,0,0,0} represents the integer     0
        */

      int sum[4];
      bool overflow;
      overflow = arrayAdd(op1, op2, sum, 4);
      arrayPrint(op1,   4);
      cout <<" + ";
      arrayPrint(op2, 4);
      cout <<" = ";
      arrayPrint(sum,  4);
      cout << endl;// print out  1581  +  2930  =  4511
      if(overflow)// e.g. overflow  happens  when  9812  +  1306
            cout  <<"The  sum  is  wrong  due  to  overflow"<< endl;

      return0;
}
```

Answer:

```
/*
 * arrayAdd() adds up two n-digit positive decimal integers
 * Parameters:
 *       arrA[] and arrB[] store positive decimal integers to be added up
 *       arrSum[] stores summation
 *       arrA[], arrB[] and arrSum[] are of the same size n
 *       returns true if addition overflow occurs, false otherwise
 */ boolarrayAdd(constintarrA[],constintarrB[],intarrSum[],intn)
{
    // ADD YOUR CODE HERE

}
/* General Guidelines:
 * -0.5 for each syntax error, max -2 points. Repeated errors count only once
 */
```

Problem 7[12 points] Simulate a Virtual 2D Array Using a 1D Array

```
#include<iostream>
using namespace std;

/*
 * Function definitions of big_enough, times2, and print are assumed to be here.
 * But you are only required to implement the 2 functions:
 *       big_enough and times2
 */

int main()
{
    const int N = 100;
    int memory[N];// 1D memory

    // Initialize the array memory[] as {1, 2, 3, ....}
    for(int k = 0; k < N; ++k)
        memory[k] = 1 + k;

    int num_rows, num_cols;// #rows and #columns of the virtual 2D array
    cout <<"Enter the dimensions of the virtual 2D array: ";
    cin >> num_rows >> num_cols;

    if(big_enough(N, num_rows, num_cols))
    {
        // (x1, y1) = upper left corner; (x2, y2) = lower right corner
        cout <<"Enter the indices of the rectangular region: ";
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;

        times2(memory, num_rows, num_cols, x1, y1, x2, y2);
        print(memory, num_rows, num_cols);
    }
    else
        cerr <<"Error: Not enough memory!"<< endl;

    return0;
}
```

The above program tries to simulate a *virtual* 2D integer array of size, $num\_rows \times num\_cols$ using a 1D integer array called *memory*, which has a size of N integers. Let's call the virtual 2D array, a. You are required to implement only the following 2 functions:

(a) `big_enough`: a boolean function that checks if the 1D memory have enough space to hold the virtual 2D array of size $num\_rows \times num\_cols$.

(b) `times2`: multiply a rectangular subset of the virtual 2D array by 2. The rectangular subset is specified by the indices (x1, y1) of its upper left corner, and (x2, y2) of its lower right corner, which correspond to the virtual cells a[x1][y1] and a[x2][y2] respectively. Note that (i) indices start from 0, and (ii) if the specified rectangular region is not entirely enclosed by the 2D array, only the enclosed portion is affected.

You have to determine the function headers of the 2 functions on your own. On the other hand, you don't need to implement the `print` function, whose semantics should be obvious from the following examples.

Here are some sample runs of the above program (assuming the executable is called "a.out" and is run on a linux machine with the prompt "linux: ").

```
linux: a.out
Enter the dimensions of the virtual 2D array: 6 6
Enter the indices of the rectangular region: 1 2 4 5
1  2  3  4  5  6
7  8  18 20 22 24
13 14 30 32 34 36
19 20 42 44 46 48
25 26 54 56 58 60
31 32 33 34 35 36

linux: a.out
Enter the dimensions of the virtual 2D array: 9 20
Error: Not enough memory!

linux: a.out
Enter the dimensions of the virtual 2D array: 4 5
Enter the indices of the rectangular region: 2 2 6 6
1  2  3  4  5
6  7  8  9  10
11 12 26 28 30
16 17 36 38 40

linux: a.out
Enter the dimensions of the virtual 2D array: 4 5
Enter the indices of the rectangular region: -3 -3 2 2
2  4  6  4  5
12 14 16 9  10
22 24 26 14 15
16 17 18 19 20
```

Answer:

```
// 1 point for header (-0.5 for each wrong argument declaration)
 bool big_enough(int array_size,int num_rows,int num_cols)
{


}

// 1 point for header (-0.5 for each wrong argument declaration)
void times2(int memory[],int num_rows,int num_cols,int x1,int y1,int x2,int y2)
{




}
```

Problem 8[10 points] Recursion

Implement a recursive function `void ruler(int n)` which, given a positive integer `n`, will print a ruler-like pattern with the character symbol '-' (hyphen). You may assume without checking that `n` is greater than or equal to 1.

For example, the output for the following function calls are:

ruler(1)	ruler(2)	ruler(3)	ruler(4)
-	- -- -	- -- - --- - -- -	- -- - --- - -- - ---- - -- - --- - -- -

Answer:

```
void ruler(int n)
{// ADD YOUR CODE HERE




}
/* General Guidelines:
 * -0.5 for each syntax error, max -1 points. Repeated errors count only once
 */
```

Problem 9[20 points] 2D Array

The following incomplete program makes a histogram given a list of integer values. The program will determine the range of values with the given integer values, and divide the entire range of values into a series of intervals (bins). Then, it counts how many values fall into each interval. Finally, it draws the histogram to show the frequency of the intervals.

```
#include<iostream>
#include<iomanip>
using namespace std;

const int NUM_DATA = 10;// the number of integer data
const int NUM_BINS = 5;// the number of bins (intervals)
const int MAX_FREQUENCY = NUM_DATA;// the maximum frequency of a bin

/* Function definitions of find_min_max, count_frequency, find_intervals,
 * and make_histogram are presumed to be here
 */

/* prints the histogram created with the corresponding frequencies and interval
 * values of bins
 */
void print_histogram(const char diagram[][NUM_BINS],const int intervals[][2])
{
    cout << setw(4) <<"Freq"<< endl;
    for(int i=0; i<MAX_FREQUENCY; i++)// print the bars
    {
        cout << setw(4) << MAX_FREQUENCY - i;
        for(int j = 0; j<NUM_BINS; j++)
            cout << setw(5) << diagram[i][j] <<"    ";
        cout << endl;
    }
    cout << setw(5) <<"Bin ";
    for(int i = 0; i<NUM_BINS; i++)// print the intervals of bins
        cout <<"["<< setw(2) << intervals[i][0] <<"-"
            << setw(2) << intervals[i][1] <<"]";
    cout << endl;
}

Int main()
{
    char histogram[MAX_FREQUENCY][NUM_BINS] = {};
    int histogram_intervals[NUM_BINS][2] = {};
    cout <<"Dataset 1: "<< endl;
    int dataset1[NUM_DATA] = {12, 24, 31, 11, 9, 1, 2, 1, 2, 7};
    find_intervals(dataset1, histogram_intervals);
    make_histogram(dataset1, histogram_intervals, histogram);
    print_histogram(histogram, histogram_intervals);
```

```
    cout <<"Dataset 2: "<< endl;
    int dataset2[NUM_DATA] = {12, 24, 33, 11, 9, 36, 12, 40, 7, 8};
    find_intervals(dataset2, histogram_intervals);
    make_histogram(dataset2, histogram_intervals, histogram);
    print_histogram(histogram,  histogram_intervals);
    return0;
}
```

The sample output of the above program is:

```
Dataset 1:
Freq
  10
   9
   8
   7
   6
   5    *
   4    *
   3    *       *
   2    *       *
   1    *       *              *       *
 Bin [ 1- 7][ 8-14][15-21][22-28][29-35]
Dataset 2:
Freq
  10
   9
   8
   7
   6    *
   5    *
   4    *
   3    *
   2    *                              *
   1    *              *       *       *
 Bin [ 7-13][14-20][21-27][28-34][35-41]
```

In the first histogram, the range of the values is from 1 to 31. The first interval is from 1 to 7 with 5 values in this interval, while the last interval is from 29 to 35 with 1 value in it. In the second histogram, the range of the values is from 7 to 40. The first interval is from 7 to 13 with 6 values in this interval while the last interval is from 35 to 41 with 2 values in it.

```
/* General guideline:
 * max -2 points for syntax errors. Repeated syntax errors count only once
 */
```

(a) Implement the function `find_min_max()`.

```
/* To find the minimum and maximum integer values in the data array
 * Parameters:
 *      data: an integer array of size NUM_DATA
 *      min_data: the minimum value to be found in the data
 *      max_data: the maximum value to be found in the data
 */
void find_min_max(const int data[], int& min_data, int& max_data)
{
    // ADD YOUR CODE HERE









}
// Total: 4 points
```

(b) Implement the function `count_frequency()` to count the frequency of a bin with the interval [start, end].

```
/* To count the number of values in the data array that fall
 * between start and end inclusive
 * Parameters:
 *      data: an integer array of size NUM_DATA
 *      start: the starting value of the interval
 *      end: the ending value of the interval
 */
int count_frequency(const int data[], int start, int end)
{
    // ADD YOUR CODE HERE









}
// Total: 4 points
```

(c)Implement the function `find_intervals()` to determine the interval of each bin. You may use the function defined in part (a).

```
/* Given the integer values in the array data, determine the
 * interval of each of the NUM_BIN bins. All the intervals should have the
 * same width. In the case where the range, i.e. max value - min value + 1,
 * is not a multiple of NUM_BINS, the next larger multiple value should
 * be used for determining the intervals.
 * Parameters:
 *      data: an integer array of size NUM_DATA
 *      intervals: a 2D array containing the start and end values of every bin,
 *                 e.g. intervals[0][0] stores the start value of the first bin
 *                 and intervals[0][1] stores the end value of the first bin.
 */
void find_intervals(const int data[], int intervals[][2])
{
    // ADD YOUR CODE HERE









}
// Total: 6 points
```

(d)Implement the function `make_histogram()` to make a historgram. You may use the function defined in part (b).

```
/* Given the start and end values of each bin in the 2D array
 * intervals, and the integer values in the array data, count the number
 * of values fall into each bin and draw the histogram on a 2D array
 * diagram.
 * Parameters:
 *      data: an integer array of size NUM_DATA
 *      intervals: a 2D array containing the start and end values of every bin
 *                 e.g. intervals[0][0] stores the start value of the first bin
 *                 and intervals[0][1] stores the end value of the first bin.
 *      diagram: a 2D array containing the bars of asterisks representing
 *               the frequencies of the bins
 */
void make_histogram(const int data[], const int intervals[][2],
                    char diagram[][NUM_BINS])
{
    // ADD YOUR CODE HERE









}
// Total: 6 points
```

---------- END OF PAPER ----------