COMP 2011 Midterm - Spring 2016 – HKUST

- Date: March 21, 2016 (Monday)
- Time Allowed: 2 hours, 7:00 pm - 9:00 pm
- Instructions:
  1. This is a closed-book examination.
  2. There are 8 questions on 18 pages (including the cover page).
  3. Write your answers in the space provided in black/blue ink. NO pencil please.
  4. All programming codes in your answers must be written in ANSI C++.
  5. You may use only the C++ language features and constructs learned in class so far. For example, no pointers, C++ classes, string class, etc.
  6. For programming questions, you are NOT allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also can not use any library functions not mentioned in the questions.

Student Name	Solution
Student ID	
ITSC email	
Lecture and Lab Section	

I have not violated the Academic Honor Code in this examination (signature):_____________

Problem	**Score / Max. score**
1	/12
2	/6
3	/8
4	/16
5	/8
6	/12
7	/12
8	/26
Total	**/100**

Problem 1 True/False Questions *[12 marks]*

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.

T [✓ **F**] (a) The following variable definition is legal. That is, it is syntactically correct and can be compiled without errors.

```
int 1st_number = 123;
```

T [✓ **F**] (b) A C++ program that prints two lines of output must contain two output statements using cout.

[✓ **T**] **F** (c) Not all C++ operators are evaluated from left to right.

T [✓ **F**] (d) The default case is required in the switch statement.

[✓ **T**] **F** (e) The following code would print the values from 1 to 10.

```
int n = 0;
while (n<10)
   cout << ++n << endl;
```

[✓ **T**] **F** (f) Executing the following code results in an infinite loop.

```
for (int i=0; i = 10; i++)
   if (i > 10) break;
```

T [✓ **F**] (g) The following code is syntactically correct and should print whether integer **value** is odd or even:

```
switch (value % 2) {
   case 0:
      cout << “Even integer” << endl;
   case 1:
      cout << “Odd integer” << endl;
}
```

[✓ **T**] **F** (h) Creating overloaded functions with identical parameter lists and different return types is a syntax error.

[✓ **T**] **F** (i) The following code will cause a compilation error.

```
void fn(const int arr[], int n)
{
   for (int i=0; i<n; i++)
      arr[i]++;
}
```

T [✓ **F**] (j) Referring to an element outside the array bounds is a syntax error.

T [✓ **F**] (k) The following code is legal, that is, it is syntactically correct and can be compiled without errors.

```
void f(double a)
{
   float a;
   cout << a << endl;
}
```

[✓ **T**] **F** (l) Executing the following code will result in an infinite loop.

```
int x = 0, total = 0;
while (x <= 100)
   total += x;
   ++x;
```

Problem 2 Break and Continue *[6 marks]*

(a) [4 marks] What is the output of the following program?

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

    int main()
    {
       for (int i = 0; i < 4; i++)
       {
          for (int j = 0; j < i; j++)
          {
             if ((i+j)%2)
                break;
             cout << i+j << '_';  // print one underscore after the value of
    i+j
          }
       }
       return 0;
   }
```

Answer: ___2_ ____

(b) [6 marks] If we replace the **break** statement by a **continue** statement in the program of part (a), what is the new output?

Answer: ______2_4_ ____

Grading Scheme:

2 marks for each number

-1 mark for missing the underscore character

Problem 3 Operators *[8 marks]*

Given the following variable definitions and initializations:

```
int a = 1, b = 2, c = 3;
```

Determine the values for the following expressions:

Expression	**Value**
`a < c`	`True`
`(a < !b) || (!!a)`	`True`
`(a + b) < (!c + c)`	`False`
`a + b`	`3`
`(a += b += 1 + 2)`	`6`
`(a – c) > 0 ? ++a : c++`	`3`

Grading Scheme: 2 marks for each correct answer

Problem 4 Parameter Passing *[16 marks]*

Determine the output of the following programs which call various functions using different parameter passing methods.

(a) [4 marks]

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

void mystery(char x, char y[], int n)
{
   for (int i=0, j=(n-1); i<j; i+=2, j-=2)
   {
      x = y[j];
      y[j] = y[i];
      y[i] = x;
   }
   y[n] = '\0';
}

int main()
{
   char name[] = "m+r orpg+acginm";

   mystery(name[8], name, 11);
   cout << name << endl;

   return 0;
}
```

Answer: ______c++ program______________________________

Grading Scheme: 4 marks for correct answer, no partial mark will be given.

(b) [8 marks]

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

int f(int a, int b)
{
   if (a < b)
      a = b;
   else
      b = a;
   return a+b;
}

int g(int& a, int& b)
{
   if (a < b)
      a = b;
   else
      b = a;
   return a+b;
}


int h(int a, int& b)
{
   if (a < b)
      a = f(a, b);
   else
      b = g(a, b);
   return a+b;
}

int main()
{
   int a = 1, y = 2;
   int& x = a;

   cout << f(x, y) << endl;
   cout << a << ", " << x << ", " << y << endl;
   cout << g(x, y) << endl;
   cout << a << ", " << x << ", " << y << endl;
   cout << h(x, y) << endl;
   cout << a << ", " << x << ", " << y << endl;

   return 0;
}
```

4 // 1 mark

1, 1, 2 // 0.5 marks each, 1.5 marks in total

4 // 1 mark

2, 2, 2 // 0.5 marks each, 1.5 marks in total

6 // 1.5 mark

2, 2, 4 // 0.5 marks each, 1.5 marks in total

Answer: ____

(c) [4 marks]

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

int a = 1;

void decrement(int& a) { a--; }

int main(void)
{
   decrement(a);

   if (a >= 0)
   {
      int a = 2;
      decrement(a);
      cout << a << endl;
   }
   cout << a << endl;
   return 0;
}
```

1

0

Answer: ______________________________________

Grading Scheme:

2 marks for each number

Problem 5 Function Overloading *[8 marks]*

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

int f(bool a, bool b=true)
{
   cout << "f(bool, bool)" << endl;
   return a+b;
}

int f(int a, int b)
{
   cout << "f(int, int)" << endl;
   return a+b;
}

double f(double& a, double& b)
{
   cout << "f(double&, double&)" << endl;
   return a+b;
}

int main()
{
   int x = f(2);
   double y = x;
   double z = f(1, 2);
   z = f(y, 3);
   z = f(z, y);

   return 0;
}
```

What is the output of the above program?

```
f(bool, bool)
f(int, int)
f(int, int)
f(double&, double&)
```

Answer: ____

Grading Scheme:

2 marks for correct call scanning from top to bottom; 8 when they are all correct.

Problem 6 Character Array *[12 marks]*

An English sentence is read into a 1-dimensional C++ char array s. Suppose the sentence consists of the 28 characters in either lower or upper case (‘A’ to ‘Z’ and ‘a’ to ‘z’) and the space character (' ') and hypen ('-'). Implement the function `count_unique_letters()` to find the number of unique characters in a sentence. Your implementation should be a case-insensitive approach which means that a character in the upper case and lower case are regarded as the same character during counting.

- You are NOT allowed to use any string library function except the following:
  `int strlen(const char str[ ]);`
  which may be used to get the length of any string str.

Below is the sample main function:

```
int main()
{
   char text1[100] = "Today and tomorrow";
   cout << "No. of letters = " << count_unique_letters(text1) << endl;

   char text2[100] = "Object-oriented Programming";
   cout << "No. of letters = " << count_unique_letters(text2) << endl;

   return 0;
}
```

And it’s corresponding output:

```
No. of letters = 10
No. of letters = 16
```

You may assume that the text[] array has at most 100 characters including the null character.

```
int count_unique_letters(const char text[])
{
   // Add your code here

   char letters[100] = "";
   int count = 0;

   for (int i=0; i<strlen(text); i++)
                                // looping through each char: 2 marks
   {
      bool found = false;

                                // case conversion: 3 marks
      char lcase = text[i];
      if ((lcase >= 'A') && (lcase <= 'Z'))
         lcase = lcase + 'a' - 'A';

                            // updating the list of unique characters: 4 marks
                            // the number of unique characters: 2 marks
      // search for the character
      for (int j=0; j<count; j++)
      {
         if (lcase == letters[j])
         {
            found = true;
            break;
         }
      }

      // increment the count
      if (found == false)
         letters[count++] = lcase;
   }
   return count;        // return the result: 1 mark

}
```

/* Grading Scheme

-0.5 for each syntax error; max 1 mark

-2 marks for failing to handle case-insensitive matching

-1 for defining an array with non-constant size, e.g. int

Note: both the followings will receive full marks:

1. finding unique alphabets (case insensitive) and space only
2. finding unique alphabets (case insensitive) and any other non-alphabetical characters

*/

Problem 7 Recursion *[12 marks]*

The following is the main function to reverse a number:

```
int main()
{
   int num;
   cin >> num;
   cout << reverse(num, num_digits(num)) << endl;
   return 0;
}
```

For examples, if the user inputs 123, the output will be 321; and if the user inputs 980, the output will be 89. You have to implement the following functions to complete the above program.

Note: you are not allowed to use any library functions.

(a) [5 marks] Implement the recursive function, num_digits(), to count and return the number of digits in a number. For example, the number 1234 has 4 digits and the number 900 has 3 digits.

```
int num_digits(int n)
{
   // Add your code here

   if ((n > 0) && (n < 10))   // base case: 2 marks
      return 1;
   return (1 + num_digits(n / 10));
                           // recursive call & return value: 3 marks

}
```

(b) [7 marks] Implement the recursive function to reverse a number, n, given the number of digits, d.

```
int reverse(int n, int d)
{
   // Add your code here

   if (n == 0)  // base case: 2 marks
      return 0;

   int pow = 1;  // calculate the power of 10: 2 marks
   for (int i = 0; i < d-1; i++)
     pow *= 10;

   return (n % 10) * pow + reverse(n / 10, d - 1);
                  // recursive call & return value:  3 marks
}
```

/* Grading Scheme: -0.5 for each syntax error, max -1 mark for the whole question*/

Problem 8 1D & 2D array *[26 marks]*

/* Grading Scheme: -0.5 for each syntax error, max -2 marks for the whole question*/

(a) [16 marks]

In this question we will implement a score management system. It stores the scores of the quizzes for each student. We can think of the quiz score for a student as a data record and represent it as a row of information. Here is the conceptual view of this collection of data:

Quiz 1	**Quiz 2**	**...**	**Quiz M**
50.5	72.0		91.8

Examples of the arrays are:

```
double scores1[4] = {50.5, 72.0, 90.5, 65.0};
double scores2[5] = {80.3, 90.0, 100, 75.0, 80.0};
```

To provide operations of finding the mean, median and the nth smallest score for analyzing the scores, complete the implementation of the followings.

(Note: the array of scores is not sorted in any order of scores.)

(i) [4 marks] Implement the function, `compute_average()`, to calculate the average scores (mean) for a number of quizzes, Quiz 1 to Quiz **n** in the array **`arr`**. For examples, for the array:

```
double scores1[4] = {50.5, 72.0, 90.5, 65.0};
```

`compute_average(scores1, 3)` will return the value 71.

`compute_average(scores1, 4)` will return the value 69.5.

You can assume that n is smaller than the size of the array.

```
double compute_average(const double arr[], int n)
{
  // Add your code here

  double sum = 0.0;  // initialization: 1 mark
  for (int i = 0; i < n; i++)
     sum += arr[i];
                     // sum of n array elements: 2 marks
  return (sum / n);  // sum/n: 0.5 mark, return result: 0.5 mark

}
```

(ii) [10 marks] Implement the function, `find_nth_smallest()`, to find the ***n-th smallest*** score among the **`m`** quiz scores in the array **`arr`**. For examples, for the array:

```
double scores2[5] = {80.3, 90.0, 100, 75.0, 80.0};
```

`find_nth_smallest(scores2, 5, 1)` will return the smallest (minimum) value 75.0. `find_nth_smallest(scores2, 5, 3)` will return the third smallest value 80.3. `find_nth_smallest(scores2, 5, 5)` will return the fifth smallest (maximum) value 100.0.

You can assume that m is smaller than the size of the array and n is smaller than n.

```
double find_nth_smallest(const double arr[], int m, int n)
{
  // Add your code here

  // Solution 1: by finding the number of elements smaller
  int found_index = -1;

  for (int i = 0; i < m; i++)     // loop through each score: 2 marks
  {
    // for each element, find the rank of it:
    // by comparing it with the other elements
    // and count the number of elements smaller than it
                                // compare with other elements
                                // and finding the rank: 6 marks
    double max = arr[i];
    int order = 1;
    for (int j = 0; j < m; j++)
    {
      if ((i != j) && (max > arr[j]))
          order++;
    }
    // if the element has rank n (has n-1 elements smaller than it)
    // stop the loop to return that element
    if (order == n)
    {
      found_index = i;
      break;
    }
  }
  return (arr[found_index]);    // return the value: 2 marks

  /*
  // Solution 2: by finding the number of elements larger
  int found_index = -1;
  for (int i=0; i<m; i++)
  {
    double min = arr[i];
    int order=0;
    for (int j=0; j<m; j++)
    {
      if ((i!=j) && (min < arr[j]))
      {
          order++;
      }
```

```
            }
            if (order == (m - n))
            {
                found_index = i;
                break;
            }
        }

        */

    }
```

/* Grading Scheme:

It is ok to use sorting algorithms if none of the followings were in your solution:

1. defining a new array using a variable size, e.g. double temp[m] (-1 mark)
2. defining a new array with a literal constant, e.g. double temp[10] (-1 mark)
3. changing the value of the arr elements (arr is a const array) (-1 mark) */

(iii) [4 marks] Implement the function, `find_median()`, to find the median among m quiz scores in the array **`arr`**.

The *median* of a finite set of numbers is defined to be the middle one if the numbers are arranged from lowest to highest (e.g., the median of {5, 3, 3, 11, 9} is 5). If there is an even number of numbers, then the median is defined to be the mean of the two middle values (e.g. the median of {7, 5, 9, 3} is (5 + 7) / 2 = 6).

For examples, for the array:

```
    double scores2[5] = {80.3, 90.0, 100, 75.0, 80.0};
```

`find_median(scores2, 5)` will return the value 80.3.

`find_median(scores2, 4)` will return the value 85.15.

You can assume that m is smaller than the size of the array.

Hint: You may use the function, `find_nth_smallest()`, implemented in (ii).

```
double find_median(const double arr[], int m)
{
    // Add your code here

    int mindex = m/2;
    if (m%2)
        return (find_nth_smallest(arr, m, mindex + 1));
                 // case of m is an odd number: 2 marks
    else
        return ((find_nth_smallest(arr, m, mindex) +
                 find_nth_smallest(arr, m, mindex + 1)) / 2);
                 // case of m is an even number: 2 marks
}
```

/* Grading Scheme:

It is also ok to use a similar algorithm as part (ii). Full marks if the algorithm is correct and able to handle both cases.*/

(b) [8 marks]

To improve the system, we can think of the data records for a number of students as a table of rows of scores. Together with the additional column of students' average scores and the additional row of quizzes' average scores, here is our conceptual view of this collection of data:

Student	**Quiz 1**	**Quiz 2**	**...**	**Quiz M**	***Student Average***
Student 1	50.5	72.3		91.8	*88.3*
Student 2	100.0	0.0		88.5	*70.5*
...					
Student N	90.5	90.5		100.0	*95.5*
Quiz Average	*78.2*	*66.5*		*80.8*	*0*

You may assume this element is always 0.

We will use one 2-dimensional array to store

- all the quiz scores for the students in the first M columns of the first N rows,
- the average scores of each quiz in the N+1 row, and
- the average scores of each student in the M+1 column.

Initially, the 2-dimensional array only contains the scores of all the quizzes for the students. Next, we calculate and store the average scores for each student. We also calculate and store the average scores for each quiz. You are given the following global variables, function prototypes and the definition of the main function:

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

const int QUIZ_NUM = 5;

double compute_average(const double[], int);
double find_nth_smallest(const double[], int, int);
double find_median(const double[], int);
void print_all_scores(const double[][QUIZ_NUM+1], int);
void compute_all_averages(double[][QUIZ_NUM+1], int);

int main()
{
   const int STUDENT_NUM = 6;
   double scores[STUDENT_NUM+1][QUIZ_NUM+1] =
                                          { {100.0, 95.5, 98.0, 88.1, 75.0},
                                             {55.2, 73.4, 87.3, 89.2, 60.0},
                                             {100.0, 99.5, 95.0, 100.0, 0.0},
                                             {25.5, 90.0, 0.0, 0.0, 50.0},
                                             {91.5, 90.0, 100.0, 80.5, 91.0},
                                             {90.5, 100.0, 46.0, 99.0, 100.0} };
   cout << "Initial scores ..." << endl;
   print_all_scores(scores, STUDENT_NUM);
   cout << endl;

   cout << "Calculate average scores ..." << endl;
   compute_all_averages(scores, STUDENT_NUM);
   print_all_scores(scores, STUDENT_NUM);
   cout << endl;

   return 0;
}
```

And the corresponding sample output:

```
Initial scores ...
Student 1      100    95.5   98     88.1   75     0
Student 2      55.2   73.4   87.3   89.2   60     0
Student 3      100    99.5   95     100    0      0
Student 4      25.5   90     0      0      50     0
Student 5      91.5   90     100    80.5   91     0
Student 6      90.5   100    46     99     100    0
Quiz Avg       0      0      0      0      0      0

Calculate average scores ...
Student 1      100    95.5   98     88.1   75     91.32
Student 2      55.2   73.4   87.3   89.2   60     73.02
Student 3      100    99.5   95     100    0      78.9
Student 4      25.5   90     0      0      50     33.1
Student 5      91.5   90     100    80.5   91     90.6
Student 6      90.5   100    46     99     100    87.1
Quiz Avg       77.1167 91.4   71.05  76.1333 62.6667 0
```

Implement the function, `compute_all_averages()`, to calculate

- the average scores for each students of a number of quizzes and store the results into the last column of the 2-dimensional array, `scores[][]`. For example, the average score of Student 1 is stored in the last element of the first row, and the average score of Student 3 is stored in the last element of the third row.
- the average scores for each quiz of a number of students and store the results into the last row of the 2-dimensional array, `scores[][]`. For example, the average score of Quiz 1 is stored in the first element of the last row, and the average score of Quiz 3 is stored in the third element of the last row.

The number of students is given in `student_num` and the number of quizzes in QUIZ_NUM. You may call the function, `compute_average()`, implemented in part (a)(i).

```
void compute_all_averages(double scores[][QUIZ_NUM+1],
                                 int student_num)
{
    // Add your code here
```

```
                    // compute the average scores for each student: 3 marks
     for (int i=0; i<student_num; i++)
        scores[i][QUIZ_NUM] = compute_average(scores[i], QUIZ_NUM);

                    // compute the average scores for each quiz: 5 marks
     for (int i=0; i<QUIZ_NUM; i++)
     {
        double sum = 0.0;
        for (int j=0; j<student_num; j++)
        {
           sum+=scores[j][i];
        }
        scores[student_num][i] = sum/student_num;
     }
}
```

/* Rough work — DON'T DETACH THIS PAGE */

/* Rough work — DON’T DETACH THIS PAGE */