COMP 2011 Midterm - Spring 2014 - HKUST

Date: March 29, 2014 (Saturday)
Time Allowed: 2 hours, 10:30am – 12:30pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **9** questions on **18** pages (including this 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 the 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 **cannot** use any library functions not mentioned in the questions.

Student Name	Model Answers
Student ID	
Email Address	

For T.A. Use Only

Problem	Score
1	/ 15
2	/ 8
3	/ 8
4	/ 7
5	/ 10
6	/ 10
7	/ 10
8	/ 20
9	/ 12
Total	/ 100

Problem 1 [15 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 1.5 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

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

```
const char _888_[ ] = "_888_";
```

T [✓ **F**] (b) It is legal in C++ to define a function inside another function. For example, the following code is legal in C++:

```
void f(int x, int y)
{
    int max(int a, int b) { return (a > b) ? a : b; }
    return 1 + max(x, y);
}
```

[✓ **T**] **F** (c) One may define 2 functions of the following prototypes in the same C++ program in the same file.

```
int add(int, int);
int add(int&, int);
```

T [✓ **F**] (d) Given that the size of char is 1 byte, the following statement still will compile but will generate a runtime error during the initialization of the char variable **x**.

```
#include <iostream>
using namespace std;
int main(void) { char x = 1000; cout << x << endl; return 0; }
```

T [✓ **F**] (e) Since a C++ program can have only one **main** function, the **main** function cannot call itself recursively.

[✓ **T**] **F** (f) Given the following definition and initialization of 3 int variables

```
int a = 3, b = 4, c = 5;
```

what is the truth value of the following boolean expression?

```
((a+b) > c) && b
```

T **F** (g) The following use of *const* in the declaration of the formal parameter **x** will cause compilation error.

```
int test(const int x) { return ++x; }
```

T **F** (h) In C++, global variables are initialized to zero.

T **F** (i) The following 2D array definition compiles and runs without any errors.

```
int a[ ][2] = { { 1 } , { 2 , 3 } };
```

T **F** (j) What is the truth value of the following boolean expression?

```
(5/4 == 6/5)
```

Problem 2 [8 points] Integer Division and Remainder Functions

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

int main(void)
{
    int j = 1;

    while (j <= 15)
    {
        ++j;
        if (j/5 != 1)                    // '/' operator is changed to '%' operator in part (b)
            continue;
        else
            cout << j << ' ';
    }

    cout << endl;
    return 0;
}
```

(a) [4 points] What is the output of the above program?

Answer: **5 6 7 8 9**

(b) [4 points] If we replace the *division* operator '/' by the *mod* operator '%', what is the new output?

Answer: **6 11 16**

Grading scheme:
(a) 1 point for each of the first 3 numbers, scanning from left to right, then the remaining 1 point for completely correct answer.
(b) 1 point for each number in the answer, scanning from left to right; 4 when they are all correct.

Problem 3 [8 points] Break and Continue

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

int main(void)
{
    int A[3][3] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
        {
            if(i == j)
                continue;                    // "continue" is changed to "break" in part (b)
            A[i][j] = 4;
        }

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << A[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

(a) [4 points] What is the output of the above program?

```
0 4 4
4 0 4
4 4 0
```

Answer: ______

(b) [4 points] If we replace the *continue* statement by *break*, what is the new output?

```
0 0 0
4 0 0
4 4 0
```

Answer: ______

Grading scheme:
(a) 1 point for each row, and 1 point for the format.
(b) 1 point for each row, and 1 point for the format.
But all the numbers on a row must be all correct.

Problem 4 [7 points] Scope

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

void foo(int& x) { x--; }

int main(void)
{
    int a = 1;
    int& b = a;
    foo(b);

    if ((a++) == 0)
    {
        int a = 1;
        foo(a);
        cout << a << ' ' << b << endl;
    }

    cout << a << ' ' << b << endl;
    return 0;
}
```

What is the output of the above program?

```
0 1
1 1
```

Answer: ____

Grading scheme:
4 points for the first correct row, and 3 points for the other row.
But all the numbers on a row must be all correct.

Problem 5 [10 points] Overloaded Functions and Functions with Default Arguments

```
/* External function prototypes */
int F(bool b, int i = 0);
int F(int i, int j);
int F(int i, bool b = true);
int F(int i, float f);
int F(float f);
float F(float& f);

int main(void)
{
    int x = 42;
    float y = 3.14;
    float& yref = y;

    /* A FUNCTION CALL STATEMENT IS TO BE ADDED HERE */
    return 0;
}
```

A function call statement is taken, one at a time, from the following table and inserted to the above program to check if it is a valid statement. A statement is valid if after it is added, the program can be separately compiled with no errors. You get 1 point for each **correct** answer, -0.5 points for each **incorrect** answer, and 0 point if the question is **unanswered**.

Statement to Add	Valid?	Invalid?
(a) `x = F(0, 0);`	✓	
(b) `x = F(x, 3);`	✓	
(c) `x = F(y, 3);`		✓
(d) `x = F(x, y);`	✓	
(e) `x = F(y, x);`		✓
(f) `x = F(true, 3);`	✓	
(g) `x = F(yref);`		✓
(h) `y = F(yref);`		✓
(i) `x = F('a');`	✓	
(j) `x = F();`		✓

Problem 6 [10 points] Print Recursive Calls

```
int num_zeros(int n)                                  /* File: num_zeros.cpp */
{
    if (n == 0)                                       // Base case #1
        return 1;
    else if (n < 10 && n > 0)                         // Base case #2
        return 0;
    else
        return num_zeros(n/10) + num_zeros(n%10);
}
```

The recursive function, `int num_zeros(int)`, above is an example in our lecture notes on *recursion*. It finds the number of zeros in a non-negative integer.

Write another recursive function,

```
        void print_num_zeros_calls(int n);
```

to print out the recursive calls of `num_zeros` on the input n. For example, if $n = 120803$, the output of `print_num_zeros_calls(120803)` is as follows:

```
call num_zeros(12080)
call num_zeros(1208)
call num_zeros(120)
call num_zeros(12)
call num_zeros(1)
call num_zeros(2)
call num_zeros(0)
call num_zeros(8)
call num_zeros(0)
call num_zeros(3)
```

Your answer must be a **recursive** function that works with the `main` function on the next page to produce the above results.

Note that the function `print_num_zeros_calls` does not compute the number of zeros in the given number but only tells you how the function `num_zeros` is called to produce the answer of `num_zeros`.

Answer:

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

void print_num_zeros_calls(int x)
{
    // ANSWER: ADD YOUR CODE HERE

    if (x > 9)        // 2 points for knowing that there are printouts only when x > 9
    {
        cout << "call num_zeros(" << x/10 << ")" << endl;                    // 2 points
        print_num_zeros_calls(x/10);                                          // 2 points

        cout << "call num_zeros(" << x%10 << ")" << endl;                    // 2 points
        print_num_zeros_calls(x%10);                 // 2 points; this is redundant though
    }
}

int main(void)
{
    cout << "Enter a positive integer:  ";
    int x;
    cin >> x;

    print_num_zeros_calls(x);

    return 0;
}
```

Grading scheme:
- for each syntax error, - 0.5 point; max. 1 point
- the order of the codes is important
- the last line of code is redundant but the concept is there. You will get the 2 points if your answer doesn't violate the concept.

Problem 7 [10 points] Compare Character Strings

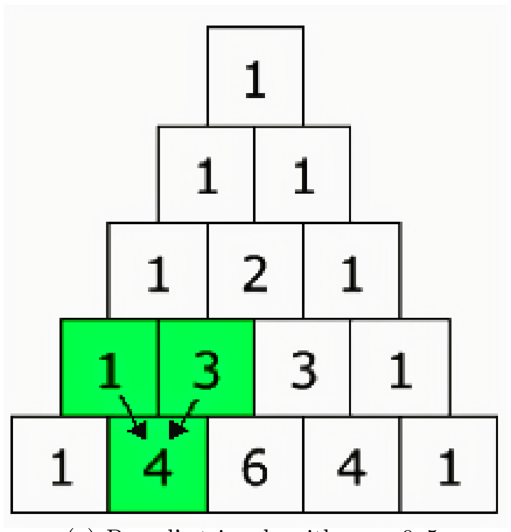
In the C++ standard library, there is a function `strcmp` that will take 2 C++ character strings and compare them lexicographically. Here you are going to implement a similar function called `str_compare` that is defined as follows:

- It takes 2 C++ character strings, s1 and s2.
- The characters they contain have an ASCII code that is smaller than 128.
- s1 is equal to s2 only if they have the same length and they contain exactly the same characters.
- To compare the 2 strings, scan them from left to right until the end of one string. Check their first differing characters. If that character of s1 is smaller than that of s2, then s1 is said to be smaller than s2.
- Character **x** is said to be smaller than character **y** if their difference (**x** - **y**) is negative; they are equal if the difference is zero; **x** is greater than **y** if the difference is positive.
- `str_compare` should return -1 if s1 is smaller than s2, 0 if they are equal, and +1 if s1 is greater than s2.

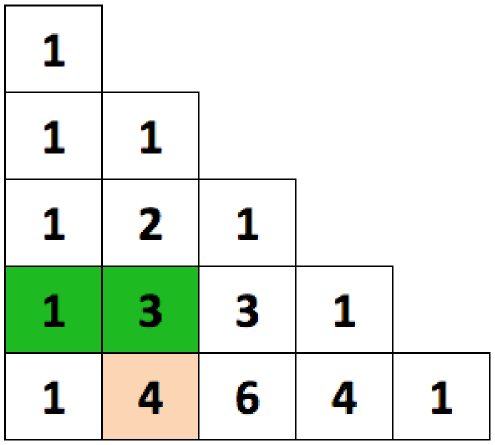
Answer:

```
int str_compare(const char s1[ ], const char s2[ ])
{                                          /* ANSWER: ADD YOUR CODE HERE */
    /* You are not allowed to use the lib function 'strcmp' */
    // 4 points for skipping all the common characters
    int j = 0;
    while ((s1[j] == s2[j]) && s2[j] != '\0')
        ++j;
    // 6 points: 2 points for each case
    int diff = s1[j] - s2[j];
    if (diff < 0)
        return -1;
    else if (diff > 0)
        return 1;
    else
        return 0;
}
/** Grading scheme:
 * - for each syntax error, - 0.5 point; max. 1 point
 * - can't stop at the null char: -1 point
 * - get the 3 cases but return the wrong value: -0.5 point for each case
 * - get the idea of a case but doesn't cover all possibilites: 1 point only
*/
```

Problem 8 [20 points] Recursive Generation of Pascal's Triangle



(a) Pascal's triangle with rows 0–5.



(b) C++ representation.

Pascal's triangle is a triangular array of the binomial coefficients. (You don't need to know what binomial coefficients are to answer this question.) In programming, one may generate the Pascal's triangle row by row, starting from the top row $\mathbf{r} = 0$. Here are the rules,

- For row $\mathbf{r}$=n, there will be $n + 1$ integers.
- Except for row 0 which contains only one integer 1, all other rows start and end with the integer 1.
- Starting at row 2, except for the first and the last number in the row, each of the middle numbers is the sum of 2 numbers in the row immediate above as shown in the figures. This is the recurrence relation to be used for the recursive generation of the numbers.
- You may assume that the maximum number of rows is 1000 if you think you need this.

Implement 2 versions of the function `pascal_triangle` to generate the $n+1$ numbers in the nth row of the Pascal's triangle, using the given function prototype: an iterative version and a recursive version. You MUST use the recurrence rule given above for their generation, and are NOT allowed to directly use the formula of binomial coefficients.

You may assume that $n + 1$ is always smaller than the size of the array that stores the coefficients.

(a) [10 points] Implement an **iterative** solution that uses loops to generate a row of the Pascal's triangle.

Answer:

```
void pascal_triangle(int row_coef[ ], int n)
{
    /* ANSWER: ADD YOUR CODE HERE */
    row_coef[0] = 1;

    if (n == 0)
        return;                              // Base case #1 – correct row 0: 1 point

    row_coef[1] = 1;                         // Base case #2 – correct row 1: 1 point

    for (int k = 2; k <= n; ++k)                         // loop condition: 1 point
    {
        for (int j = k-1; j > 0; --j)                    // loop condition: 1 point
            row_coef[j] += row_coef[j-1];             // recurrence formula: 5 points
        row_coef[k] = 1;                                 // last coefficient: 1 point
    }
}

/*
 * Grading scheme:
 * - for each syntax error, - 0.5 point; max. 1 point
 * - base case for row 0: 1 point
 * - base case for row 1: 1 point
 * - recursion part: 7 points
 * - set the last coefficient to 1: 1 point
*/
```

(b) [10 points] Implement a **recursive** solution to generate a row of the Pascal's triangle.

Answer:

```
// row_coef[ ]: to store coefficients in one row of the Pascal's triangle
// n: index of a row of the Pascal's triangle; it starts from 0
void pascal_triangle(int row_coef[ ], int n)
{
    /* ANSWER: ADD YOUR CODE HERE */
    /* Another way to catch the base case more explicitly

    row_coef[0] = 1;
    if (n == 0) // Base case: 2 points
        return;

    */
    if (n < 0)                                              // Base case: 2 points
        return;

    pascal_triangle(row_coef, n-1);                         // 2 points

    for (int j = n-1; j > 0; --j)                           // 5 points
        row_coef[j] += row_coef[j-1];

    row_coef[n] = 1;                                        // 1 point
}

/*
 * Grading scheme:
 * - for each syntax error, - 0.5 point; max. 1 point
 * - base case for row 0: 2 point
 * - recursion part: 7 points
 * - set the last coefficient to 1: 1 point
*/
```

Problem 9 [12 points] Length of the Longest Word in a Sentence

An English sentence is read into a 1-dimensional C++ char array `s`. To make this problem simple, a sentence consists of only words separated by one or more spaces, and each word is comprised of the 26 English letters in either lower or upper case. There are no other characters, not even punctuation. Implement the function `max_word_length` to output the longest word in the sentence. If there are multiple words with the same longest length, output the first one.

EXAMPLE	**SENTENCE**	**OUTPUT**
Example #1	" he is a good student"	"student"
Example #2	"Hong Kong University of Science and Technology"	"University"
Example #3	" Hong Kong is city "	"Hong"

You function actually computes 2 things: the array index of the first character of the longest word (in variable **start_index**, and the length of the longest word (by the return value).

Grading scheme #1: if you follow our solution

(a) for each syntax error, -0.5 point; max. 1 point

(b) forget to initialize start_index or max_length: -0.5 point each

(c) no or wrong return statement: -1 point

(d) no penalty if the answer consider leading spaces nor trailing spaces

Grading scheme #2: if you don't use nested loops, additional rules besides (a)-(d)

(e) correct breaking condition: 1 point

(f) store the length of current word: 2 points

(g) know when to start a new word: 2 points

Answer:

```
/* Input variables:
 *    const char s[]: Given sentence consists of only letters and space ONLY.
 *    start_index: At the end, it saves the index of the first character of
 *                      the longest word in s.
 *
 * Return value: the length of the longest word in s.
 */
int max_word_length(const char s[ ], int& start_index)
{
    // ANSWER: ADD YOUR CODE HERE
    int i = 0;                                   // Index of the first char of the next word
    int max_length = 0;
    start_index = 0;

    while (s[i] != '\0')                                    // Stopping condition: 1 point
    {
        int j = i;                                       // Correct initialization: 1 point

        // Scan thru a word: 4 points
        while (s[j] != ' ' && s[j] != '\0' )
            j++;

        // Update info of the longest word: 4 points
        if (j-i > max_length)
        {
            max_length = j-i;
            start_index = i;
        }
        // End of sentence?: 1 points
        if (s[j] == '\0')
            break;

        // Update the starting index of the next word: 1 point
        i = j+1;
    }

    return max_length;
}
```