COMP 2011 Midterm - Fall 2015 - HKUST

Date: October 17, 2015 (Saturday)

Time Allowed: 2 hours, 2–4pm

Instructions:

1. This is a closed-book, closed-notes examination.
2. There are 8 questions on **16** 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 disagreement.*
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	
Student ID	
Email Address	
Lecture & Lab Section	

For T.A.

Use Only

Problem	**Score**
1	/ 10
2	/ 9
3	/ 9
4	/ 8
5	/ 10
6	/ 12
7	/ 21
8	/ 21
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.

T **F** (a) The character `'c'` and character string `"c"` require the same amount of storage.

T **F** (b) The following definition and initialization of the constant integer identifier `NUM` is legal. That is, it is syntactically correct and compiles without error(s).

```
const int NUM;
cin >> NUM;
```

T **F** (c) After executing the following piece of code, the value of the variable `x` will be 1 and that of the variable `y` will be 0.

```
int x = 0, y = 0;
if (!true)
    x++;
else
    y++;
```

T **F** (d) Executing the following code results in an infinite loop.

```
for (int i = 0; true; i++) { continue; }
```

T **F** (e) The following expression is always false, regardless of the value of integer `i`.

```
(i < 10 ? i / 10 : -i / 10)
```

T **F** (f) The following piece of code will cause an infinite loop.

```
int z = 10;

while (z > 0)
    if (z % 2)
        z--;
```

T **F** (g) A function with void as its return type cannot have any return statements in its function body.

T **F** (h) Compiling the following code results in a compilation error.

```
int a[3];
a[-1] = 0;
```

T **F** (i) Assume that there is a logical operator called "exclusive-or" `^` which works as follows and is left-to-right associative:

x	y	x^y
true	true	false
true	false	true
false	true	true
false	false	false

The following boolean expression will be evaluated to `true`.

```
(x < 1) ^ (x == 1) ^ (x > 1)
```

T **F** (j) Suppose `foo(int& x, unsigned& y)` is a function with its two parameters passed by reference. The function call `foo(a, 10)` is invalid, where `a` is an `int` variable already defined.

Problem 2 [9 points] Conditional

Consider the following program:

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

enum Color { RED, GREEN, BLUE };

int main( )
{
    int num;
    Color color;

    cout << "Enter a non-negative integer:  ";
    cin >> num;
    color = static_cast<Color>(num % 3);

    switch (color)
    {
        case RED:
            if (num % 2) { cout << RED; break; }

        case GREEN:
            while (true)
            {
                if (num <= GREEN)
                    break;
                cout << num;
                num -= 3;
            }

        case BLUE:
            cout << (num ? RED+GREEN : GREEN+BLUE);
            break;
    }

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

Give the program output for each number entered by the user:

(a) Input: 0

Answer: ____

(b) Input: 1

Answer: ____

(c) Input: 3

Answer: ____

(d) Input: 4

Answer: ____

(e) Input: 6

Answer: ____

(f) Input: 10

Answer: ____

Problem 3 [9 points] Loops

What is the expected output of the following program?

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

int main( )
{
    int x = 3, y = 60, z = 10;

    while ( (z > 0) && (y / 10) )
    {
        if ( !(z % 2) )
            y /= x;
        else
            x += 2;

        z--;
        cout ≪ "x = " ≪ x ≪ ", y = " ≪ y ≪ ", z = " ≪ z ≪ endl;
    }

    return 0;
}
```

Answer: ____

Problem 4 [10 points] Function Parameter Passing Method

Consider the following program:

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

void f(int& a, double b)
{
    cout << a-- << '\t' << b << endl;
}

int g(double x, int& y) { return (x / ++y); }

int main( )
{
    int m = 20, n = 10;
    f(m, g(--m, n));

    cout << m << '\t' << n << endl;
    return 0;
}
```

(a) What is the output when the program is run?

Answer: ____

(b) One of the formal parameters of `f()` and `g()` has the type `int&`. Suppose it is changed to `int`. What is the output when the revised program is run?

Answer: ____

Problem 5 [10 points] Array and C String

An arithmetic expression is input as a C string with the following 3 requirements:

- all operands are single-digit integers;
- all operators are either '+' or '-' (that is, only addition and subtraction are supported);
- there are no whitespaces between any operands and operators.

For example, the expression "5-9+8-1" is evaluated to 3. Complete the following program to evaluate such expressions. You may assume that the input expression is always *non-empty* and *valid*.

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

int main( )
{
    char expression[101];
    cout << "Input your integer expression:  ";
    cin >> expression;
    int length = strlen(expression);

    // ADD YOUR CODE HERE: To evaluate the input expression















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

Problem 6 [12 points] Recursion

Consider the following program:

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

void callme_maybe(int x)
{
    if (x > 3)
        return;

    callme_maybe(++x);
    cout ≪ x;
    callme_maybe(x++);
}

int main( )
{
    int x;
    cout ≪ "Here’s my number:  ";
    cin ≫ x;

    callme_maybe(x);
    cout ≪ endl;
    return 0;
}
```

Give the program output for each number entered by the user:

(a) Input: 3

Answer: ____________________

(b) Input: 2

Answer: ____________________

(c) Input: 1

Answer: ____________________

Problem 7 [21 points] Prime Numbers

(a) Write the function `int count_primes(int n)` which, given a positive integer `n`, counts the number of primes that are no greater than `n`. For example, `count_primes(7)` should return 4, since there are 4 primes that are no greater than 7: 2, 3, 5, 7. To solve this problem, you must use a function `bool is_prime(int x)` which returns `true` if `x` is prime and `false` otherwise. (You do not need to write this `is_prime( )` function).

```
int count_primes(int n)
{      // ADD YOUR CODE HERE









}
```

(b) Twin primes are consecutive odd numbers $(p,\ p+2)$ that are both prime. Examples of twin prime pairs are (5, 7) and (29, 31). Write a function `int count_twin_prime_pairs(int n)` which, given a positive integer `n`, counts the number of such twin prime pairs that are no greater than `n`. For example, `count_twin_prime_pairs(30)` should return 4, since there are 4 twin prime pairs no greater than 30: (3, 5), (5, 7), (11, 13), (17, 19). Note that (29, 31) does not qualify since 31 is greater than 30.

You must also use the `is_prime()` function for this part.

```
int count_twin_prime_pairs(int n)
{      // ADD YOUR CODE HERE









}
```

(c) Write a function `int get_separated_primes(int k)` which, given a gap k, returns the smallest prime p, such that both p and $p + k$ are consecutive primes (i.e., $(p,\ p + k)$ are the smallest consecutive primes that have a gap of k). Note that p and $p + k$ must be consecutive primes, thus there should be no other primes between p and $p+k$. For example, `get_separated_primes(4)` should return 7, since (7, 11) is the smallest consecutive primes pair with a gap of 4.

You must also use the `is_prime()` function for this part.

```
int get_separated_primes(int k)
{
    // ADD YOUR CODE HERE

}
```

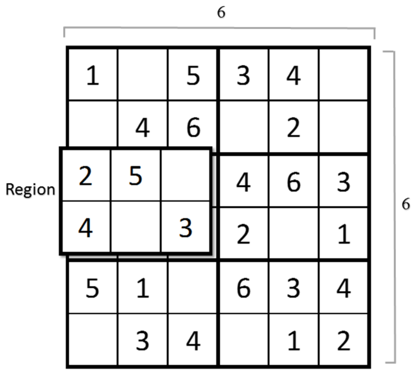
(d) An isolated prime is a prime number p such that neither $p-2$ nor $p+2$ is prime. In other words, p is not part of any twin prime pair. For example, 2 is an isolated prime because both 0 and 4 are not prime; on the other hand, 11 is not because 9 is not prime though 13 is. Write a function int `count_isolated_primes(int n)` which, given a positive integer n, counts the number of primes no greater than n.

Important note: You cannot have loops in this function, but you may use any functions that you wrote for parts (a) and (b) above. Hint: The prime number 5 is the only prime that belongs to two different twin prime pairs: (3, 5) and (5, 7). Moreover, be careful with the numbers around `n` (the boundary).

```
int count_isolated_primes(int n)
{
    // ADD YOUR CODE HERE
























}
```

Problem 8 [21 points] Mini Sudoku

Mini-Sudoku is a number-placement puzzle. It is played over a 6x6 grid that is further divided into six 2x3 sub-grids called “regions”. Below is an example of a mini-Sudoku puzzle:



1		5	3	4	
	4	6		2	
2	5		4	6	3
4		3	2		1
5	1		6	3	4
	3	4		1	2

When the game starts, some of the grid cells are already filled with numbers.

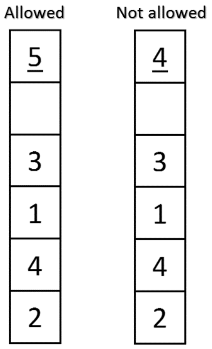
1		5	3	4	
	4	6		2	
2	5		4	6	3
4		3	2		1
5	1		6	3	4
	3	4		1	2

The objective is to fill the other empty cells with integers between 1 to 6 (1 number only in each cell) according to the following rules:

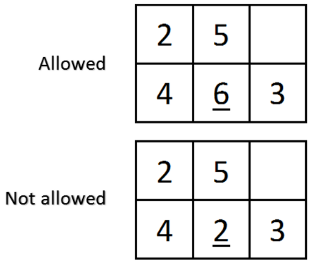
- A number can appear only once on each row:

Allowed	1	2	5	3	4	
Not allowed	1	3	5	3	4	

- A number can appear only once on each column:



- A number can appear only once on each region:



You are asked to write a program to play this mini-Sudoku game. The above mini-Sudoku example will be represented by a 2-dimensional integer array where empty cells are represented by 0. Below is an example of such 2D `int` array:

```
int puzzle[6][6] = {{1, 0, 5, 3, 4, 0}, {0, 4, 6, 0, 2, 0}, {2, 5, 0, 4, 6, 3},
                    {4, 0, 3, 2, 0, 1}, {5, 1, 0, 6, 3, 4}, {0, 3, 4, 0, 1, 2}};
```

Complete the following incomplete program by writing the 3 functions required in parts (a)–(c), of which prototypes are given below. Although in the given `main` function, the puzzle is initialized with some hard-coded values, your solution should work with any properly initialized 6x6 mini-Sudoku puzzles.

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

void display(const int puzzle[ ][6]);                                  // Part (a)
bool check_full(const int puzzle[ ][6]);                               // Part (b)
bool check_rules(const int puzzle[ ][6], int number, int row, int column);   // Part (c)
```

```
int main( )                                                   // The Mini Sudoku puzzle
{
    int puzzle[6][6] = { {1, 0, 5, 3, 4, 0}, {0, 4, 6, 0, 2, 0},
                         {2, 5, 0, 4, 6, 3}, {4, 0, 3, 2, 0, 1},
                         {5, 1, 0, 6, 3, 4}, {0, 3, 4, 0, 1, 2} };
    int row = 0, column = 0, number = 0;
    char cmd = ’y’;

    while (cmd == ’y’)
    {
        display(puzzle);                                      // Print the puzzle

        cout ≪ "Please choose the position [1-6]." ≪ endl;
        cout ≪ "e.g.  To place a number on the 2nd row, the 3rd column, ";
        cout ≪ "the input will be:  2 3" ≪ endl;
        cin ≫ row ≫ column;
        cout ≪ "Please enter the number between 1 to 6:  ";
        cin ≫ number;

        if (check_rules(puzzle, number, row, column)) // Check if the placement is valid
        {
            puzzle[row-1][column-1] = number;                 // Update the puzzle

            if (check_full(puzzle))
            {
                cout ≪ "Congratulations!  The puzzle is solved!" ≪ endl;
                break;
            }

            cout ≪ "Great choice :)  Please continue!" ≪ endl;
        }
        else
            cout ≪ "Seems to be a bad choice :( Try again!" ≪ endl;

        cout ≪ "Continue (y/n)?  ";
        cin ≫ cmd;
    }

    return 0;
}
```

(a) Write the `display` function which takes a mini-Sudoku puzzle as the input and prints the puzzle in a 6x6 grid format like the following:

```
1 0 5 3 4 0
0 4 6 0 2 0
2 5 0 4 6 3
4 0 3 2 0 1
5 1 0 6 3 4
0 3 4 0 1 2
```

```
void display(const int puzzle[ ][6])
{       // Part (a): ADD YOUR CODE HERE















}
```

(b) Write the `check_full` function that takes a mini-Sudoku puzzle as the input parameter and returns true if all the cells in the puzzle are filled with numbers between 1 to 6, otherwise, it returns false.

```
bool check_full(const int puzzle[ ][6])
{       // Part (b): ADD YOUR CODE HERE















}
```

(c) Write the `check_rules` function that takes as the input parameters, a mini-Sudoku puzzle, a number (between 1 and 6), and a row index and a column index which together represent the position of the number to be placed. It checks if the number to be placed on the position (row, column) fulfills the mini-Sudoku rules described above. The function returns true if the number placement is valid (i.e., it satisfies the mini-Sudoku rules), otherwise false. You may assume that a user will always enter an integer number between 1 to 6 for both the row, the column and the number.

```
bool check_rules(const int puzzle[ ][6], int number, int row, int column)
{
    // Part (c): ADD YOUR CODE HERE












}
```

/* Rough work — You may detach this page */

/* Rough work — You may detach this page */

/* Rough work — You may detach this page */