COMP 2011 Midterm Exam - Fall 2019 - HKUST

Date: November 2, 2019 (Saturday)
Time Allowed: 2 hours, 2–4pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **6** questions on **18** 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 **cannot** use any library functions not mentioned, nor the `auto` keyword in defining identifiers in the questions.

Student Name	
Student ID	
Email Address	
Lecture & Lab Section	

For T.A. Use Only

Problem	Score
1	/ 12
2	/ 9
3	/ 18
4	/ 16
5	/ 25
6	/ 20
Total	/ 100

EMPTY PAGE

Problem 1 [12 points] C++ Controls

What is the output of the following program?

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

int main()
{
    int num = 1;

    while (true)
    {
        ++num;

        if (num == 8)
            break;

        if (!(num % 3))
            continue;

        if (num % 2)
            num -= 3;
        else
            num += 3;

        cout << num << " ";
    }

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

Answer: ____

Problem 2 [9 points] Parameters Passing and Return Type

Fill in the blanks in the header of the following function, `fun` , with appropriate parameter-passing mechanism and return type, so that the program can be compiled successfully without any compilation errors and the output is:

```
6
60
1, 1200
```

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

_______________ fun( _______________ x, _______________ y)

{
   x = 2 * x;
   y = y * x;
   return y;
}


int main()
{
   int a = 1, b = 3, result = 0;

   cout << fun(a, b) << endl;

   result = fun(a+4, b);
   cout << result << endl;

   fun(a, b) *= 10;
   cout << a << ", " << b << endl;

   return 0;
}
```

Problem 3 [18 points] Lambda Expressions

Write down the output for each of the following codes. If the code cannot be compiled, write "COMPILATION ERROR". If it compiles but may crash when it is run, write "RUNTIME ERROR".

(a)
```
float x = -2, y = 1;
cout << [](float a, float b) { a=(a<0)?-a:a; return ((a+b)/2); } (x, y) << ", ";
cout << x << ", " << y << endl;
```

Answer: ____________________

(b)
```
float x = -2, y = 1;
cout << [](float a, float b) -> int { a=(a<0)?-a:a; return ((a+b)/2); } (x, y)
     << ", ";
cout << x << ", " << y << endl;
```

Answer: ____________________

(c)
```
float x = -2, y = 1;
cout << [=]() { x=(x<0)?-x:x; return ((x+y)/2); } () << ", ";
cout << x << ", " << y << endl;
```

Answer: ____________________

(d)
```
float x = -2, y = 1;
cout << [=]() mutable { x=(x<0)?-x:x; return ((x+y)/2); } () << ", ";
cout << x << ", " << y << endl;
```

Answer: ____________________

(e)
```
float x = -2, y = 1;
cout << [&]() { x=(x<0)?-x:x; return ((x+y)/2); } () << ", ";
cout << x << ", " << y << endl;
```

Answer: ____________________

(f)
```
float x = 1, y = 0;
auto f = [&x, y]() { x=(x<0)?-x:x; return ((x+y)/2); };
x = -2; y = 1;
cout << f(); cout << ", " << x << ", " << y << endl;
```

Answer: ____________________

Problem 4 [16 points] Recursion: Same or Assimilation Game

In your lab5, you should have written a recursive solution for the “Same or Assimilation” game.

This problem is a modified version of the game: the game board is smaller with 6×6 cells, and there are only 2 symbols ‘A’ or ‘B’. Here is a brief description of the game again. The top leftmost corner has the coordinates $(0, 0)$ while the bottom rightmost cell has the coordinates $(5, 5)$. It is an iterative game and at each iteration, the symbol originally at $(0, 0)$ and at all its connected cells with the same symbol are changed to the other symbol.

```
 1  /* Input parameters:
 2     1st parameter: 2D game board of size 6 x 6
 3     2nd parameter: old symbol to be replaced
 4     3rd parameter: new symbol used to replace the old symbol
 5     4th parameter: row index of the location to start the replacement
 6     5th parameter: column index of the location to start the replacement
 7
 8     It starts at a given location (4th & 5th parameters).
 9     - If its symbol at the location is NOT the same as the old symbol, do nothing.
10     - Otherwise, replace it by the new symbol, and then recursively update the
11       symbols of all array elements which are connected with it (i.e., the 4 elements
12       which are East, West, North, and South of it).
13  */
14
15  void lab5_core_F(char board[][6], char old_symbol, char new_symbol, int row, int col)
16  {
17     if (row < 0 || row >= 6 || col < 0 || col >= 6 || board[row][col] != old_symbol)
18        return;
19
20     board[row][col] = new_symbol;
21     cout << '(' << row << ", " << col << ')' << endl;         /**** ADDED ****/
22
23     lab5_core_F(board, old_symbol, new_symbol, row, col + 1); // East
24     lab5_core_F(board, old_symbol, new_symbol, row, col - 1); // West
25     lab5_core_F(board, old_symbol, new_symbol, row - 1, col); // North
26     lab5_core_F(board, old_symbol, new_symbol, row + 1, col); // South
27  }
```

Above is the core recursive function, `lab5_core_F()`, in the solution. But the program is modified by adding line #21 to print out the cell whose symbol is changed. The game board is initialized as shown in the next page.

```
      0   1   2   3   4   5
     --- --- --- --- --- ---
0   | A | A | A | A | B | A |
     --- --- --- --- --- ---
1   | A | A | A | B | B | A |
     --- --- --- --- --- ---
2   | A | A | B | A | B | B |
     --- --- --- --- --- ---
3   | B | A | A | B | B | B |
     --- --- --- --- --- ---
4   | A | B | A | A | B | B |
     --- --- --- --- --- ---
5   | A | B | A | A | A | B |
     --- --- --- --- --- ---
```

The recursive function is first called from the `main()` as follows:

```
lab5_core_F(board, board[0][0], 'B', 0, 0)
```

and the program starts to flip some cells from ‘A’ to ‘B’ and prints out their coordinates. Note that there will be no output for cells that are reached but their contents are not changed. The first 2 lines of outputs are given. Write down the next 8 lines of outputs in their correct order, which represent the order of cells whose contents will be changed from ‘A’ to ‘B’.

Answer:

```
(0, 0)
(0, 1)
```

____

____

____

____

____

____

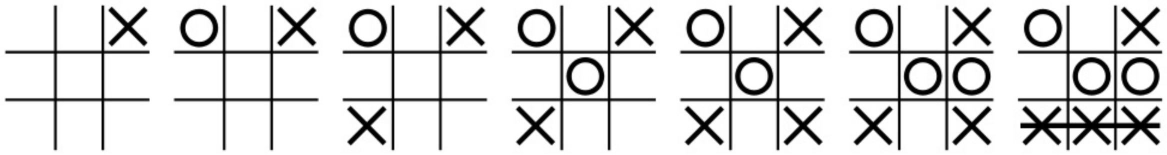
____

____

Problem 5 [25 points] Array and Structure

Tic-tac-toe is a classic game for two players, X and O, who take turns marking the cells in a 3×3 game board. The player who succeeds in placing three of his/her marks in a horizontal, vertical, or diagonal line is the winner. If all cells are marked and no player wins, it's a tie.

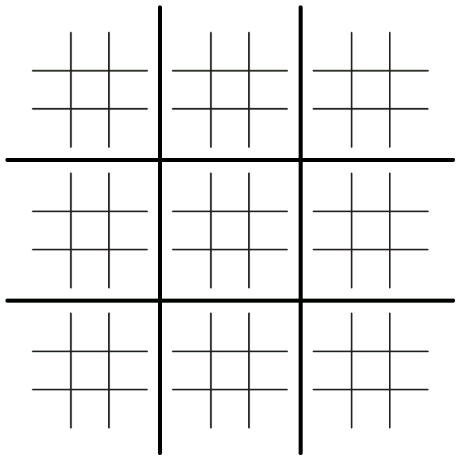
The following example game is won by the first player, X:



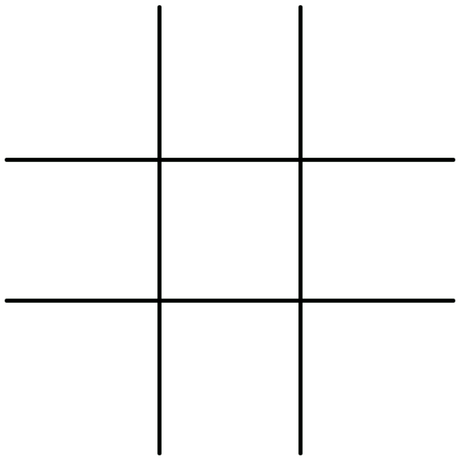
A simple Tic-tac-toe

We enjoyed the game when we were young, but it just isn't as exciting as it used to be. Let's triple the fun with Super Tic-tac-toe!

We are all familiar with Tic-tac-toe, the 3×3 game board. Well, in Super Tic-tac-toe, you play on a 9×9 physical game board, constructed by 3×3 usual Tic-tac-toe game boards. In other words, you may think of having a 3×3 "super board" in which each cell is a usual Tic-tac-toe board. To win the game, you have to win three of the large cells in the super board in a line (vertical, horizontal or diagonal). And to win a large cell, you have to win the usual Tic-tac-toe game in the cell.



9×9 physical board for players



3×3 super board for internal usage

To implment the Super Tic-tac-toe game, the following data structures are defined in a header file `superTictactoe.h`.

```
/* File: superTictactoe.h */
const int N = 3;

// In small Tic-tac-toe, a cell is
//     EMPTY: when no player has marked it
//     PLAYER1/PLAYER2: when the player has marked it
// In super board of Super Tic-tac-toe, a cell is
//     EMPTY: when no player has won the corresponding small Tic-tac-toe
//     PLAYER1/PLAYER2: when the player has won the corresponding small Tic-tac-toe
//     TIE: when there is a tie in the corresponding small Tic-tac-toe
enum Status {EMPTY, PLAYER1, PLAYER2, TIE};

struct Array{
    Status grid[N][N];          // 3x3 2D array
};

struct TicTacToe { // Super Tic-tac-toe struct
    Array physicalboard[N][N];  // 9x9 physical board, visible to player
    Array superboard;           // 3x3 super board, internal usage
};
```

Three utility functions for `Array` are provided.

```
/*  Utility functions for Array struct */
void setArray(Array& array, Status status){
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            array.grid[i][j] = status;
}

void copyArray(Array& dest, const Array& src){
    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            dest.grid[i][j] = src.grid[i][j];
}

void printArray(const Array& array){
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++)
            cout << array.grid[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}
```

Below is a sample output for the last two steps of a Super Tic-tac-toe game execution. X enventually wins the game.

```
Player 2: (O)
Please give the cell location(row column):  (range [0, 8])
8 6
Physical board:
   0  1  2   3  4  5   6  7  8
 |---------|---------|---------|
0| X       |    X  O |         |
1| X  O    |    O  X |         |
2| X     O | O     O |         |
 |---------|---------|---------|
3| X  O  O | O  X  O | X  X    |
4| O  X  X | X  X  X | O  O    |
5| X  O  X | O  X  X |         |
 |---------|---------|---------|
6| O  X  O |         |         |
7| X  X  O |         |    X    |
8| O  O  X |         | O       |
 |---------|---------|---------|
Super board:
1 2 0
1 1 0
3 0 0

Player 1: (X)
Please give the cell location(row column):  (range [0, 8])
3 8
Player 1 wins a small tic-tac-toe!
Player 1 wins the game!!!
Physical board:
   0  1  2   3  4  5   6  7  8
 |---------|---------|---------|
0| X       |    X  O |         |
1| X  O    |    O  X |         |
2| X     O | O     O |         |
 |---------|---------|---------|
3| X  O  O | O  X  O | X  X  X |
4| O  X  X | X  X  X | O  O    |
5| X  O  X | O  X  X |         |
 |---------|---------|---------|
6| O  X  O |         |         |
7| X  X  O |         |    X    |
8| O  O  X |         | O       |
 |---------|---------|---------|
Super board:
1 2 0
1 1 1
3 0 0
```

In this question, you'll need to work on a few key functions in SuperTictactoe C++ program.

(a) [6 points] Implement the function, `void initGame(TicTacToe& game)`.

```
// Aim:   Initialize the Super Tic-tac-toe.
//        Cells in both physical board and super board are set to EMPTY.
// Input: game - a struct which holds both the physical and super board.

void initGame(TicTacToe& game)
```

(b) [10 points] Implement the function `bool check3inline(const Array& array)`.

Feel free to use the helper function `same` if needed.

```
bool same(Status a, Status b, Status c) {
        return (a == b && b == c);
}

// Aim:    Check whether there's a a horizontal, vertical,
//         or diagonal line marked.
// Input:  array - a struct which holds a 3x3 board.
// Output: true if found 3-in-a-line, false otherwise.

bool check3inline(const Array& array)
```

(c) [9 points] Complete function `void printBoards(const TicTacToe& game)`. The output format should be in the SAME format as the sample output below.

```
Physical board:
   0  1  2   3  4  5   6  7  8
 |---------|---------|---------|
0| X        |    X  O |         |
1| X  O    |    O  X |         |
2| X     O | O     O |         |
 |---------|---------|---------|
3| X  O  O | O  X  O | X  X    |
4| O  X  X | X  X  X | O  O    |
5| X  O  X | O  X  X |         |
 |---------|---------|---------|
6| O  X  O |         |         |
7| X  X  O |         |    X    |
8| O  O  X |         | O       |
 |---------|---------|---------|
```

```cpp
// Aim:  Visualize the game board
// Input: game - a struct which holds both the physical and super board
void printBoards(const TicTacToe& game) {
    const char symbol[] = {' ', 'X', 'O'}; // ' ': empty, 'X': player1, 'O': player2
    cout << "Physical board:" << endl;
    cout << "   0  1  2   3  4  5   6  7  8" << endl;
    cout << " |---------|---------|---------|" << endl;
    // YOUR CODE STARTS HERE







}
```

Problem 6 [20 points] C String, Array and Recursion

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

int string2int(const char s[], int x[]);
bool is_pingpong(const int x[], int num_digits, int stepsize = 1, int diff = 1);
void to_pingpong(int x[], int num_digits);

/********* YOU DON'T NEED TO IMPLEMENT THIS FUNCTION **********/
// Aim:    To convert a non-negative integer in the array x[] digit by digit to
//         a C string in s[]. The number of digits in the integer is num_digits.
// Return: Nothing
void int2string(const int x[], int num_digits, char s[]);

int main()
{
    const int LENGTH = 10; // Max number of digits in the given number
    char s[LENGTH+1];      // Given number as a C string
    char s2[LENGTH+1];

    cout << "Enter the next number: ";
    while (cin >> s)
    {
        int x[LENGTH] = { }; // Store each digit in an int array
        int num_digits = string2int(s, x);
        bool is_ppn = is_pingpong(x, num_digits);

        cout << "Is " << s << " a PPN: " << boolalpha << is_ppn << endl;
        cout << "Is " << s << " a generalized (2, 1)-PPN: "
             << boolalpha << is_pingpong(x, num_digits, 2, 1) << endl;
        cout << "Is " << s << " a generalized (2, 2)-PPN: "
             << boolalpha << is_pingpong(x, num_digits, 2, 2) << endl;

        if (!is_ppn)
        {
            to_pingpong(x, num_digits);
            int2string(x, num_digits, s2);
            cout << s << " is converted to this PPN: " << s2 << endl;
        }

        cout << "Enter the next number: ";
    };
    return 0;
}
/***** Definitions of the above 4 functions are supposed to be here ******/
```

In Assignment 1, you are asked to determine if a given non-negative integer is a *ping-pong number* (PPN). Again, an integer is a PPN if all its adjacent digits differ by 1. All single-digit numbers are considered as PPNs. For example, 4 and 567678 are PPNs, while 14 and 21098 are not. At that time, you are not allowed to use array. In this midterm question, however, we would like you to do that exactly using array(s) instead.

The driver program is given above, which, in each "while" iteration, reads in an input non-negative number as a C string, converts it to an integer and puts it in an int array digit by digit, and checks if it is a PPN or a generalized PPN with various step sizes and differences. If it is not a PPN, at the end of the iteration, it will be converted to a PPN with the following rules:

- if any prefix of the number is a PPN, keep the prefix unchanged (e.g., 1, 12, 123, 1235 and 12356 are prefixes of the number 12356, and 123 is its greatest prefix that is a PPN),
- then change the remaining digits so that the converted number will be **the smallest PPN possible with the prefix**. Here are some examples: 123 → 123, 12358 → 12321, 4680 → 4321, 29988 → 21010.

Study the program output for some example inputs below, and then answer the 4 questions afterwards. For the coding questions, you may use any other functions declared in the driver program. You may assume the inputs are always valid as well.

```
Enter the next number: 12345
Is 12345 a PPN: true
Is 12345 a generalized (2, 1)-PPN: false
Is 12345 a generalized (2, 2)-PPN: true
Enter the next number: 192837
Is 192837 a PPN: false
Is 192837 a generalized (2, 1)-PPN: true
Is 192837 a generalized (2, 2)-PPN: false
192837 is converted to this PPN: 101010
Enter the next number: 4680
Is 4680 a PPN: false
Is 4680 a generalized (2, 1)-PPN: false
Is 4680 a generalized (2, 2)-PPN: false
4680 is converted to this PPN: 4321
Enter the next number: 28
Is 28 a PPN: false
Is 28 a generalized (2, 1)-PPN: true
Is 28 a generalized (2, 2)-PPN: true
28 is converted to this PPN: 21
```

(a) [2 points] 45778 is not a PPN. Convert it to a PPN according to the 2 rules given above.

Answer: ____

(b) [4 points] Implement the function `string2int`.

```
// Aim: To convert a non-negative integer stored as a C string in s[] to the
//      int array x[] digit by digit. E.g., if s contains ''123'' then after
//      calling this function, x[0]=1, x[1]=2, x[2]=3.
// Return: number of digits in s (and equivalently in x).
int string2int(const char s[], int x[])
```

(c) [7 points] Implement the function, `is_pingpong`. Your solution must be a **recursive** function. No loops in your code. Iterative solution receives 0 point.

```
// Aim:     Check if the given integer in x is a generalized ping-pong number.
// Inputs:
//    - x: stores a non-negative integer digit by digit in an int array
//    - numb_digits: number of digits in the given number
//    - stepsize: Between 1 to 4 inclusive. Default value = 1.
//    - diff: Between 1 to 4 inclusive. Default value = 1.
bool is_pingpong(const int x[], int num_digits, int stepsize, int diff)
```

(d) [7 points] Implement the function `to_pingpong`. Your solution must be a **recursive** function. No loops in your code. Iterative solution receives 0 point.

```
// Aim:    If the given integer stored in x[] with the given number of digits
//         (num_digits) is NOT a ping-pong number, convert it to a ping-pong
//         according to the rules in the question.
void to_pingpong(int x[], int num_digits)
```

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