COMP 2011 Final - Spring 2017 – HKUST

- Date: May 22, 2017 (Monday)
- Time Allowed: 3 hours, 4:30 pm - 7:30 pm
- Instructions:
    1. This is a closed-book examination.
    2. There are 9 questions on 31 pages (including the 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 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	
Student ID	
ITSC email	
Seat No	

For T.A. Use Only

Problem	Score / Max. score
1	/ 11
2	/ 4
3	/ 8
4	/ 9
5	/ 8
6	/ 15
7	/ 15
8	/ 15
9	/ 15
Total	**/ 100**

Problem 1 *[11 marks]* C++ Basic Programming

In this question, you are implementing a time calculator. The following is the incomplete program:

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

bool isDigit(char c);  // Given
int charToInt(char c);  // Given
bool strToTime( /* to be completed in Part (a) */ );  // Part a
void timeAddition( /* to be completed in Part (b) */ );  // Part b

int main()
{
   char timeStr[99];
   cout << "What time is it now? ";
   cin >> timeStr;
   int hour = 0, minute = 0;
   if (strToTime(timeStr, hour, minute))
   {
      cout << "The time now is " << hour << ":" << minute << endl;
      timeAddition(hour, minute, 17);
      cout << "After 17 minutes, the time will be "
           << hour << ":" << minute << endl;
      timeAddition(hour, minute, 15);
      cout << "After a further 15 minutes, the time will be "
           << hour << ":" << minute << endl;
      timeAddition(hour, minute, 360);
      cout << "After a further 360 minutes, the time will be "
           << hour << ":" << minute << endl;
   }
   return 0;
}
```

and the corresponding sample output:

```
What time is it now? 20:40
The time now is 20:40
After 17 minutes, the time will be 20:57
After a further 15 minutes, the time will be 21:12
After a further 360 minutes, the time will be 3:12
```

(a) [6 marks] As shown in the main function above, the user will input the current time as a C string in "HH:MM" format and 24-hour format. You need to complete a function **`strToTime`** to "extract" the hour and minute from a C string to integers.

The function has 3 parameters. Parameters *hour* and *minute* are integer variables that are **passed by reference**. Parameter **`str`** is a constant C string (array of character).

This function returns a **Boolean value**. It returns true if the format is valid, otherwise, it returns false.

A valid format "HH:MM" means

- The string length must be 5

- The 1st , 2nd, 4th and 5th characters must be digits (i.e. between '0' to '9'), the 3rd character must be ':' (i.e. the colon).
- HH lies between 00 and 23 inclusively. MM lies between 00 and 59 inclusively.

For example, suppose **h** and **m** are integer variables,

- **strToTime("20:11", h, m)** returns true, and **h** and **m** will be 20 and 11 respectively.
- **strToTime("07:05", h, m)** returns true, and **h** and **m** will be 7 and 5 respectively.
- **strToTime("99:99", h, m)** returns false.
- **strToTime("COMP2011", h, m)** returns false.

Fill-in the blanks and complete the definition of the **strToTime** function in the box below.
You may use the following functions:

```
bool isDigit(char c);
```

which returns true if the character, **c**, is a digit, i.e. '0', '1', …, '9'.

```
int charToInt(char c);
```

which returns the integer value of a digit, **c**, i.e. 0 for the digit '0', 1 for the digit '1', etc.
You may also use the following library function:

```
int strlen(const str[]);
```

which returns the length of a C string **str**.
You cannot call any other external functions and you cannot use the string class.

```
bool strToTime(const char str[], _____________ hour, _____________ minute)
{
   // Add your code here









}
```

(b) [5 marks] Next, you need to complete the function **timeAddition** that has 3 parameters to calculate the time after a certain amount of minutes. Parameters **hour** and **minute** are integer variables that are **passed by reference**. Parameter **minuteAdded** is an integer variable.

Note that **hour** should be in the range of 0 and 23 inclusively and **minute** should be in the range of 0 and 59 inclusively. You will need to handle the carry if they exceed the range (i.e., when hour >= 24 or when minute >= 60).

For example, suppose h = 20 and m = 11,

- After calling **timeAddition(h, m, 17)**, **h** and **m** will be 20 and 28 respectively.
- After calling **timeAddition(h, m, 118)**, **h** and **m** will be 22 and 9 respectively.
- After calling **timeAddition(h, m, 349)**, **h** and **m** will be 2 and 0 respectively.

Fill-in the blanks and complete the definition of the **timeAddition** function. You cannot use any external functions.

```
void timeAddition( __________ hour, __________ minute, int minuteAdded)
{
   // Add your code here









}
```

Problem 2 *[4 marks]* Scope and Function Overloading

What is the output of the following program?

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

int n = 10;

void function(double a, int b, int& n) {
   for (n = 1; n < 2; n++) {
      double& n = a;
      if (a > b) {
               int& n = b;
         n++;
      } else
         n = b;
      n++;
   }
   n = a + b;
   cout << "fn1"<< endl;
}

void function(double x, double y, int& number);

int main() {
   function(3.0, 3.3, n);
   cout << n << endl;
   return 0;
}

void function(double a, double b, int& n)  {
   for (n = 1; n < 2; n++) {
      double& n = a;
      if (a > b) {
         double& n = b;
         n++;
      } else
         n = b;
      n++;
   }
   n = a + b;
   cout << "fn2" <<endl;
}
```

Answer: _____

Problem 3 *[8 marks]* C++ Class

If the code given below is compiled as it is, there will be compilation errors. List the lines with errors and explain what those errors are.

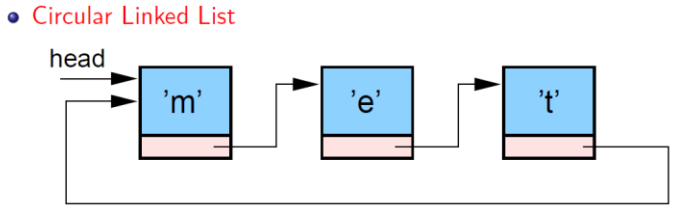
```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Course
5.  {
6.     private:
7.        int courseCode;
8.        int numStudents;
9.        void setDifficulty(float d);
10.
11.    public:
12.       Course();
13.       float finalDifficulty;
14.       void giveEveryoneAs();
15.       void setEasyFinal() const;
16. };
17.
18. Course::Course() { courseCode = 0; numStudents = 0; }
19.
20. void Course::giveEveryoneAs()
21. {
22.    cout << "Everyone gets an A+" << endl;
23. }
24.
25. void Course::setDifficulty(float d)
26. {
27.    finalDifficulty = d;
28. }
29.
30. void Course::setEasyFinal() const
31. {
32.    finalDifficulty = 0;
33. }
34.
35. int main()
36. {
37.    Course comp2011(10);
38.    comp2011.setDifficulty(2.5);
39.    comp2011.giveEveryoneAs();
40.    comp2011.numStudents = 100;
41.    comp2011.setEasyFinal();
42.
43.    return 0;
44. }
```

Fill in the following table:

Line Number	Code with Error	Reason for Error

Problem 4 *[9 marks]* Circular Linked List

A circular linked list is a special type of linked list such that the tail of the list is pointing to the head of the linked list, as illustrated in the following diagram:



Be aware that, in this example, none of the nodes on this list has a next pointer pointing to NULL.

Given the following structure definition of a node in the circular linked list:

```
struct ll_node
{
    char data;
    ll_node* next;
};
```

(a) [4 marks] Implement the function **`int count_length(const ll_node* head)`** which returns the number of nodes on any circular linked list.

```
int count_length(const ll_node* head)
{
   const ll_node* tmp = head;

   // Add you code here









}
```

(b) [5 marks] Implement another function **void delete_first_node(ll_node*& head)** which deletes the first node in the circular linked list pointed by head.

```
void delete_first_node(ll_node*& head)
{
   // Add you code here









}
```

Problem 5 *[8 marks]* Recursion

In this question, your task is to write a recursive function **findSmallestOdd** which finds and returns the smallest odd number in the given array. Otherwise, it returns 0 if there is no odd number in the given array.

For example, with the main function below:

```
int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int b[] = {5, 4, 3, 2, 1};
    int c[] = {4, 2, 3, 7, 5};
    int d[] = {10, 13, 11, 14, 12, 15, 7, 2, 5};

    cout << findSmallestOdd(a, 5) << endl;
    cout << findSmallestOdd(b, 5) << endl;
    cout << findSmallestOdd(c, 5) << endl;
    cout << findSmallestOdd(d, 9) << endl;
    return 0;
}
```

The following will be the output:

```
1
1
3
5
```

Complete the recursive function **findSmallestOdd** below. No loop of any kind is allowed.

```
int findSmallestOdd(int array[], int size)
{
    // Add your code here









}
```

Problem 6 *[15 marks]* C++ Class Definition

Suppose that you are working for the ITSC and you have been given a task to program the Student Print Budget Top Up Feature. A **`Transaction`** class is given to you in the header file, transaction.h, with the following class definition:

```
enum TransactionType { CASH, CREDIT_CARD };

class Transaction
{
   private:
      TransactionType type;
      int amount;

   public:
      Transaction(TransactionType, int);  // initialize the type & amount
      void setTransaction(TransactionType, int); // set the type & amount
      TransactionType getTransactionType() const;          // return type
      int getAmount() const;                              // return amount
};
```

You may assume all the member functions of the **Transaction** class have been defined in a separate source file, transaction.cpp.

You have to define and implement the **`PrintBudget`** class in printbudget.h and printbudget.cpp respectively according to the followings:

- There are 3 private member variables:
  - A pointer to a dynamic character array, **`username`**. It is for storing the username of the corresponding ITSC account
  - An integer, **`balance`**, which is for storing the current balance, i.e. total amount, of print budget
  - A Transaction pointer, **`lastTransaction`**, which points to a dynamic Transaction object
- There are 4 public member functions:
  - A general constructor, which initializes the username and balance, and also sets the **`lastTransaction`** pointer to NULL
  - A destructor, which deallocates all the dynamically allocated members
  - An accessor, **`displayBudget`**, which displays the username and balance of the print budget
  - A mutator, **`addAmountToBudget`**, which validates a transaction and adds the corresponding amount to the current balance of the print budget. If a transaction is valid, it increments the balance by the amount and returns true, otherwise, returns false. The amount of a valid transaction should be:
    - Larger than zero and less than 100 if the transaction type is cash, i.e. **CASH**

- Between 20 to 100 inclusively if the transaction type is credit card, i.e. **CREDIT_CARD**

Moreover, if the transaction is valid, it also modifies the **lastTransaction** pointer to points to the transaction.

Both the **Transaction** and **PrintBudget** classes will be used in the following main function in main.cpp:

```
#include <iostream>
#include "printbudget.h"

using namespace std;

int main()
{
   Transaction *t1 = new Transaction(CASH, 200);
   PrintBudget pb1("Andrew", 100);
   cout << "Current Print Budget:" << endl;
   pb1.displayBudget();
   if (pb1.addAmountToBudget(t1))
   {
      cout << "Updated Print Budget:" << endl;
      pb1.displayBudget();
   } else
      cout << "Failed to add the print budget" << endl;

   PrintBudget pb2("Sathish");
   cout << "Current Print Budget:" << endl;
   pb2.displayBudget();
   if (pb2.addAmountToBudget(new Transaction(CREDIT_CARD, 60)))
   {
      cout << "Updated Print Budget:" << endl;
      pb2.displayBudget();
   } else
      cout << "Failed to add the print budget" << endl;

   return 0;
}
```

The program is compiled by:

```
g++ -c transaction.cpp
g++ -c printbudget.cpp
g++ -c main.cpp
g++ -o printbudget main.o printbudget.o transaction.o
```

And executed with the corresponding output:

```
Current Print Budget:
Username:Andrew
Balance:100
Failed to add the print budget
Current Print Budget:
Username:Sathish
Balance:300
Updated Print Budget:
Username:Sathish
Balance:360
```

(a) [5 marks] Complete the **PrintBudget** class definition in the header file, printbudget.h.

```
#include <iostream>
#include <cstring>
#include "transaction.h"
using namespace std;

class PrintBudget
{
   private:
      // Add the private members here









   public:
      // Add the prototypes of the public member functions here









};
```

(b) [10 marks] Implement the member functions of the **PrintBudget** class in the source file, printbudget.cpp. You may use the library functions

```
char* strcpy(char* dest, const char* source);
```

which copies the contents of the C string **source** to the C string **dest**, or

```
int strlen(const char* str);
```

which returns the length of the C string **str**. You cannot use any other library functions.

```
#include "printbudget.h"
// Define the constructor here
```

```
// Define the destructor here




// Define the member function displayBudget here




// Define the member function addAmountToBudget here
```

Problem 7 *[15 marks]* Dynamic Arrays

In the lab exercises, you have implemented the board game of finding and avoiding different agents. Now, in this question, you are asked to do something similar. You have to find the closest TA for our students.

You need to write a C++ program to simulate students finding the closest TA in the lab. Assume the lab is in the form similar to a game board, with LAB_ROWS rows and LAB_COLUMNS columns, which are defined as constant integers:

```
const int LAB_ROWS = 6;
const int LAB_COLUMNS = 15;
```

Suppose the structure definitions of **PersonPosition** and **Lab** are given to you below.

```
struct PersonPosition
{
    int row, column;
};

struct Lab
{
    int numOfTAs, numOfStudents;
    PersonPosition *TAs, *students;

};
```

PersonPosition structure has two members, **row** and **column**. They represent the row number and the column number that a person is on the game board. The **row** and **column** variables of **PersonPosition** is in the range of [0, LAB_ROWS) and [0, LAB_COLUMNS) respectively (i.e. 0 ≤ **row** < LAB_ROWS and 0 ≤ **column** < LAB_COLUMNS).

Lab structure has four member. The pointer members, TAs and students, should point to dynamic arrays of **PersonPosition** objects, while the integer members, **numOfTAs** and **numOfStudents**, represent the number of TAs and students respectively.

Now you need to implement some functions. The following main function is given as an example:

```
int main() {
   srand(time(0));
   Lab lab = {3, 67};
   // initialize the lab with randomly assigned TAs and students
   initializeAndRandomizeLab(&lab);
   // find the closest TA for each student in the lab
   calculateTheClosestTAs(&lab);
}
```

Note that you are NOT allowed to use any library functions unless specified.

(a) [2 marks] Complete the **calculateManhattanDistance** function based on the given prototype. It returns the Manhattan distance of two **PersonPosition** objects, **pos1** and **pos2**. Recall that the Manhattan distance formula is $|x_1 - x_2| + |y_1 - y_2|$. You CANNOT use any library function.

```
int calculateManhattanDistance(const PersonPosition* pos1,
                               const PersonPosition* pos2)
{
   // Add you code here









}
```

(b) [5 marks] Fill-in the blanks and complete the **assignRandomPositions** function based on the given prototype and the incomplete definition. The first parameter, **people**, points to a dynamic array of **PersonPosition** objects and the second parameter, **numOfPeople**, is the number of **PersonPosition** objects to be considered, i.e., the array size of **people**. For every element of the array, **people**, assign a random position in the lab such that every element in the array, **people**, is different (i.e., not in the same position). Be reminded that the lab dimension is LAB_ROWS * LAB_COLUMNS and you can assume that it is always big enough for all students and TAs. You may use the **calculateManhattanDistance** function in part (a) and the library function

```
int rand();
```

which returns a random integral number.

You may assume that **srand(time(NULL))** is already called.

```
void assignRandomPositions(PersonPosition* people, int numOfPeople)
{

   // Iterate through each element in the people array

   for ( ____________________________________________________ ){
      bool isPositionRepeated = false;
      PersonPosition* pos = new PersonPosition;

      // Keep looping if the generated position is repeated,
      // i.e. already generated for the previous positions
      do {

         // Add your code here









      } while (isPositionRepeated);


      ____________________________________________________________
      delete pos;
   }
}
```

(c) [3 marks] Complete the **initializeAndRandomizeLab** function which initializes the Lab object, **lab**. You can assume that the **numOfTAs** and **numOfStudents** of lab have already been assigned with some integers. Then, you need to dynamically allocate **TAs** and **students** as arrays of sizes **numOfTAs** and **numOfStudents** respectively. Afterwards, separately call **assignRandomPositions** function to randomly assign positions to the two arrays, **TAs** and **students**. The position of a TA and a student can be the same.

```
void initializeAndRandomizeLab(Lab* lab)
{
   // Add your code here









}
```

(d) [5 marks] Complete the **calculateTheClosestTAs** function based on the given prototype and incomplete definition. For every student of the lab, the function will output the closest TA of the student, i.e., the TA with the least Manhattan distance. You may use the **calculateManhattanDistance** function in part (a).

```
void calculateTheClosestTAs(const Lab* lab)
{

   // Iterate through all students in the lab
   for (int i = 0; i < lab->numOfStudents; i++) {

      // closestTA is the array index of the closest TA
      int closestTA = -1;

      // Add your code here

      cout << "Student " << i+1 << " should find TA "
           << closestTA+1 << endl;

   }
}
```

Problem 8 *[15 marks]* Dynamic Data Structure

WannaCry is a "ransomewere" which attacks computers running Microsoft Windows. If a computer is attacked, the files may be encrypted (locked with secret) and the files name will be appended ".WCRY" at the end. You are required to write a program to simulate a file system and count the number of encrypted files by checking if the files have ".WCRY" at the end.
Suppose a file system has two types of items, namely, file or folder, only. They are represented by **`TYPE_FILE`** and **`TYPE_FOLDER`** from the enum **`ItemType`**.

```
enum ItemType
{
   TYPE_FILE, TYPE_FOLDER
};
```

If an item is a folder, it can have two types of sub-items in it.
Therefore, to represent an item in the file system, the following structure is defined:

```
struct Item
{
   char name[255]; // name of the file/folder
   ItemType type; // type of the file/folder,
                  // i.e. either TYPE_FILE or TYPE_FOLDER
   Item** subItems; // dynamic array of Item* of array size subItemCount
   int subItemCount; // array size of subItems
};
```

If an Item object has the **`TYPE_FILE`** type, then the **`subItems`** pointer will be set to NULL and **`subItemCount`** will bet set to 0.
If the Item object has the **`TYPE_FOLDER`** type, then the addresses of the sub-items will be stored in the dynamic array, **`subItems`**.

(a) [4 marks] Complete the definition of the function **constructItem** which returns a dynamically allocated item. The item should be initialized with the parameters, **name**, **type** and **subItemCount**. If the item type is folder and the **subItemCount** is larger than 0, you have to dynamically allocate the **subItems** to be an array of Item pointers with array size **subItemCount**, otherwise, set **subItems** to NULL.

For example, the function call **constructItem("COMP2011", TYPE_FOLDER, 3)** returns a dynamic item with its member variables, **name** as "COMP2011", **type** as TYPE_FOLDER, **subItems** pointing to a dynamic array of **Item** pointers, and **subItemCount** as 3.

You may use the library function

```
char* strcpy(char* destination, const char* source);
```

to copy the contents of the C string **source** to the C string **destination**.

```
Item* constructItem(const char* name, ItemType type = TYPE_FILE,
                    int subItemCount = 0)
{
   // Add your code here









}
```

(b) [4 marks] Suppose that there is a file system with the following structure.

- A folder “COMP2011”, it contains the following three items:
  - A file “notes.docx.WCRY”
  - A folder “assignment3”, it contains the following two items
    - A file “submit.cpp.WCRY”
    - A file “skeleton.zip.WCRY”
  - A file “pastpaper.txt”

Complete the **`createTestItem`** function which returns a pointer that points to the Item representing “COMP2011” and its file structure using the **`constructItem`** function defined in part (a).

```
Item* createTestItem()
{
   Item* root = constructItem("COMP2011", TYPE_FOLDER, 3);
   // Add your code here
   return root;
}
```

(c) [4 marks] Complete the **countWCryFile** function which counts the number of files that ends with ".WCRY" in any file system. For example,

countWCryFile(createTestItem()) returns 3
countWCryFile(constructItem("anything.WCRY")) returns 1
countWCryFile(constructItem("empty_folder", TYPE_FOLDER, 0)) returns 0

You may use the library function

```
int strlen(const char* str);
```

which returns the length of the C string **str**. You cannot use any other library functions.

```
int countWCryFile(const Item* item)
{
   // Add your code here









}
```

(d) [3 marks] Complete the **deleteAllItems** function which deallocates the whole file system pointed by **item** and reset the **item** pointer to NULL. It should be able to handle any file system.

```
void deleteAllItems(Item*& item)
{
   // Add your code here









}
```

Problem 9 Stack and Queue *[15 marks]*

Suppose you are an IT intern to President Donald Trump. He needs two sets of people, his **ADVISORS** and people to be in the **GOVERNMENT**. He managed the two sets of people using stack and queue.

- As years of service is valued in government, people in the **GOVERNMENT** is logically organized using a stack with LIFO – **the Last Hired Person will be the First Fired Person**.
- President Trump thinks queue is the best – so his **ADVISORS** will be logically organized using a queue with FIFO – **the First Hired Person will be the First Fired Person.**

Every month, orders will be issued to make changes to the two sets of people. The orders will be checked for legality (explained below), executed and reflected **in the next month**. The orders are in the format:

ACTION POST from/to ORGANIZATION

- **ACTION** can be either **Hire** or **Fire**
- **ORGANIZATION** will be either **GOVERNMENT** or **ADVISORS**
- **POST** will be the **title of a person** in ORGANIZATION

For examples,

- Hire SECY-STATE to Government (meaning "Push/Enqueue SECY-STATE to GOVERNMENT stack")
- Fire MEDIA-ADV from ADVISORS (meaning "Pop/Dequeue MEDIA-ADV from ADVISORS queue")

The order has to be legal according to the following:

- An order with action **Hire** to any Organization (GOVERNMENT or ADVISORS) is always legal and can be executed.
- An order with action **Fire** is legal only if it fires someone who is at the **top** of the GOVERNMENT stack or at the **front** of ADVISORS queue. These orders can be executed.
    - Note: If the person to be fired in the order is NOT at the **top** of GOVERNMENT stack or the **front** of ADVISORS queue, that order must be ignored and cannot be executed.

President Trump finds the above rules frustrating as he cannot fire people as he want. As a smart HKUST student, you offered him a work-around:

> "You can add the words, **AT ANY COST**, to the fire order. When an order "Fire X from Y AT ANY COST" is received, all people in organization Y, preceding X, will be fired by legitimate operations (dequeue or pop). Once X is at the **top** of GOVERNMENT stack or **front** of ADVISORS queue, X can be fired legally. Then, those preceding people will be re-hired, **in the same order of firing (first fired will be first hired back)** into the same organization. Finally, the organizations will still have the same people, but the person you want to fire is no longer there".

For example, given the following state of ADVISORS (queue)

ADVISORS (queue)

MEDIA-ADV	TRADE-ADV	HEALTH-ADV	ENERGY-ADV	FUNDING-ADV			

(front)

Order "Fire ENERGY-ADV from ADVISORS" is **illegal** and will not be executed. However, Order "Fire ENERGY-ADV from ADVISORS AT ANY COST" will be executed.

The new state of ADVISORS (queue) will be

FUNDING-ADV	MEDIA-ADV	TRADE-ADV	HEALTH-ADV				

(front)

(a) [10 marks] One of your internship duties is to track changes of people. Given below are the orders in the past five months. Modify the GOVERNMENT stack and ADVISORS queue below to display changes accurately. The first month's orders have been done as an example.

Notes:

- Only legal orders will be executed. The orders are executed sequentially.
- The GOVERNMENT stack and ADVISORS queue should show the combined changes of all orders issued in the previous month. Changes made in one month will be carried on to next month.

Date: 1 December 2016

GOVERNMENT (stack)

ATTY-GEN	NSA	SURG-GEN	FBI-DIR	PRESS-SECY			

(top)

ADVISORS (queue)

TRADE-ADV	SECURITY-ADV	TWITTER-ADV	HEALTH-ADV				

(front)

Orders:

1) Fire ATTY-GEN from GOVERNMENT.
2) Hire ACTING-ATTY-GEN to GOVERNMENT.

Date: 1 January 2017

GOVERNMENT (stack)

ACTING-ATTY-GEN	NSA	SURG-GEN	FBI-DIR	PRESS-SECY			

(top)

ADVISORS (queue)

TRADE-ADV	SECURITY-ADV	TWITTER-ADV	HEALTH-ADV				

(front)

Orders:

1) Fire TRADE-ADV from ADVISORS.
2) Hire ARMY-GEN to GOVERNMENT.

Complete the followings

Date: 1 February 2017

GOVERNMENT (Stack)

(top)

ADVISORS (Queue)

(front)

Orders:

1) Hire IVANKA to ADVISORS
2) Hire JARED to ADVISORS

Date: 1 March 2017

GOVERNMENT (Stack)

(top)

ADVISORS (Queue)

(front)

Orders:

1. Fire FBI-DIR from GOVERNMENT.
2. Fire SECURITY-ADV from ADVISORS.

Date: 1 April 2017

GOVERNMENT (Stack)

(top)

ADVISORS (Queue)

(front)

Orders:

1. Fire FBI-DIR from GOVERNMENT **AT ANY COST**.
2. Hire CIA-DIR to GOVERNMENT

Date: 1 May 2017

GOVERNMENT (Stack)

(top)

ADVISORS (Queue)

(front)

(b) [5 marks] We discussed in class the array implementations of stack and queue. They are further modified to store the title of a person:

```
#include <iostream>                                  /* File: stack-queue.h */
#include <cstdlib>
#include <cstring>

using namespace std;
const int BUFFER_SIZE = 100;
const int MAX_STR_LENGTH = 50;


class stack
{
  private:
    char data[BUFFER_SIZE][MAX_STR_LENGTH];
    int top_index;
  public:
    stack();                                          // Default constructor
    bool empty() const;                     // Check if the stack is empty
    bool full() const;                       // Check if the stack is full
    int size() const;       // Give the number of data currently stored
    const char* top() const;      // Retrieve the value of the top item
    void push(const char*);                  // Add a new item to the top
    void pop();                                       // Remove the top item
};

class queue // Circular queue
{
  private:
    char data[BUFFER_SIZE][MAX_STR_LENGTH];
    int num_items;
    int first;
  public:
    queue();                                           // Default constructor
    bool empty() const;                    // Check if the queue is empty
    bool full() const;                      // Check if the queue is full
    int size() const;       // Give the number of data currently stored
    const char* front() const; // Retrieve the value of the front item
    void enqueue(const char*);              // Add a new item to the back
    void dequeue();                          // Remove the front item
};
```

Now implement a non-member function, **`fire_from_stack_at_any_cost`**, which checks and executes the fire orders in formats of "**Fire** X from **GOVERNMENT AT ANY COST**" according to the description above. After executing the function and fire the person with the title, **`title`**, the GOVERNMENT stack, **gov**, will reflect the change.

Note:

- You can assume that there exists a person with the title, **`title`**, in the stack, **gov**, and the length of **`title`** is always less than **MAX_STR_LENGTH**.
- You cannot define any additional object other than a stack or a queue only.

- You can use the **strcmp()** library function:

  int strcmp(const char* str1, const char* str2);

  which returns 0 if the contents of both CStrings (**str1** & **str2**) are equal, otherwise, returns a non-zero value.

```
#include "stack-queue.h"                                    /* fire.cpp */

void fire_from_stack_at_any_cost(stack& gov, const char* title)
{
   // Add your code here









}
```

/* Rough work — Do NOT detach this page */

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

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