COMP 2011 Midterm Exam - Fall 2017 - HKUST

Date: October 21, 2017 (Saturday)
Time Allowed: 2 hours, 2–4pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **9** 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 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 also **cannot** use any library functions not mentioned in the questions.

Student Name	SOLUTION
Student ID	
Email Address	
Lecture & Lab Section	

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 4
3	/ 7
4	/ 6
5	/ 6
6	/ 13
7	/ 12
8	/ 14
9	/ 28
Total	/ 100

Problem 1 [10 points] Switch

Vowels are represented by the letters, 'a', 'e', 'i', 'o', 'u'. Consonants are represented by the remaining 21 letters. In particular, two consonant letters, 'w' and 'y', are also regarded as semi-vowels. For example, the word "happy" has 1 vowel, 4 consonants and 1 semi-vowel. The word "birthday" has 2 vowels, 6 consonants and 1 semi-vowel.

Complete the following program that counts the number of letters representing vowels, semi-vowels and consonants in an English word. You may assume that all the characters in the word are lower case alphabets (i.e., 'a' to 'z'). Note that you cannot use any if-statement in your code.

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

int main()
{
   int vowel = 0, semivowel = 0, consonant = 0;
   char word[20];
   cout << "Enter a word: ";
   cin >> word;
   //strlen() returns the length of the C string
   for (int i=0; i<strlen(word); i++) {
      switch (                 ) {









      }
   }
   cout << "num of vowels:" << vowel << endl;
        << "num of semi-vowels:" << semivowel << endl;
        << "num of consonants:" << consonant << endl;
   return 0;
}
```

Solution:

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

int main()
{
   int vowel = 0, semivowel = 0, consonant = 0;
   char word[20];

   cout << "Enter a word: ";
   cin >> word;
   for (int i=0; i<strlen(word); i++)
   {
      switch (word[i])
      {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            vowel++; break;
         case 'w':
         case 'y':
            semivowel++;
         default:
            consonant++;
      }
   }
   cout << "num of vowels:" << vowel << endl
        << "num of semi-vowels:" << semivowel << endl
        << "num of consonants:" << consonant << endl;
   return 0;
}
```

Problem 2 [4 points] enum

What is the output when the following code is executed?

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

int main()
{
   enum Fruits { MELON = 100, APPLE, ORANGE };
   enum Vegatables { CARROT, TOMATO, CUCUMBER = 20, EGGPLANT };

   Fruits favourite = APPLE;
   Vegatables leastFavourite = TOMATO;
   cout << favourite << "#" << leastFavourite << endl;

   return 0;
}
```

Answer: 101#1

Problem 3 [7 points] Operator

C++ has the special pre- and post-increment operator ++ for integers. Let's implement these 2 operators by 2 separate functions `pre_increment` and `post_increment`, respectively so that the following program

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

/* The definition of function pre_increment will be put here */

/* The definition of function post_increment will be put here */

int main()
{
    int x = 5, y = 10;

    cout << "result of pre_increment: " << pre_increment(x) << endl;
    cout << "after pre_increment: x = " << x << endl;

    cout << "result of post_increment: " << post_increment(y) << endl;
    cout << "after post_increment: y = " << y << endl;

    return 0;
}
```

will give the output below:

```
result of pre_increment: 6
after pre_increment: x = 6
result of post_increment: 10
after post_increment: y = 11
```

Remarks:

i. You are <u>not</u> allowed to use the C++'s built-in (pre- and post-increment) ++ operators in your answer.

ii. Both functions will take an integer variable as their only argument.

iii. You have to decide the exact function header of the 2 functions yourselves.

Answer:

(a) [3 points]

```
/* The definition of function pre_increment */
```

(b) [4 points]

```
/* The definition of function post_increment */
```

Solution:

```
/* (a) 3 points */
int pre_increment(int& a)
{
    return (a += 1);
}
```

```
/* (b) 4 points */
int post_increment(int& a)
{
    int original_a = a;
    a += 1;
    return original_a;
}
```

Problem 4 [6 points] Function parameter passing

Fill in the blanks with appropriate parameter-passing mechanism, so that the output of the following program is `8 6 6`.

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

int func( _____ a, _____ b)

{
    int temp, sum;
    sum = a + b;
    temp = a;
    a = b;
    b = temp;
    return sum;
}

int main()
{
    int a = 3, b = 5;
    cout << func(b, a) << " ";
    cout << func(a, b) << " ";
    cout << func(b, a) << endl;
    return 0;
}
```

Solution: int&a, int b

Problem 5 [6 points] Functions

In each of the following programs, there is an error. Please identify the error and explain how to correct it.

(a) [3 points]

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

void print_user_sum(double x, int y, char user){
   cout << user << " " << (x + y) << endl;
}

void print_user_sum(int x, double y, char user){
   cout << user << " " << (x + y) << endl;
}

int main(){
   print_user_sum(0.5, -0.5, 'A');
   return 0;
}
```

Answer:

(b) [3 points]

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

void mystery(double x, int y = 0, char z){
   cout << x << y << z << endl;
}

void mystery(double x, char y){
   cout << x << y << endl;
}

int main(){
   mystery(0.5, '*');
   return 0;
}
```

Answer:

Solution:

(a) [3 points]

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

/*
 * Possible correction 1: remove either of the following
 * two functions.
 */
void print_user_sum(double x, int y, char user)
{
   cout << user << " " << (x + y) << endl;
}

void print_user_sum(int x, double y, char user)
{
   cout << user << " " << (x + y) << endl;
}

/*
 * Possible correction 2: add the following function:
 */
void print_user_sum(double x, double y, char user)
{
   cout << user << " " << (x + y) << endl;
}

int main()
{
   /* Error: Compilation error as neither 1st or 2nd version
    * is preferred over the other for the following
    * function call.
    */
   print_user_sum(0.5, -0.5, 'A');
   return 0;
}
```

(b) [3 points]

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

/* Error: default arguements should be specified at
 * the end of the function
 */
/* Possible correction 1: modify the last argument
 * to a default arguement
 */
```

```
void mystery(double x, int y = 0, char z = '\0')
/* Possible correction 2: remove the default arguement
void mystery(double x, int y, char z)
 */
{
   cout << x << y << z << endl;
}
/* Possible correction 3: remove the above function
 */

void mystery(double x, char y)
{
   cout << x << y << endl;
}

int main()
{
   mystery(0.5, '*');
   return 0;
}
```

Problem 6 [13 points] Loops

Implement a function `hat(int height)` which prints a hat pattern of a given height. Below are example hat patterns with height = 1, 3, and 6, respectively. *Note*: the row numbers below are not part of the program output; it is for your reference only.

Output of `hat(1)`

```
*                                                    row 1
```

Output of `hat(3)`

```
##*##                                                row 1
#*#*#                                                row 2
*###*                                                row 3
```

Output of `hat(6)`

```
#####*#####                                          row 1
####*#*####                                          row 2
###*###*###                                          row 3
##*#####*##                                          row 4
#*#######*#                                          row 5
*#########*                                          row 6
```

Answer:

```
// You may assume height > 0 for simplicity
void hat(int height)
{









}
```

Solution:

```
void hat(int height){
    int row, pound, star;
    for(row=1; row<=height; row++){
        for(pound=1; pound<=height-row; pound++)
            cout << "#";
        for(star=1; star<=2*row-1; star++)
            if(star == 1 || star == 2*row-1)
                cout << "*";
            else
                cout << "#";
        for(pound=1; pound<=height-row; pound++)
            cout << "#";
        cout << endl ;
    }
}
```

Problem 7 [12 points] Array and Loops

What is the output of the following program?

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

void mysterious(int a[], int& n)
{
    int k = 0;
    for (int i = 1; i < n; i++)
    {
        int j = 0;
        while (j <= k && a[i] != a[j])
            j++;

        if (j > k)
        {
            k++;
            a[k] = a[i];
        }
    }
    n = k + 1;
}

int main()
{
    int arr[] = {58, 26, 91, 26, 70, 70, 91, 58, 58, 58, 66};
    int size = sizeof(arr)/sizeof(int);

    mysterious(arr, size);

    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << "size = " << size;
    cout << endl;
    return 0;
}
```

Answer: _____

Solution:

58 26 91 70 66 size = 5

```
void eliminate_duplicates(int a[], int& n)
{
    // Keeps track of the end of the subarray of distinct integers
    int last_unique = 0;

    // Start i at the second cell
    for (int i = 1; i < n; i++)
    {
        // Determine whether a[i] is already present among the
        // integers in cells a[0],...,a[last_unique].
        int j = 0;
        while (j <= last_unique && a[i] != a[j])
            j++;

        if (j > last_unique)// then a[i] is not present earlier,
            a[++last_unique] = a[i]; //put a[i] into the list
    }

    n = last_unique + 1;
}
```

Problem 8 [14 points] Array and Recursion

This question is about finding the maximum value among values stored in a multi-dimensional array using recursion.

(a) Write the recursive function `array_max` to return the maximum value stored in the first n cells of a 1-dimensional integer array `a` of size N, where $n \leq N$.

(b) Write the recursive function `matrix_max` to return the maximum value stored in the sub-matrix of a 2-dimensional integer array `a` with M rows and N columns as shown in Fig. 1. The sub-matrix's dimensions are $r \times c$ where $r \leq M$ and $c \leq N$.

Note: You may make use of the function `array_max` from part (a) to solve part (b) (even if you cannot answer part (a)).

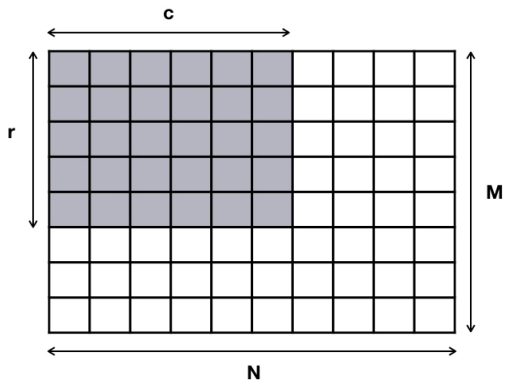

Figure 1: Illustration of part (b).

Your implementations should support the following driver code:

```
const int M = 10, N = 8;

int x[N] = { /* assume proper initialization */ };
int y[M][N] = { /* assume proper initialization */ };

cout << array_max(x, 5) << endl;
cout << matrix_max(y, 9, 7) << endl;
```

Answer:

(a) [6 points] `/* Recursive definition of function array_max */`

(b) [8 points] `/* Recursive definition of function matrix_max */`

Solution:

```
int array_max(int a[], int N)
{
    if (N == 1)
        return a[0];

    int rem_max = array_max(a, N-1);

    return (a[N-1] > rem_max) ? a[N-1] : rem_max;
}
```

```
int matrix_max(int a[][8], int R, int C)
{
    int last_row_max = array_max(a[R-1], C);

    if (R == 1)
        return last_row_max;

    int rem_max = matrix_max(a, R-1, C);

    return (last_row_max > rem_max) ? last_row_max : rem_max;
}
```

Problem 9 [28 points] 2D Character Array

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

const char SPACE = ' ';
const char PERIOD = '.';
const int LINE_LEN = 256;
const int WORD_LEN = 15;
const int MAX_NUM_WORDS = 14;
const int VOCAB_SIZE = 10;

/* Assume that the definition of function same_word is here */
/* Assume that the definition of function get_words_from_sentence is here */
/* Assume that the definition of function spell_check is here */
/* Assume that the definition of function print_found_words is here */

int main()
{
    const char dictionary[VOCAB_SIZE][WORD_LEN+1] = {
        "and", "boy", "cats", "dogs", "evening",
        "fun", "google", "honey", "let", "play" };

    char words[MAX_NUM_WORDS][WORD_LEN+1] = { };
    char sentence[LINE_LEN] = { };
    int found_words_index[MAX_NUM_WORDS];

    cin.getline(sentence, LINE_LEN, '\n');
    int num_words = get_words_from_sentence(sentence, words);
    int num_words_found =
        spell_check(dictionary, VOCAB_SIZE, words, num_words, found_words_index);

    cout << "\nNumber of words in the sentence = " << num_words << endl;
    for (int k = 0; k < num_words; ++k)
        cout << "word " << k << " : " << words[k] << endl;

    cout << "\nThe following words are found in the dictionary:" << endl;
    print_found_words(words, found_words_index, num_words_found);
    return 0;
}
```

In the above incomplete program, a 2-dimensional const char array

```
    const char dictionary[VOCAB_SIZE][WORD_LEN+1];
```

is used as a dictionary to store a list of `VOCAB_SIZE` words; each word has a maximum length of `WORD_LEN`. A sentence is read in by calling `cin.getline` once. To make things simple, it

is assumed that the user is very cooperative:

- only English words are entered in small letters;
- words are separated by exactly one space, and the last word always ends with a period and then the newline character;
- there are no leading spaces in the input sentence;
- an empty sentence is represented by a single period in the input;
- the number of words in a sentence will not be more than `MAX_NUM_WORDS`.

Afterwards, words are extracted from the input sentence into another 2-dimensional char array, `words`, which are output to the screen. Those words are then spell-checked using the given `dictionary`, and the program will save the indices of the correctly spelled words in an int array `found_words_index` and print them out. For example, given the following input sentence:

"let us play with my cute dogs and your lazy dogs this evening."

the program will output:

```
Number of words in the sentence = 13
word 0 : let
word 1 : us
word 2 : play
word 3 : with
word 4 : my
word 5 : cute
word 6 : dogs
word 7 : and
word 8 : your
word 9 : lazy
word 10 : dogs
word 11 : this
word 12 : evening

The following words are found in the dictionary:
word 0 : let
word 2 : play
word 6 : dogs
word 7 : and
word 10 : dogs
word 12 : evening
```

Implement the 3 functions:

- `get_words_from_sentence`
- `spell_check`
- `print_found_words`

as well as a helper function called `same_word` which will be used by the function `spell_check` so that the program runs with the expected output above.

Note:

- Whenever a variable/quantity is better to be specified as `const`, you are required to do so, otherwise marks will be deducted.
- You are not allowed to write any additional helper function(s) other than the required function `same_word`.
- You are not allowed to use any C/C++ string function(s) from the Standard C++ library.

(a) [4 points] Implement the function `same_word` which checks if two given C strings `a` and `b` are the same; it returns true if so, otherwise false. Its function header is given to you.

(b) [10 points] Implement the function `get_words_from_sentence` which extracts words from the given 1D char array, `sentence`, into a 2D char array, `words`. It should return the number of extracted words.

(c) [10 points] Implement the function `spell_check` that checks which of the extracted words from part (b) can be found in the dictionary. If found, their indices are saved into an int array, `found_words_index`, and it returns the number of words found.

(d) [4 points] Implement the function `print_found_words` that prints out the words in the sentence that can be found in the dictionary according to the above output format.

Answer:

(a) [4 points] `/* The definition of function same_word */`

```
bool same_word(const char a[], const char b[])
```

(b) [10 points] `/* The definition of function get_words_from_sentence */`

(c) [10 points] `/* The definition of function spell_check */`

(d) [4 points] `/* The definition of function print_found_words */`

Solution:

```
/* (a) 4 points */
bool same_word(const char a[], const char b[])
{
    int k = 0;

    for (; a[k] && b[k]; ++k)
        if (a[k] != b[k])
            return false;

    // true if both are '\0' as at least one of them must be
    return (a[k] == b[k]);
}
```

```
/* (b) 10 points */
int get_words_from_sentence(const char sentence[LINE_LEN],
                            char words[][WORD_LEN+1])
{
    if (sentence[0] == PERIOD)
        return 0;

    int num_words = 0;


    for (int j = 0; sentence[j] != '\0'; ++j, ++num_words)
    {
        for (int k = 0; sentence[j] != SPACE && sentence[j] != PERIOD; ++k, ++j)
            words[num_words][k] = sentence[j];
    }

    return num_words;
}
```

```
/* (c) 10 points */
int spell_check(const char dictionary[][WORD_LEN+1],
                int dictionary_size,
```

```
                const char words[][WORD_LEN+1],
                int num_words,
                int found_words_index[])
{
    int num_words_found = 0;

    for (int n = 0; n < num_words; ++n)
    {
        for (int k = 0; k < dictionary_size; ++k)
            if (same_word(words[n], dictionary[k]))
            {
                found_words_index[num_words_found++] = n;
                break;
            }
    }

    return num_words_found;
}
```

```
/* (d) 4 points */
void print_found_words(const char words[][WORD_LEN+1],
                       int found_words_index[],
                       int num_words_found)
{
    for (int k = 0; k < num_words_found; ++k)
        cout << "word " << found_words_index[k] << " : "
             << words[found_words_index[k]] << endl;
}
```

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