COMP 2011 Final - Spring 2020 – HKUST

- Date: Jun 2, 2020
- Time Allowed: 3 hours, 6:30 pm - 9:30 pm

Student Name	SOLUTION
Student ID	
ITSC email	
Seat Number	

For T.A. Use Only

Problem	**Score / Max. score**
1 string	/8
2 function overloading	/8
3 class	/19
4 control flow 2D matrix	/7
5 stack with linked list	/24
6 syntax error	/10
7 recursion and binary tree	/24
Total	**/100**

Part 1

Problem 1 String [8 points]

```
const int MAX_CHARS = 256;
bool mystery(const char str1[], const char str2[])
{
    int str1_cnt[MAX_CHARS] = {}, str2_cnt[MAX_CHARS] = {};
    int cnt;

    if (strlen(str1) != strlen(str2))
        return false;

    for (cnt = 0; str1[cnt] != '\0'; cnt++)
        str1_cnt[str1[cnt]]++;

    for (cnt = 0; str2[cnt] != '\0'; cnt++)
        str2_cnt[str2[cnt]]++;

    for (cnt = 0; cnt < MAX_CHARS; cnt++)
        if (str1_cnt[cnt] != str2_cnt[cnt])
            return false;

    return true;
}
```

Given the above function definition, what is the output of the following code?

```
cout << boolalpha << mystery("amy", "may") << endl;
```

Answer: true

```
cout << boolalpha << mystery("c++", "abc") << endl;
```

Answer: false

```
cout << boolalpha << mystery("spears", "pears") << endl;
```

Answer: false

```
cout << boolalpha << mystery("listen", "silent") << endl;
```

Answer: true

// Grading Scheme: 2 points each,

// Partial credit of 1 point if answer gets the correct truth value (TRUE, FALSE, True, False, 1, 0)

Problem 2 Function Overloading and Default Arguments [8 points]

For each of the following programs, state whether the program can be compiled successfully without any error. If yes, write the output; if no, explain the error(s). Clearly label each part of your answer with (a), (b), (c), (d).

(a)

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

int f(int a, double b) { return (a-b); }
double f(int a, double b) { return (a+b); }

int main() {
   int result = f(10, 0.5);
   cout << result << endl;
   return 0;
}
```

(b)

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

void f(int a) { a--; cout << a << endl; }
void f(int& a) { a++; cout << a << endl;}

int main()
{
   f(10);
   return 0;
}
```

(c)

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

double f(int a, double b) { return (a*b); }
double f(int a, double b, double c = 0.5) { return (a*b*c); }

int main() {
   cout << f(10, 0.5) << endl;
   return 0;
}
```

(d)

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

double f(const int* x, int& y) {
     (*x)*=(++y);
}

int main() {
   cout << f(10, 5) << endl;
   return 0;
}
```

Program	**Can the program compile successfully and run? YES/NO**	**If Yes, what is the output? If No, what is/are the error(s)?**
(a)	No (1 point)	The two functions have the same signature (1 point)
(b)	Yes (1 point)	9 (1 point)
(c)	No (1 point)	Both functions are matched with the function call. The compiler cannot decide which is better. (1 point)
(d)	No (0.5 point)	Error 1: x is a pointer to constant integer. The content of the object pointed to by x cannot be changed through the pointer. The following is wrong: (*x) = … (0.5 point) Error 2: we cannot pass a literal constant by pointer (parameter x). (0.5 point) Error 3: we cannot pass a literal constant by reference (parameter y). (0.5 point)

Problem 3 Function Overloading and Default Arguments [19 points]

Here is an incomplete `food` class definition in the header file `food.h`:

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

class food{
public:
   food();
   food(const char* name, int portion_weight, int calories);
   ~food();
   // accessor
   bool same(const food &other) const;

   /* one more member function prototype to be added here */

   // mutator
   /* one more member function prototype to be added here */

private:
   char *name;
   int portion_weight;
   int calories;
};
```

And an example driver program `main.cpp`:

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

using namespace std;


int main(){
   food fruit1, fruit2("orange", 100, 49);

   fruit1.print();
   fruit1.set("green apple", 70, 50);
   fruit1.print();
   fruit1.set("apple", 100, 52);
   fruit1.print();
   fruit2.print(2);

   food temp("apple", 88, 68);
   cout << fruit1.same(fruit2) << endl;
   cout << fruit1.same(temp) << endl;

   return 0;
}
```

And its output is given below:

```
Name: green apple
Total portion weight: 70
Total calories: 50
Name: apple
Total portion weight: 100
```

```
Total calories: 52
Name: orange
Total portion weight: 200
Total calories: 98
0
1
```

Study the header file, the driver program and its output carefully and work on the following questions.

(a) There is ONE mutator member function prototype and also ONE accessor member function prototype missing in the `food` class definition. Write the two member function prototypes below.

Note: Whenever a member function is an accessor, or a function argument should not be modified, you are required to specify it with `const`, otherwise marks will be deducted.

Answer:

```
void set(const char* _name, int _weight, int _calories);
//Or
// void set(const char _name[], int _weight, int _calories);
// 2 points in total,
// -0.5 point for each mistake on the function name, each parameter

void print(int num_poritions = 1) const;
// 2 points in total,
// -0.5 point for each mistake the function name, paramter,
// default arguement, const keyword
```

(b) Implement all the member functions (including the ones in part (a)) in `food.cpp`.

- You must write the complete member function definitions.
- You may use `strcmp()`, `strcpy()` and `strlen()` in the `CString` library functions if needed.

```
/* strcpy() copies the string pointed to by s2 to the string pointed to
* by s1, returning pointer s1.
* The string pointed to by s1 should have enough memory to copy the
* characters from the string pointed to by s2.
*/
char* strcpy(char* s1, const char* s2);

/* The strcmp() function compares the two given C strings s1 and s2, and
* returns zero if the two strings are the same. If the two strings are
* different, then it returns an integer less/greater than zero if for the
* first characters that they differ, the character in s1 is
* smaller/larger than that in s2 in terms of their ASCII codes.
```

```
*/
int strcmp(const char *s1, const char *s2);

/* strlen() calculates the length of the C string s, excluding the
* terminating null character.
*/
int strlen(const char* s);
```

- The dynamic array, `name`, should have just enough memory (without excess) to store the `CString`.
- It should generate EXACTLY the same output with the given driver program above.
- You should ensure that no memory leak will occur in your implementation.

```
#include "food.h"
/* Default Constructor */
food::food() // 2 points in total
{
  name = nullptr; // 1 point
  portion_weight = 0;  // 0.5 point
  calories = 0; // 0.5 point
}

/* Other Constructor */
food::food(const char* name, int portion_weight, int calories)
// 3 points total
{
   /* Solution 1 */
   this->name = nullptr;
   // 1 point for initializing this->name = nullptr;
    // this pointer is needed if the parameter names are name,
    // portion_weight & calories.
    // it is ok to change the parameter names in the function header

   set(name, portion_weight, calories);
   // 2 points


   /* Solution 2 */
   this->name = new char(strlen(name)+1); // 1.5 point
   strcpy(this->name, name);  // 0.5 point
   this->portion_weight = portion_weight; // 0.5
   this->calories = calories; // 0.5
  // this pointer is needed if the parameter names are name,
  // portion_weight & calories.
  // it is ok to change the parameter names in the function header
}

/* Destructor */
food::~food() // 2 points in total
{
   if (name!=nullptr) // Good practice: 0 point
      delete [] name;  // 2 points
}

/* Set member function: Solution 1 */
```

```
void food::set(const char* _name, int _weight, int _calories)
// 3.5 points in total
{
   if (_name)  // 0.5 point
   // Or if (_name != nullptr)
   {
      if (name)  // Good practice, 0 point
         delete [] name; // 0.5 point
      name = new char(strlen(_name)+1); // 1 point
      strcpy(name, _name);  // 0.5
      portion_weight = _weight; // 0.5
      calories = _calories; // 0.5
   }
}
/* Set member function: Solution 2 */
void food::set(const char _name[], int _weight, int _calories)
// 3.5 points in total
{
   if (strlen(_name) != 0)  // 0.5 point
   // Or if (_name[0] != '\0')
   {
      if (name)  // Good practice, 0 point
         delete [] name; // 0.5 point
      name = new char(strlen(_name)+1); // 1 point
      strcpy(name, _name);  // 0.5
      portion_weight = _weight; // 0.5
      calories = _calories; // 0.5
   }
}

/* Same member function */
bool food::same(const food& other) const // 2 points in total
{
   if (name && other.name) // 0.5 point
     return !strcmp(name, other.name); // 1 point
   return false; // 0.5 point
}

/* Print member function */
void food::print(int num_portions) const  // 2.5 points in total
{
   if (name!=nullptr) // 1 point
   {
```

```
      cout << "Name: " << name << endl; // 0.5 point
      cout << "Total portion weight: " << portion_weight*num_portions <<
endl; // 0.5 point
      cout << "Total calories: " << calories*num_portions << endl; // 0.5
points
   }
}

// -2 points for missing food:: for the whole part
// if food:: is missing for one or 2 functions, consider it as
// a syntax error
// -0.5 pt per syntax error (repeated error count as one),
// max -4 points in total on syntax errors
```

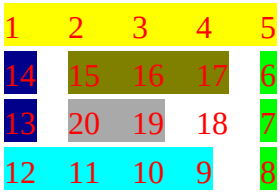
Part 2

Problem 4 Control Flow in 2D Array *[7 points]*

What is the output of the following program?

```
#include <iostream>
using namespace std;
const int M = 4;
const int N = 5;

void mysterious(const int arr[], int mat[][N]){
    int a = 0, b = M - 1;
    int c = 0, d = N - 1;
    int index = 0;

    while (1){
        if (c > d) break;
        for (int i = c; i <= d; i++)
            mat[a][i] = arr[index++];
        a++;

        if (a > b) break;
        for (int i = a; i <= b; i++)
            mat[i][d] = arr[index++];
        d--;

        if (c > d) break;
        for (int i = d; i >= c; i--)
            mat[b][i] = arr[index++];
        b--;

        if (a > b) break;
        for (int i = b; i >= a; i--)
            mat[i][c] = arr[index++];
        c++;
    }
}
int main(){
    int arr[N*N] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                     16, 17, 18, 19, 20, 21, 22, 23, 24, 25 };
    int mat[M][N];
    mysterious(arr, mat);
    for (int i = 0; i < M; i++){
        for (int j = 0; j < N; j++)
            cout << mat[i][j] << '\t';
        cout << '\n';
    }
    return 0;
}
```

Answer:



1 point for each color

Problem 5 Stack with Linked List Implementation *[24 points]*

A stack is an abstract data type (ADT) with last-in-first-out (LIFO) policy. In this question we implement a stack with essentially unbounded capacity using a linked list. Each node in the linked list is a structure as described below:

```
struct stackNode {
    int data;
    stackNode* next;
};
```

The head node of the linked list, which corresponds to the top of the stack, is pointed by the variable `top` of type `StackNode*`. For an empty stack, `top` has a `nullptr` value.

Recall that the push and pop operations of a stack only happen at the top. The stack in this question supports the following operations:

```
void push(stackNode*& top, int value); // push an item into the stack
int pop(stackNode*& top); // remove the top item from the stack and
                          // return its value (-1 if the stack is empty)
int size(const stackNode* top); // return the number of items in the
                                // stack (0 if the stack is empty)
void reverse(stackNode*& top); // reverse the order of the stack
```

For simplicity, we assume that all values in the stack are positive integers.

(a) [8 points] Implement the `int pop(stackNode*& top)` function that removes the top item from the stack and returns its value. It returns `-1` if the stack is empty.
Write the complete function definition.

Answer:

```
int pop(stackNode*& top){
    if (top == nullptr) // test empty stack, 1 POINT
        return -1; // return -1, 1 POINT
    else{
        stackNode* topNode = top;
        int topValue = top->data;
        top = top->next; // update top pointer, 2 POINTs
        delete topNode; // delete top item, 2 POINTs
```

```
        topNode = nullptr; // good practice, 0 POINT
        return topValue; // return top value, 2 POINTs
    }
}
Grading scheme:
- memory leak, i.e., lost the stack: -3 points
```

(b) [8 points] What is the output of the following program in which the `main()` function uses the `push`, `pop`, and `size` operations described above?

```
int main(){
    stackNode* stack = nullptr;
    const int MAX = 4;
    for (int i = 0; i < 10; i++){
        if (i % 4 == 0){
            if (size(stack)!= 0)
                cout << "pop: " << pop(stack) << endl;
        }
        else{
            if (size(stack) < MAX){
                push(stack, i);
                cout << "push: " << i << endl;
            }
        }
    }
    return 0;
}
```

Answer:

```
push: 1 // longest sub-string from the beginning, 1 POINT each line
push: 2
push: 3
pop: 3
push: 5
push: 6
pop: 6
push: 9
```

(c) [8 points] Implement the `reverse(stackNode*& top)` function that reverses the order of the stack.

Hints/assumptions:

- You may assume that the `push()`, `pop()` and `size()` functions have been implemented and can be used if necessary.
- When additional memory space is used, you should make sure that there is no memory leak.

Write the complete function definition.

Answer:

```
void reverse(stackNode*& top)
{
    stackNode* reverseStack = nullptr;
    while (size(top) > 0) // iterate over all items, 2 POINTS
        push(reverseStack, pop(top)); // reverse order, 4 POINTS
    top = reverseStack; // reset stack top, 2 POINTS
}
```

Grading scheme:

- any memory leak: -2 points

Part 3

Problem 6 Syntax Errors *[10 points]*

Consider each of these independent lines of code as being the only line in your `main()` function:

```
1   while();
2   while(0) {cout << "hello"}
3   for(int i = 0; ; i++) {cout << i;}
4   int a = 1; a = *(&(*(&a)));
5   int *b = nullptr; b = *(&(*(&b)));
6   int c = 1; c = &(*(&(*c)));
7   char d = 'd'; ++(++d);
8   char a, b = '3', c[] = "hello";
9   int e[] = {1, 2, 3};
10  int f[2][2] = {1, 3, 1, 3};
```

List the numbers of all the lines that have syntax errors.
(For example, if lines 4, 6, 8 are the ones containing syntax errors, your answer should be: **`4 6 8`**)

Answer:

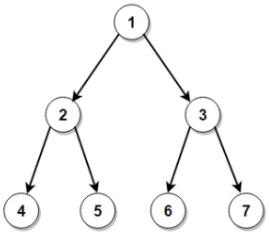
`1 2 6`

1 point for each of the 10 lines being correctly included or omitted from the answer

Problem 7 Recursion and Binary Tree *[24 points]*

NOTE: For Problem 7, all the functions implemented must be recursive and cannot have any helper functions. Iterative solutions or solutions with helper function(s) will not receive credit.

A Binary Tree is an abstract data type (ADT) with tree structure of `nodes` in which each node has at most two children, referred to as the `left` child and the `right` child. The topmost node in the tree is the `root`.



Below is the structure definition of a binary tree node:

```
struct Node{
    int data;
    Node *left, *right;
};
```

In this question, you are asked to write a few recursive functions to perform specific operations on the binary tree.

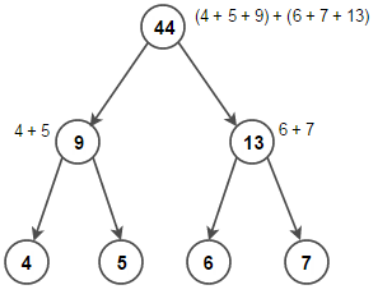
(a) [8 points] Write a recursive function `isIdentical` which checks whether two trees are identical. The function header is given below. Complete the function body.

```
// Recursive function that takes the root nodes of the two trees
// and returns a boolean value specifying whether they are
//  identical or not

bool isIdentical(const Node* x, const Node* y)
{
```

Answer:

```
    // if both trees are empty, return true [2 pts]
    if (x == nullptr && y == nullptr)
        return true;

    // if both trees are non-empty and the value of their root
    // node matches, recur on left and right subtrees [1, 1, 2, 2 pts]
    return (x != nullptr && y != nullptr) && (x->data == y->data) &&
           isIdentical(x->left, y->left) &&
           isIdentical(x->right, y->right);
```

(b) [8 points] In a "sum tree", the value at each node is equal to the sum of all the values in its left and right subtrees. Leaf nodes, which have neither a left nor right child, can have any value. Below is an example of a sum tree:



Write a recursive function `isSumTree` which checks whether a binary tree is a sum tree. If it is a sum tree, the function should return the sum of all of its data values, otherwise it should return -1. The function header is given below. Complete the function body.

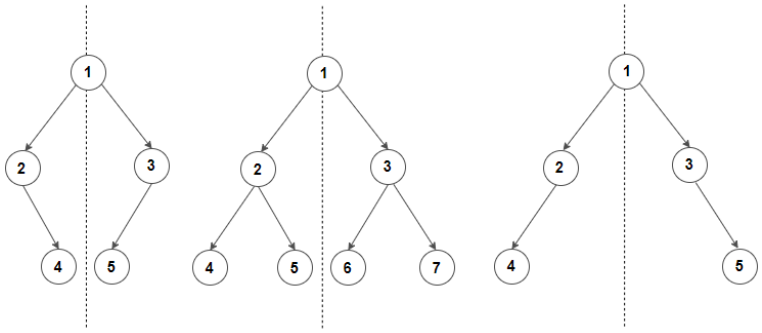
Note: For part (b), you may assume that all nodes have non-negative data.

```
//Binary tree node structure (same as in part a)
struct Node{
    int data;
    Node *left, *right;
};

// Recursive function to check if given binary tree is a sum tree
// If it is a sum tree, sum of all of its data values is returned
// otherwise, -1 is returned

int isSumTree(const Node *root)
{
```

Answer:

```
    // base case: empty tree [1 pt]
    if (root == nullptr)
        return 0;
    // base case: leaf node [1, 1 pt]
    if (root->left == nullptr && root->right == nullptr)
        return root->data;
    // if root's value is equal to sum of all elements present in its
    // left and right subtree [2, 2 pts]
    if (root->data == isSumTree(root->left) + isSumTree(root->right))
        return 2*root->data;
    return -1; [1 pt]
```

(c) [8 points] A binary tree has a symmetric structure if the left subtree is a "mirror" of the right subtree. Below are examples of binary trees with symmetric structure:




```
//Binary tree node structure (same as in previous parts)
struct Node{
    int data;
    Node *left, *right;
};

// function to check if a given non-empty binary tree is symmetric
// it returns true if the binary tree is symmetric, false otherwise
bool isSymmetric(const Node *root)
{
    // return true if left subtree and right subtree are
    // mirror images of each other
    return isMirror(root->left, root->right);
}
```

You are given the `isSymmetric` function above, which checks if a given non-empty binary tree is symmetric or not. `isSymmetric` calls `isMirror`. Complete the recursive function `isMirror` whose header is given below.

```
// Function to check if subtree rooted at x and y are mirror images
// of each other
// it returns true if yes, false otherwise
bool isMirror(const Node *x, const Node *y)
{
```

Answer:

```
    // base case: if both tree are empty [2 pt]
    if (x == nullptr && y == nullptr)
        return true;

    // return true if
```

```
// 1. both trees are non-empty and [2 pt]
// 2. left subtree is mirror image of right subtree and [2 pt]
// 3. right subtree is mirror image of left subtree [2 pt]
return (x != nullptr && y != nullptr) &&
       isMirror(x->left, y->right) &&
       isMirror(x->right, y->left);
```

Grading notes:

1. Writing "isMirror(x->left,y->right);" and "isMirror(x->right,y->left);" separately without "&&"" or "return" is worth at most 2 marks.
2. "x->left" and "y->right" should not be used before making sure x/y are NOT nullptr
3. 1 mark will be deducted if you compare the data of the nodes. It is clear from the given description and examples that we only care about the structure but not the data/numbers stored in the nodes