COMP 2011 Final - Spring 2015 - HKUST

Date:	May 26, 2015 (Tue)
Time Allowed:	3 hours, 4:30 pm - 7:30 pm
Instructions:	1. This is a closed-book examination. 2. There are **10** questions on **19** pages (including the cover page and two scratch pages at the end). 3. Write your answers in the space provided in black/blue ink. **NO** pencil please. 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 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 **can not** use any library functions not mentioned in the questions.

Student Name	
Student ID	
Email Address	

Problem	Score
1	/6
2	/8
3	/8
4	/8
5	/8
6	/10
7	/10
8	/12
9	/15
10	/15

Problem 1 [6 points] Structure definition

This question is related to C++ structure. We want to define a new user-defined data type by using struct for a picture. A picture consists of the following members.

- its name represented by a C++ character string of at most 255 characters (excluding the terminating character)
- its width represented by an integer
- its height represented by an integer
- an enum (given below) representing the type of the picture, such as .jpeg, .png

```
enum ImageType
{
        jpeg = 0,
        png = 1,
        bmp = 2,
        tif = 3,
};
```

- a char pointer that points to the actual data of the picture (note that usually char variable is used to efficiently store an integer between 0 - 255)

Design a C++ struct definition for the above structure. You can use your own names for the variables in the definition.

Your answer:

Problem 2 [8 points] Pointer and Scope

Consider the following C++ code

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

int  *p;
void pp(int  a, int  *b)
{
        int  c=4;
        *p=*b+c;
        a=*p-c;
        cout<<"(2)"<<a<<' '<<*b<<' '<<*p<<endl;

}

int main()
{
        int a=1, b=2, c=3;
        p=&b;
        pp(a+c, &b);
        cout<<"(1)"<<a<<' '<<b<<' '<<*p<<endl;
        return 0;
}
```

What is the output of the program?

The output is

Problem 3 [8 points] If-else Statement
Consider the following C++ code

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

int main(){
        int x=3;
        int y=4;
        double z=5;

        if(x/y)
                cout<<"Value of x/y is "<<x/y<<endl;
        else{
                x++;
                cout<<"Value of x/y is "<<x/y<<endl;
        }

        if(x/z)
                cout<<"Value of x/z is "<<x/z<<endl;
        else{
                x++;
                cout<<"Value of x/z is "<<x/z<<endl;
        }

        return 0;
}
```

What is the output of the program?

The output is

Problem 4 [8 points] Parameter Parsing
Consider the following C++ code

```
# include  < iostream>
using namespace std;
void  f1( int I )
{
        I += 10 ;
}

void  f2( int * I )
{
        *I += 10 ;
}

void  f3( int& n )
{
        n += 10;
}

int  main( )
{
        int  I = 0;
        f1( I );
        cout <<"I  is  "<< I << endl;
        f2( &I );
        cout <<"I  is  "<< I << endl;
        f3( I );
        cout <<"I  is  "<< I << endl;
        return 0;
}
```

What is the output of the program?

The output is

Problem 5 [8 points] Stack

A stack is an abstract data type that stores elements in a LIFO (last in first out) fashion. Below is a stack implementation you learned in class. What is the output of the main function?

```
#include <iostream>
#include <cstdlib>

using namespace std;
const int BUFFER_SIZE = 100;
class int_stack
{
private:
        int data[BUFFER_SIZE];
        int top_index;
public:
        int_stack()     // Default Constructor
        {
                top_index = -1;
        }

        bool empty() const// check if the stack is empty
        {
                return (top_index == -1);
        }
        bool full() const// check if the stack is full
        {
                return (top_index == BUFFER_SIZE);
        }

        int size() const// give the number of stored data
        {
                return top_index+1;
        }

        int top() const // retrieve the value of the top item
        {
                if(!empty())
                        return data[top_index];
                cerr << "Warning: Stack is empty; Can't retrieve any
                    data!" << endl;
                exit(-1);
        }

        void push(int x)// add a new item to the top of the stack
        {
                if(!full())
                {
                        data[++top_index] = x;
                }
                else
                {
                        cerr << "Error: Stack is full; can't add ("
                            << x << ")!" << endl;
                        exit(-1);
```

```
                }
        }
        void pop()      // remove the top item from the stack
        {
                if(!empty())
                {
                        --top_index;
                }
                else
                {
                        cerr << "Error: Stack is empty; can't remove
                            any data!" << endl;
                }
        }
};

int main()
{
        int_stack s;
        int a = 1, b = 2;
        int *c;
        c = new int [3];
        for(int i = 0; i < 3; i++)
        {
                c[i] = i;
        }
        s.push(a);
        s.push(b);
        cout << s.top() << endl;
        s.pop();
        cout << s.top() << endl;
        s.push(c[1]);
        s.push(c[2]);
        c[2]++;
        cout << s.top() << endl;
        s.pop();
        cout << s.top() << endl;
        cout << s.size() << endl;

        delete [] c;
        return 0;
}
```

Your answer:

Problem 6 [10 points] Mysterious function on linked list

Consider the following mysterious function on linked list:

```
struct node
{
        int value;      // integer stored in a linked list node
        node* next;     // points to the next node
};


node* mystery(node* head,int x)
{
        node* a=new node;
        node* list1=a;
        node* b=new node;
        node* list2=b;

        for (node *cur=head;cur!=NULL;cur= cur->next){
                if(cur->value < x){
                        list1->next=cur;
                        list1=cur;
                }
                else{
                        list2->next=cur;
                        list2=cur;
                }
        }

        list2->next=NULL;
        list1->next=b->next;
        return a->next;
}
```

Suppose we have one linked list p as follows, where p points to the head of the linked list:

$$7 \to 6 \to 5 \to 4 \to 3 \to 2 \to 1$$

(a) What is the returned linked list q after the following program segment is executed, where q points to the head of the linked list? (You need to print all integers stored in the linked list.)

```
int k=4;
node* q = mystery(p,k);
```

Answer: ____

(b) What task does the function 'mystery' perform?

Answer: ____

```
}
```

Problem 7 [10 points] Rotate array

Given one 2D array, your task is to rotate this array by $180°$. Here is one example for illustration:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

After Rotation ⇒

16	15	14	13
12	11	10	9
8	7	6	5
4	3	2	1

The function input is an array $s[n][n]$, with an equal number of rows and columns. **NO** new array can be defined.

```
void rotate(int s[][n], int num_cols){
//Add Your Answer Code Here









}
```

Problem 8 [12 points] Number Spanning Square

Number Spinning Square is a 2-d array that starts with 1 as the top-left element. The elements increase by 1 in a spinning way, from the outer loop to the inner loop. The following figure gives an example of a number-spinning square of size 6×6.

1	20	19	18	17	16
2	21	32	31	30	15
3	22	33	36	29	14
4	23	34	35	28	13
5	24	25	26	27	12
6	7	8	9	10	11

Figure 1: a number-spinning square with size 6×6

Given the length of the square N, write a program to generate such a number-spinning square. The main program structure is given below, and the output function is also given to you.

```
#include <iostream>
#include <cstring>
#include <iomanip>

using namespace std;

const int MAX_LEN = 20;
int nss[MAX_LEN][MAX_LEN];

void fill_nss(int n)
{
    // (1) complete this method
```

```
}

void output(int n)
{
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            cout << setw(4) << nss[i][j];
        }
        cout << endl;
    }
}

int main()
{
    int n;
    cout << "Please give array size" << endl;
    cin >> n;

    fill_nss(n);
    output(n);

    return 0;
}
```

Problem 9 [15 points] Linked Lists

In mathematics, univariate polynomials (polynomials with only one variable x) can be represented as

$$P_n(x) = a_0x^0 + a_1x^1 + a_2x^2 + ...a_nx^n, \tag{1}$$

with an ascending powers. Therefore the unitary polynomial can be determined by using $(n + 1)$ coefficients, $a_0, a_1, ..., a_n$, if the maximum power is n. But storing all the coefficients can be inefficient when the polynomial is sparse. Suppose we have the following sparse polynomial:

$$P(x) = 1 + 3x^{1000} - 2x^{2000}. \tag{2}$$

The coefficients for $x^i, i = 0, 1, 2...2000$ are $[1, 0, 0, ..., 3, 0, 0, ... - 2]$. If we store all the coefficients, we need to store 2001 integers, which is inefficient while there are only 3 non-zero coefficients for the polynomial. An efficient alternative is to use a linked list, and store only the non-zero coefficients (1st elements) and their corresponding exponents/powers (2nd elements), like

$$HEAD \rightarrow (1, 0) \rightarrow (3, 1000) \rightarrow (-2, 2000) \rightarrow NULL. \tag{3}$$

Suppose we have defined the linked list node structure as follows.

```
struct LNode
{
    float coef; // coefficient
    int  exp; // exponent or power
    struct LNode *next;
}
```

We have created two linked lists A and B representing two univariate polynomials. Pa and Pb are the heads of the two linked lists. qa is the working node of A and qb is the working node of B. Adding two polynomials is similar to insertions on A, by comparing the exponents of the nodes that qa and qb are pointing to, we can derive the following computation rules under three conditions:

(1) $qa->exp < qb->exp$
The node pointed by qa in the linked list A will be in the summation result. As such it is kept. You only need to point qa to the next node.

(2) $qa->exp > qb->exp$
You need to insert the node qb into linked list A, and point qb to the next node in B.

(3) $qa->exp == qb->exp$
We need to add the coefficients of the two terms. If the sum of the coefficients is zero, you need to delete both working nodes in A and B, and point qa and qb to the next nodes; If the sum of coefficients is not zero, you need to modify the coefficient in A, delete the node in B, and point qa and qb to the next nodes.

You need to complete the codes for adding this two polynomials, and store the summation result in linked list A. The function $addpoly$ has two parameters Pa and Pb, representing the heads of the two polynomials. (Note: Pa and Pb do not store any polynomial coefficients or exponents. The first polynomial coefficients are stored in the nodes pointed by $Pa->next$ and $Pb->next$. As a result, in the code given to you, qa and qb point to $Pa->next$ and $Pb->next$ respectively at the beginning.)

```
void addpoly(LNode *Pa, LNode *Pb)
{
        float x;
        // qa is pointing to the current node in polynomial A
        // qb is pointing to the current node in polynomial B
        // qaoprev is pointing to the node one before qa
        LNode *qa,*qb,*qaprev;
        qa=Pa->next;
        qb=Pb->next;
        qaprev=Pa;
        while(qa&&qb){
                if(qa->exp<qb->exp ){
                // the exponent in A is smaller than that in B
                //ADD YOUR CODE HERE











                }

                else if(qa->exp>qb->exp){
                // the exponent in A is larger than that in B
                //ADD YOUR CODE HERE
















                }
```

```
                else
                {
                // the exponent in A equals that in B
                //ADD YOUOR CODE HERE










                }
        }//while
        if(qb)
                qaprev->next=qb;
        delete Pb;
}
```

Problem 10 [15 points] Binary Tree

Here is the definition of binary tree node:

```
/* File: btree.h */
struct btnode
{
        int data;
        btnode* left;
        btnode* right;
};
```

In this question, you need to fulfill three functions: ”SarrToBt”, ”SizeTree” and ”PathSum”. Do **NOT** modify the function name, return type, and formal parameter list of any function.

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

btnode* SarrToBt(int sarr[],int start, int end);
int SizeTree(btnode* root);
bool PathSum(btnode* root, int sum);

int main()
{
        int sarr[]={1,2,3,4,5,6,7}; // Sorted Array
        int start=0;
        int end=sizeof(sarr)/sizeof(sarr[0])-1;

        // (a) Construct a binary tree given a sorted array
        btnode* q=SarrToBt(sarr,start,end);

        // (b) Find the size of a binary tree
        int stree=SizeTree(q);
        cout<<"The size of this tree is "<<stree<<endl;

        // (c) Determine whether there exists a path from root to
            leaf whose sum equals to the given sum
        int sum=9;
        bool psum=PathSum(q,sum);
        cout<<"There exists a path with a sum of "<<sum<<": "<<(psum
            ? "true":"false")<<endl;

        return 0;
}
```

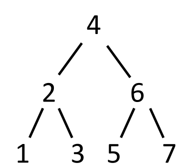
(a) Given a sorted array in ascending order, you need to convert it into a binary tree.

Here are the **requirements** on this constructed binary tree:

- This constructed binary tree has to be height-balanced. For each node in the height-balanced tree, the height difference of its left sub-tree and right sub-tree cannot be larger than 1. Definition of the height of a node in a binary tree: the longest path length from this node to the leaf. A leaf is a node with no left sub-tree and no right-tree. The height of a leaf is zero.

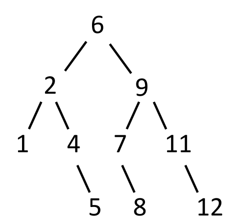
- For each node in this binary tree, all node values in its left sub-tree have to be smaller than those in its right sub-tree.
- When converting a sorted array to a binary tree, the median of this array has to be chosen as the tree root (the array index of the median is given as $\frac{n-1}{2}$ (integer division), where n is the number of elements in this array).

Examples:
Input sorted array: sarr[ ] $= \{1, 2, 3, 4, 5, 6, 7\}$. Constructed height-balanced binary tree:



The corresponding heights are $0, 1, 0, 2, 0, 1, 0$, respectively. The size for the entire tree is 7.

Input sorted array: sarr[ ] $= \{1, 2, 4, 5, 6, 7, 8, 9, 11, 12\}$. Constructed height-balanced binary tree:



The corresponding heights are $0, 2, 1, 0, 3, 1, 0, 2, 1, 0$, respectively. The size for the entire tree is 10.

```
btnode* SarrToBt(int sarr[], int start, int end){
//Add Your Answer Code Here
```

```
}
```

(b) Given a binary tree, calculate its size (number of nodes).

```
int SizeTree(btnode* root)
{
//Add Your Answer Code Here

}
```

(c) Determine whether there exists a path from the root node to the leaf node, such that the sum of nodes' values along this path equals to the given sum value.

```
bool PathSum(btnode* root, int sum)
{
//Add Your Answer Code Here

}
```

Scratch Paper

Scratch Paper