PRINCE OF SONGKLA UNIVERSITY

FACULTY OF ENGINEERING

Final Examination: ภาคการศึกษาที่ 1 **Academic Year:** 2548

Date: 5 ตุลาคม พ.ศ. 2548 **Time:** 13.30-16.30 น.

Subject Number: 240-204 **Room:** R200, R300

Subject Title: Data Structures and Computer Programming Techniques

---

ทุจริตในการสอบ มีโทษขั้นต่ำ คือ ปรับตกในรายวิชาที่ทุจริต และพักการเรียน 1 ภาคการศึกษา

อ่านรายละเอียดของข้อสอบ และคำแนะนำให้เข้าใจก่อนเริ่มทำข้อสอบ

รายละเอียดของข้อสอบ:

เวลา 3 ชั่วโมง (180 คะแนน: 180 นาที)
เอกสารมีทั้งหมด 8 หน้า (ไม่รวมหน้านี้) ประกอบด้วยข้อสอบ 4 หน้าและโค้ดประกอบ 4 หน้า
ข้อสอบมีทั้งหมด 4 ข้อ
คำตอบของข้อที่ **1-2**: ให้ทำในสมุดเล่มสีชมพูเท่านั้น
คำตอบของข้อที่ **3-4**: ให้ทำในสมุดเล่มสีเหลืองเท่านั้น
(หากตอบคำถามผิดเล่มจะถูกหักคะแนนในข้อนั้น 50%)

สิ่งที่สามารถนำเข้าห้องสอบได้:

อนุญาต: กระดาษขนาด A4 ที่เขียนด้วยตนเอง 1 แผ่น และเครื่องเขียนต่าง ๆ
ไม่อนุญาต: หนังสือ และเครื่องคิดเลข

คำแนะนำ:

- พยายามทำทุกข้อ
- คำตอบทั้งหมดจะต้องเขียนลงในสมุดคำตอบ
- คำตอบส่วนใดอ่านไม่ออก จะถือว่าคำตอบนั้นผิด
- อ่านคำสั่งในแต่ละข้อให้ชัดเจนว่า เขียนโปรแกรมบางส่วน เขียนฟังก์ชัน หรือเขียนทั้งโปรแกรม รวมไปถึงข้อกำหนดเพิ่มเติม และหมายเหตุในข้อนั้น ๆ
- การเขียนโปรแกรมในแต่ละข้อ อาจจะไม่ต้องเขียนตามคำสั่งย่อยทั้งหมด แต่คะแนนจะลดลงตามส่วน และหากในข้อใหญ่หนึ่งข้อ นักศึกษาไม่สามารถทำข้อย่อยข้อแรก ๆ ได้ นักศึกษาสามารถทำข้อย่อยหลัง ๆ โดยให้อ้างอิงเหมือนนักศึกษาทำข้อย่อยข้อแรก ๆ ได้
- การเขียน code จะต้องตั้งชื่อตัวแปรให้เหมาะสม และมี comment ในจุดสำคัญต่าง ๆ โดยให้ทั้งหมดเป็นไปตามหลักการเขียนโปรแกรมที่ดี

ข้อที่ 1 TREE *(30 คะแนน: 30 นาที)*

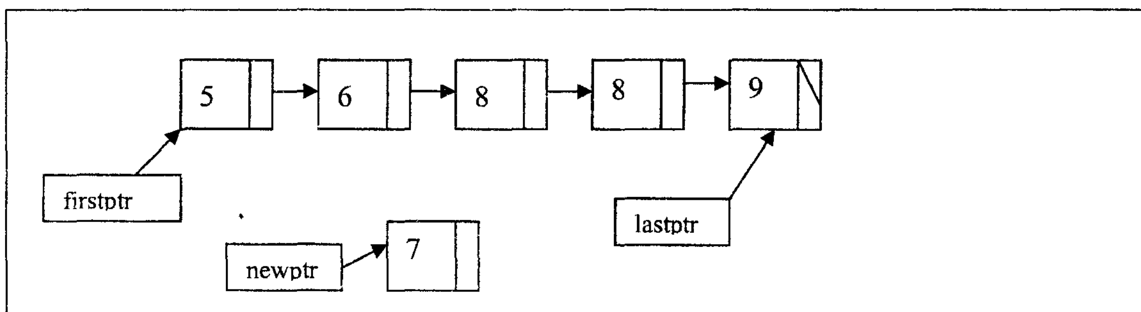
1.1 จากข้อมูลทั้ง 10 จำนวนต่อไปนี้ จงวาดต้นไม้แบบ Binary Search Tree (15 คะแนน)

13, 11, 1, 9, 5, 15, 19, 17, 50, 2

1.2 จากการวาดต้นไม้ในข้อ 1.1 จงเขียนผลการเยี่ยม node ต่าง ๆ แบบ Pre-order, Post-order และ In-order (15 คะแนน)

ข้อที่ 2 ASCENDING LINKED LIST *(30 คะแนน: 30 นาที)*

ในข้อนี้ต้องการให้สร้าง Linked List ที่มีการเรียงข้อมูลจากน้อยไปมาก (ascending order) โดยที่ข้อมูลสามารถซ้ำกันได้ดังรูป



2.1 จง**<u>วาดรูปทีละขั้นตอนของการเลื่อน pointer</u>** เพื่อแทรกโหนดที่มีเลข 7 ลงไปใน List ที่กำหนดให้ ซึ่งในการแทรกจะต้องคำนึงถึงการเรียงของข้อมูลด้วย (สามารถใช้ pointer อื่น ๆ เพิ่มเติมได้)

(15 คะแนน)

2.2 จงเขียนโค้ดของฟังก์ชั่น `insert` ซึ่งจะทำการแทรก**<u>โหนดที่มีเลขใด ๆ</u>** เข้าไปยัง Linked List โดยจะ**<u>ต้องมีการเปรียบค่าเพื่อการแทรกในตำแหน่งที่ถูกต้อง</u>** เพื่อให้เรียงลำดับกันของข้อมูลจากน้อยไปมาก โดยฟังก์ชัน insert จะต้องสนับสนุนการแทรกทั้ง 3 กรณี คือ

1. ตำแหน่งหน้าสุดของ List ที่มีข้อมูล
2. ตำแหน่งหลังสุดของ List ที่มีข้อมูล
3. ตำแหน่งระหว่างกลางของ List ที่มีข้อมูล

โดยโค้ดที่เขียนต้องคล้องจองกับโปรแกรมในหน้าที่ 5-7 และจะต้องสามารถใช้ทำงานได้ถูกต้องเมื่อแทรกโค้ดตรงบรรทัดที่ 87 ของหน้าที่ 7

(15 คะแนน)

ข้อที่ 3 POLYMORPHISM & STL *(80 คะแนน: 80 นาที)*

บริษัทจำลองแห่งหนึ่ง มีพนักงาน (Employee) สองประเภท คือ `Engineer` และ `Sales` โดยข้อมูลที่สนใจของพนักงานแต่ละคน คือ เงินเดือน และเงินรวมที่ได้รับ

- Engineer จะมีเงินเดือนประจำ และเงินที่มาจากการคำนวนคะแนน (score) จากการทำโปรเจค โดยคะแนนจะสะสมไปเรื่อย ๆ ได้

  เงินรวมที่ได้รับ = เงินเดือน + (คะแนนรวม x เงินเดือน)

  กำหนดให้ Engineer สามารถเพิ่ม score ได้ด้วยฟังก์ชัน

  ```
  void doProject(float score);
  ```

- Sales จะมีเงินเดือนประจำ และเงินที่มาจากยอดการขายในแต่ละโปรเจค โดยยอดการขายจะสะสมไปได้เรื่อย ๆ

  เงินรวมที่ได้รับ = เงินเดือน + (0.01 x ยอดขายรวม)

  กำหนดให้ยอดขายของ Sales สามารถเพิ่มได้ ผ่านฟังก์ชัน

  ```
  void sellProject(double amount);
  ```

โค้ดที่กำหนดให้ในตอนท้ายของข้อสอบชุดนี้ใน<u>หน้าที่ 8</u> เป็นนิยามของ `class Employee` ซึ่งมีฟังก์ชันสำคัญดังต่อไปนี้

- `double getSalary()` คืนค่า เงินเดือนประจำของพนักงาน
- `int getID()` คืนค่า หมายเลข ID ของพนักงาน ซึ่งจะถูกกำหนดโดยอัตโนมัติจากลำดับของการสร้างอ๊อบเจค Employee
- `char* getName()` คืนค่า ชื่อของพนักงาน
- `bool equals(Employee *e)` คืนค่า จริง เมื่อ ID ของพนักงานคนนั้นๆ กับ ID ของพนักงานที่กำหนดให้ในพารามิเตอร์ เป็น ID เดียวกัน เป็นเท็จในทางตรงกันข้าม
- `double getPaid()` คืนค่า เงินรวมที่พนักงานคนนั้นๆ จะได้รับ

จากข้อมูลที่กำหนดให้ จงตอบคำถามข้อ 3.1-3.3

3.1 โค้ดข้างล่าง เป็นโค้ดหลักของโปรแกรม จงเพิ่มเติมส่วนที่ขาดหายไป เพื่อให้โปรแกรมทำงานสอดคล้องกับผลลัพธ์ดังนี้ (20 คะแนน)

```
Pongsakorn gets 28000
Thanat gets 32500
Sanchai gets 33000
Total paid will be 93500
```

```
#include <iostream>
#include <stack>

#include "employee.h"
#include "engineer.h"
#include "sales.h"

using std::cout;
```

```
using std::endl;
using std::stack;
int main()
{
  Engineer *eng1 = new Engineer("Sanchai", 30000);
  eng1->doProject(0.1);

  Engineer *eng2 = new Engineer("Thanat", 25000);
  eng2->doProject(0.1);
  eng2->doProject(0.2);

  Sales *sls1 = new Sales("Pongsakorn", 20000);
  sls1->sellProject(300000);
  sls1->sellProject(500000);

  stack <Employee*> employees;
  employees.push(eng1);
  employees.push(eng2);
  employees.push(sls1);

  double total = 0;

  //add your code here for question 3.1

  delete eng1, eng2, sls1;
  return 0;
}
```

3.2 จงเขียนต้นแบบ และนิยามของ `class Engineer` หรือ `class Sales` เพียงหนึ่ง class ให้สอดคล้องกับสถานการณ์ที่กำหนดให้ (20 คะแนน)

3.3 จงอธิบายว่า เหตุการณ์ใดในสถานการณ์จำลอง ได้ใช้ประโยชน์จากเทคนิคที่กำหนดให้ด้านล่าง พร้อมทั้งยกโค้ดสั้น ๆ มาประกอบคำอธิบาย (40 คะแนน)

- Data Encapsulation

ตัวอย่างคำตอบ

Data Encapsulation เป็นเทคนิคที่ปกป้องข้อมูลที่สำคัญจากการเข้าถึง (ที่ไม่เหมาะสม) จากภายนอก ตัวอย่างเช่น การที่ class Employee มีการกำหนด modifier private ให้กับข้อมูล salary เพื่อให้มั่นใจว่า ภายนอกจะไม่สามารถเปลี่ยนแปลงค่า salary ได้

```
private:
        int salary;
```

- Inheritance
- Polymorphism
- Operator Overloading
- Container

ข้อที่ 4 DATA STRUCTURES CUSTOMIZATION *(40 คะแนน: 40 นาที)*

จากส่วนของโค้ด `main()` ในข้อ 3.1

```
stack <Employee*> employees;
employees.push(eng1);
employees.push(eng2);
employees.push(sls1);
```

หากมีการเพิ่มโค้ดอีกหนึ่งบรรทัดในตอนท้าย

```
employees.push(eng1);
```

เราจะพบว่า Stack จะยอมให้ `eng1` ซึ่งเคยทำการใส่ลงไปแล้ว ถูกใส่เข้าอีกครั้ง หากเราต้องการแก้ไข stack ให้ไม่รับข้อมูลซ้ำ วิธีการหนึ่งที่สามารถนำมาใช้คือการสืบทอด `stack <Employee*>` มาเพื่อเปลี่ยนแปลงคุณสมบัติในส่วนของการ push ใหม่ (customization)

4.1 จงเติมส่วนของโค้ดที่หายไปของ `class MyStack` เพื่อให้สามารถขจัดการใส่ข้อมูลซ้ำซ้อน (20 คะแนน)

```
#ifndef MYSTACK_H
#define MYSTACK_H

#include <stack>
#include <deque>
using std::stack;
using std::deque;

class MyStack : public stack <Employee*> {

  public:
    void push(Employee* em){
      //add your code here for question 4.1

    }
};
#endif
```

4.2 จงวิเคราะห์ว่า เทคนิคที่กำหนดให้ข้างล่าง จะเข้ามาช่วยเพิ่มความเร็วในการทำงานของฟังก์ชัน push ที่เขียนขึ้นใหม่ในข้อ 4.1 ได้หรือไม่ พร้อมทั้งให้เหตุผลประกอบ (20 คะแนน)

- การใช้งาน binary search
- การใช้ `vector` เป็น container ของ `stack` แทนการใช้ deque ในกรณีที่มีข้อมูลที่จำเป็นต้อง push ในปริมาณมาก ๆ


```
21    }; // end class ListNode
22
23    // constructor
24    template< class NODETYPE>
25    ListNode< NODETYPE >::ListNode( const NODETYPE &info )
26       : data( info ),
27         nextPtr( 0 )
28    {
29       // empty body
30
31    } // end ListNode constructor
32
33    // return copy of data in node
34    template< class NODETYPE >
35    NODETYPE ListNode< NODETYPE >::getData() const
```

โค้ดประกอบสำหรับคำถามข้อ 2.2

```
1      // listnode.h
2      // Template ListNode class definition.
3      #ifndef LISTNODE_H
4      #define LISTNODE_H
5
6      // forward declaration of class List
7      template< class NODETYPE > class List;
8
9      template< class NODETYPE>
10     class ListNode {
11        friend class List< NODETYPE >; // make List a friend
12
13     public:
14        ListNode( const NODETYPE & );  // constructor
15        NODETYPE getData() const;      // return data in node
16
17     private:
18        NODETYPE data;                 // data
19        ListNode< NODETYPE > *nextPtr; // next node in list
20
21     }; // end class ListNode
22
23     // constructor
24     template< class NODETYPE>
25     ListNode< NODETYPE >::ListNode( const NODETYPE &info )
26        : data( info ),
27          nextPtr( 0 )
28     {
29        // empty body
30
31     } // end ListNode constructor
32
33     // return copy of data in node
34     template< class NODETYPE >
35     NODETYPE ListNode< NODETYPE >::getData() const
36     {
37        return data;
38
39     } // end function getData
40
41     #endif
```



```
23 private :
24    ListNode < NODETYPE > *firstPtr ; // pointer to first node
25    ListNode < NODETYPE > *lastPtr ; // pointer to last node
26    ListNode < NODETYPE > *currentPtr ;
27    ListNode < NODETYPE > *previousPtr ;
28
29    // utility function to allocate new node
30    ListNode < NODETYPE > *getNewNode ( const NODETYPE & );
31 }; // end class List
32
33 // default constructor
34 template < class NODETYPE >
35 List< NODETYPE >::List()
36    : firstPtr ( 0 ),
37      lastPtr ( 0 )
38 {
```

```
1  // list.h
2  // Template List class definition.
3  #ifndef LIST_H
4  #define LIST_H
5
6  #include <iostream>
7
8  using std::cout;
9  using std::endl;
10 #include <new>
11 #include "listnode.h" // ListNode class definition
12
13 template < class NODETYPE >
14 class List {
15
16 public :
17    List(); // constructor
18    ~List(); // destructor
19
20    void insert ( const NODETYPE & );
21    bool isEmpty ( ) const;
22    void print() const;
23 private :
24    ListNode < NODETYPE > *firstPtr ; // pointer to first node
25    ListNode < NODETYPE > *lastPtr ; // pointer to last node
26    ListNode < NODETYPE > *currentPtr ;
27    ListNode < NODETYPE > *previousPtr ;
28
29    // utility function to allocate new node
30    ListNode < NODETYPE > *getNewNode ( const NODETYPE & );
31 }; // end class List
32
33 // default constructor
34 template < class NODETYPE >
35 List< NODETYPE >::List()
36    : firstPtr ( 0 ),
37      lastPtr ( 0 )
38 {
39 // empty body
40
41 } // end List constructor
42
43 // destructor
44 template < class NODETYPE >
45 List< NODETYPE >::~List()
46 {
47    if ( !isEmpty () ) { // List is not empty
48          cout << "Destroying nodes ...\n" ;
49
50    ListNode < NODETYPE > *currentPtr = firstPtr ;
51    ListNode < NODETYPE > *tempPtr ;
52
53          while ( currentPtr != 0 ) { // delete remaining nodes
54                tempPtr = currentPtr ;
55                cout << tempPtr ->data << '\n';
56                currentPtr = currentPtr ->nextPtr ;
57                delete tempPtr ;
58
59          } // end while
60
61    } // end if
62
```

```
63    cout << "All nodes destroyed\n\n" ;
64
65 } // end List destructor
66
67 // is List empty?
68 template < class NODETYPE >
69 bool List< NODETYPE >::isEmpty () const
70 {
71    return firstPtr == 0;
72
73 } // end function isEmpty
74
75 // return pointer to newly allocated node
76 template < class NODETYPE >
77 ListNode < NODETYPE > *List< NODETYPE >::getNewNode (
78 const NODETYPE &value )
79 {
80    return new ListNode < NODETYPE >( value );
81 } // end function getNewNode
82
83
84 template < class NODETYPE >
85 void List< NODETYPE >::insert ( const NODETYPE &value) {
86 ListNode < NODETYPE > *newPtr = getNewNode ( value );
87    //add your code here for question 2.2
88
89
90 }
91 template < class NODETYPE >
92 void List< NODETYPE >::print() const
93 {
94    if ( isEmpty () ) {
95          cout << "The list is empty\n\n" ;
96          return ;
97    }
98    ListNode < NODETYPE > *currentPtr = firstPtr ;
99
100   cout << "The list is: " ;
101
102   while ( currentPtr != 0 ) {
103         cout << currentPtr ->data << ' ';
104         currentPtr = currentPtr ->nextPtr ;
105   } // end while
106   cout << "\n\n" ;
107 }
108
109 #endif
110
```