บทที่ 10

Template Class

วัตถุประสงค์

1. เพื่อให้นักศึกษาทราบถึงรูปแบบในการสร้าง Template Class
2. เพื่อให้นักศึกษาเข้าใจถึงกระบวนการในการสร้าง Template Class
3. เพื่อให้นักศึกษาสามารถประยุกต์ใช้ Template Class ในการแก้ปัญหาโปรแกรมที่เหมาะสมได้

การพัฒนาโปรแกรมเชิงวัตถุนั้นในบทที่ผ่านมาเป็นการสร้างคลาสซึ่งเราต้องกำหนดรายละเอียดหรือหน้าที่รวมทั้งความสัมพันธ์ระหว่างออบเจ็กต์ต่างๆได้โดยเน้นที่ผลลัพธ์ของการทำงานมากกว่ากระบวนการทำงานโดยกระทำกับชนิดข้อมูลต่างๆในลักษณะของสถิตย์(static) เป็นการอธิบายการสร้างคลาสเบื้องต้นซึ่งการทำงานของคลาสนั้นยังมีขอบเขตของการทำงานที่จำกัดอยู่ กระบวนการสร้างคลาสของภาษา C++ นั้นยอมให้เราสามารถออกแบบฟังก์ชันหนึ่งๆให้มีการตอบสนองได้หลายรูปแบบได้ ซึ่งเป็นคุณสมบัติที่เรียกว่า Polymorphism หมายถึงการเรียกใช้ในชื่อฟังก์ชันเดียวกันแต่สามารถตอบสนองการทำงานได้หลายรูปแบบ ซึ่งคุณสมบัตินี้ตอบสนองให้การใช้งานมีความหยืดหยุ่นมากขึ้น ในบทนี้จะเป็นการอธิบายถึงตัวอย่างที่แสดงให้เห็นถึงการตอบสนองการใช้งานของผู้ใช้ โดยผู้ใช้สามารถกำหนดชนิดข้อมูลที่เหมาะสมเพื่อให้สามารถกระทำการประมวลผลตามกิจกรรมฟังก์ชันต่างๆ ในคลาสได้หลากหลายขึ้นอยู่กับการแก้ปัญหาโดยไม่ยึดติดกับชนิดข้อมูลใดชนิดข้อมูลหนึ่ง

10.1 Template Classes

Template Classes เป็นการสร้างคลาสเพื่อใช้งานที่ได้รับความนิยมมาก เพราะเปิดโอกาสให้ผู้ใช้สามารถใช้งานต่างๆได้หลากหลายขึ้น โดยสามารถกระทำการปฏิบัติการกับชนิดรองข้อมูลที่กำหนดโดยผู้ใช้งานได้ ซึ่งถึงแม้ว่าออบเจ็กต์นั้นๆจะอยู่ในคลาสเดียวกัน แต่สามารถใช้งานกับชนิดของข้อมูลแตกต่างกันได้ ดังนั้นรูปแบบของการกำหนดคลาสชนิดนี้จะเปลี่ยนแปลงไปดังนี้

รูปแบบของ Template Classes

```
template <class T>
class class-name
{
public:
```

- List of class variables, types, constants, etc. (if any) that may be accessed by name from outside the class
- Prototype for each function that may be accessed by name from outside the class

. . .

private:

- List of class variables, types, constants, etc., that are intended to be hidden from reference from outside the class
- Prototype for each function (if any) to be hiden from outside the class

. . .

};

โดย Class class-name คือ template class โดยมีพารามิเตอร์ T ซึ่ง T เป็นเนื้อที่ที่ผู้ใช้หรือระบบกำหนดขึ้นสำหรับสำหรับเก็บชนิดข้อมูลที่ผู้ใช้กำหนด ต้องอยู่ในเครื่องหมาย < > เมื่อสังเกตให้ดีจะเห็นว่าสิ่งที่เปลี่ยนไปคือการกำหนด template <class T> ก่อนกำหนดต้นแบบหรือรายละเอียดของคลาสนั้นเอง

สำหรับการประกาศอ็อบเจ็กต์ที่มีการใช้ Template Classes นี้จะมีการปรับเปลี่ยนโดยมีการส่งชนิดของข้อมูลเพื่อกระทำงานในคลาสมีรูปแบบดังนี้

การประกาศ Template Classes	
รูปแบบ:	class-name<type> an-object:
ตัวอย่าง	indexList<int> intList;

type ที่อยู่ในเครื่องหมาย < > จากรูปแบบการประกาศนี้อาจเป็นชนิดของข้อมูลอะไรก็ได้ที่ผู้ใช้ต้องการให้ปฏิบัติงานในคลาส ส่วน class-name เป็นชื่อของ Template class ที่กำหนดขึ้น สำหรับ an-object เป็นชื่อของออบเจ็กต์ที่สร้างขึ้นมาโดยจะมีความสอดคล้องกับชนิดของข้อมูลที่ได้ประกาศ ซึ่งเมื่อมีการเรียกใช้ฟังก์ชั่นทั้งหลายในคลาสจะมีการแทนชนิดของข้อมูลเพื่อให้สามารถใช้งานได้ตามที่ได้ตามที่ต้องการ

เพื่อให้เข้าใจดียิ่งขึ้นมารู้จักวิธีการสร้าง Template class เบื้องต้นดังนี้

ตัวอย่างที่ 10.1 Class dummy

เป็นตัวอย่างแรกเพื่อให้ทราบถึงวิธีการทำงานและวิธีการเรียกใช้คลาสที่ผู้ใช้สามารถกำหนดชนิดของข้อมูลเพื่อปฏิบัติงานได้แตกต่างกัน โดยในที่นี้จะกำหนด

Attribute เพียง 2 ตัว ดังนี้

1. T item; โดย T ในที่นี้คือชนิดของข้อมูลที่ถูกกำหนดด้วยผู้ใช้

2. bool defined; เป็นการตรวจสอบค่าที่เก็บในตัวแปร item ว่าถูกกำหนดแล้วหรือไม่ กรณีที่มีข้อมูลจะมีค่าเท่ากับ true มิเช่นนั้นมีค่าเท่ากับ false

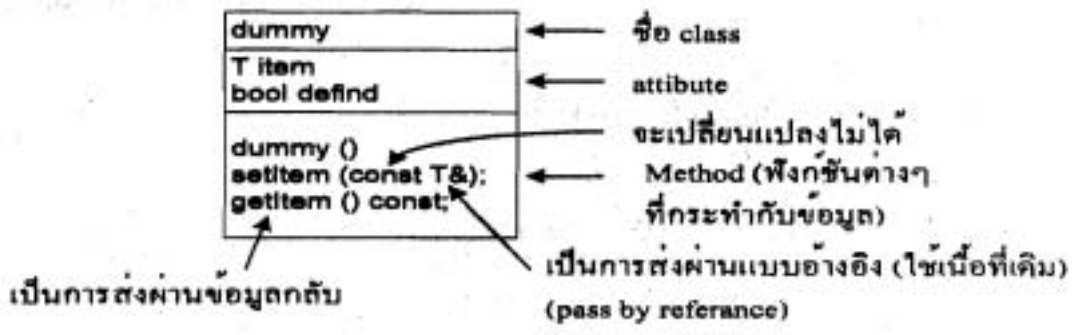
Member function

1. dummy () ; เป็น constructor ที่กำหนดค่าเริ่มต้นให้แก่ attribute ต่างๆ ซึ่งจากตัวอย่างมีเพียง 2 ตัวแปรเท่านั้น คือ item กับ defined

2. void setItem (const T&); เป็นฟังก์ชันที่ทำหน้าที่กำหนดค่าข้อมูลให้แก่ตัวแปร item โดยมีการส่งผ่านค่ามาจากจุดเรียกใช้ โดยมีการส่งผ่านค่าแบบ pass by value ซึ่งพารามิเตอร์ที่กำหนดขึ้นทำหน้าที่เป็น input สำหรับชนิดของข้อมูลจะมีเครื่องหมาย & บ่งบอกถึงการอ้างอิงถึงชนิดของข้อมูลที่ผู้ใช้กำหนดขึ้น ส่วนคำเฉพาะ const จะเป็นการกำหนดให้ทำหน้าที่อ่านอย่างเดียวแต่ไม่สามารถเปลี่ยนแปลงค่าของข้อมูลได้

3. T getItem () const ; เป็นฟังก์ชันที่ส่งค่าของ item กลับไปยังจุดเรียกใช้เป็นชนิดข้อมูล T โดย const เป็นคำเฉพาะที่บ่งบอกว่าฟังก์ชันนี้เป็น Accessor คืออ่านหรือดึง Attribute เพื่อทำงานได้อย่างเดียว ไม่มีการเปลี่ยนแปลงค่าแต่อย่างใด

เราสามารถสร้างแฟ้มต้นแบบในชื่อของ dummy.h ได้ตามรูปแบบได้ดังนี้



```
// Header file for template class dummy
// file: dummy.h
// Header file for template class dummy

#ifndef  DUMMY_H
#define DUMMY_H

template <class T>
class dummy
{
  public:
    // constructor
    dummy () ;

  // Stores a value of type T
  void setItem (const T&);  // IN: value to store

  // Retrieves a value of type T
  T getItem ()  const;
private:
  T item;                 // Storage for a value of  type T
  Bool defined;           // Indicates whether item is defined
};
#endif   // DUMMY_H
```

หลายคนอาจสงสัยว่า แล้วเราจะมีวิธีการเรียกใช้คลาส dummy นี้ได้อย่างไร และจะมีการส่งผ่านชนิดของข้อมูลไปยังคลาสที่แตกต่างกันได้อย่างไร มาพิจารณาแฟ้ม dummyTest.cpp ดังต่อไปนี้

```
// Driver function for template class dummy
// File: dummyTest.cpp
// Tests function dummy
#include "dummy.h"
#include <iostream>
#include <string>
using namespace std;
int main ( )
{
  dummy<int> numDepend;                    // object numDepend
  dummy<string> spouseName;                // object spouseName
  int num;
  string name;

  // Store data in objects numDepend and spouseName
  numDepend.setItem(2) ;
  spouseName.setItem("Caryn");
  // Retrieve and display values stored
  num = numDepend.getItem() ;
  name = spouseName.getItem() ;
  cout << num << endl;
  cout << name << endl;
  return 0;
}
```

การทำงานของโปรแกรมนี้เป็นการกำหนดออบเจ็กต์ 2 ตัวดังนี้

```
dummy<int> numDepend;                    // object numDepend
dummy<string> spouseName;                // object spouseN
```

โดยทั้งสองตัวเป็นออบเจ็กต์ในคลาส dummy แต่มีการปฏิบัติงานในคลาสแตกต่างกัน เพราะกำหนดชนิดของข้อมูลแตกต่างกัน สำหรับ numDepend มีการปฏิบัติงานกับชนิดข้อมูลเป็นเลขจำนวนเต็ม แต่ spouseName มีการปฏิบัติงานกับชนิดข้อมูลที่เป็นข้อความ สำหรับคำสั่งในการกำหนดค่าให้แก่ตัวแปร item ของทั้งสองตัวแปรแตกต่างกันดังนี้

```
numDepend.setItem(2) ;
spouseName.setItem("Caryn");
```

โดยออบเจ็กต์ numDepend จะมีการส่งผ่านค่า 2 ไปยังฟังก์ชัน setItem และออบเจ็กต์ spouseName จะส่งผ่านข้อความ "Caryn" ไปยังฟังก์ชัน setItem เห็นได้ว่ามีการเรียกใช้ฟังก์ชันชื่อเดียวกันแต่มีการปฏิบัติงานภายในฟังก์ชันแตกต่างกันโดยมีการกระทำกับข้อมูลที่มีชนิดแตกต่างกัน ซึ่งวิธีการสร้างคลาสในลักษณะนี้ทำให้เกิดความหยืดหยุ่นในการพัฒนาโปรแกรมเป็นอันมาก ทำให้โปรแกรมมีขนาดที่เล็กลงและไม่ต้องมีการสร้างฟังก์ชันที่มีการทำงานคล้ายๆกันทำให้เกิดความสับสน สำหรับการดึงข้อมูล item ของออบเจ็กต์ออกมาเพื่อทำงานต้องกระทำผ่านฟังก์ชันชื่อ getItem ดังนี้

```
num = numDepend.getItem() ;
name = spouseName.getItem() ;
```

ในการดึงข้อมูล item ออกมานั้นต้องมีตัวแปรรองรับค่าที่ถูกดึง โดยผู้ใช้ต้องประกาศหรือกำหนดตัวแปรที่รับค่าให้เป็นชนิดข้อมูลเดียวกันด้วย ซึ่งในที่นี้ num ต้องมีชนิดข้อมูลเป็น int และ ตัวแปร name ต้องมีชนิดข้อมูลเป็น string ตราวนี้เรามาดูวิธีการทำงานภายในฟังก์ชันกันว่ามีการสร้างอย่างไร ในที่นี้เราจะสร้างฟังก์ชันต่างๆในแฟ้ม dummy.cpp ดังนี้

```
// Implementation file for template class dummy
// File:  dummy.cpp
// Implementation file for template class dummy
#include "dummy.h"
#include <iostream>
using namespace std;

// constructor
template <class T>
dummy<T>: :dummy ( )
{
  defined = false;        // No value stored yet
}
// Stores a value of type t
template <class T>
void dummy<T>: :setItem(const T& aVal)     // IN:  value to store
{
    item = aVal;
    defined = true;
}
// Retrieves a value of type t
template <class T>
T dummy<T>: :getItem ( )  const
{
    if  (defined)
       return item;
        else
```

```
        cerr << "Error – no value stored!" << endl;
}
```

จะเห็นได้ว่าสิ่งที่มีการเปลี่ยนแปลงเมื่อเทียบกับการสร้างคลาสในบทที่แล้วคือการระบุ template <class T> ก่อนฟังก์ชันทุกตัว และมีการกำหนด <T> ไว้ภายหลังชื่อของคลาสก่อนเครื่องหมาย :: ในทุกๆฟังก์ชัน เพื่อให้ทราบว่าการปฏิบัติงานภายในฟังก์ชันมีการอ้างอิงถึงชนิดของข้อมูลที่กำหนดโดยผู้ใช้ แต่นี้เราสามารถทำงานกับคลาสได้โดยใช้ชนิดข้อมูลที่แตกต่างกันได้แล้ว

สำหรับการสร้าง Template class dummy นี้ เราสามารถจะสร้างรวมกันได้โดยไม่ต้องแบ่งแยก ต้นแบบ และการสร้างฟังก์ชัน ได้อีกวิธีหนึ่งดังนี้

```
// Template class dummy with member functions inserted
// File: dummyFun.h
// Definition file for template class dummy with functions
#include <iostream>
using namespace std;
#ifndef  DUMMY_FUN_H
#define DUMMY_FUN_H

template <class T>
class dummy                          // with function definitions
{
  public:
    // constructor
    dummy ()
    {
      defined = false;               // no value stored yet
    }
```

```
    // Stores a value of type t
    void setItem (const T& aVal)        // IN: the value to store
    {
      item = aVal;
      defined = true;
    }
    // Retrieves a value of type t
    T getItem ( ) const
    {
      if (defined)
      return item;
        else
          cerr << "Error – no value stored!" << endl;
    }
  private:
      T item;             // Storage for a value of type T
      bool defined;       // Indicates whether item is defined
};
#endif   // DUMMY_FUN_H
```

การสร้างโดยวิธีนี้จะไม่นิยมเพราะเราจะไม่ให้ผู้เรียกใช้งานทราบถึงการปฏิบัติงานภายในฟังก์ชันตามแนวความคิดเชิงวัตถุ เพราะเราเน้นถึงผลลัพธ์ของการใช้งานเท่านั้น ดังนั้นการกำหนดคลาสควรแยกเป็นแฟ้มๆไปจะดีกว่า

ตัวอย่างที่ 10.2 Class indexList

หลายคนอาจสงสัยว่าถ้าเป็นการสร้างคลาสซึ่งมีโครงสร้างของข้อมูลที่ซับซ้อนขึ้น เป็นกลุ่มของข้อมูลเราจะมีการกำหนด Attribute และ Member Function เป็นอย่างไร สำหรับตัวอย่างนี้มีคำตอบ ตัวอย่างนี้เป็นตัวอย่างของการสร้าง Template class ชื่อ indexList ซึ่งมีการกำหนด

Attribute ในคลาสเป็นกลุ่มของอาเรย์ 1 มิติ โดยผู้ใช้สามารถกำหนดชนิดของข้อมูลที่ถูกจัดเก็บได้แตกต่างกันขึ้นอยู่กับการปฏิบัติงาน เพื่อให้เข้าใจดียิ่งขึ้นสมมุติว่ามีการระบุรายละเอียดของคลาสมีโครงร่างคร่าวๆของการทำงานดังนี้

SPECIFICATION FOR INDEXED LIST CLASS

Attributes for Indexed List Class

elements[ ]	Array of data items
int size	Count of items

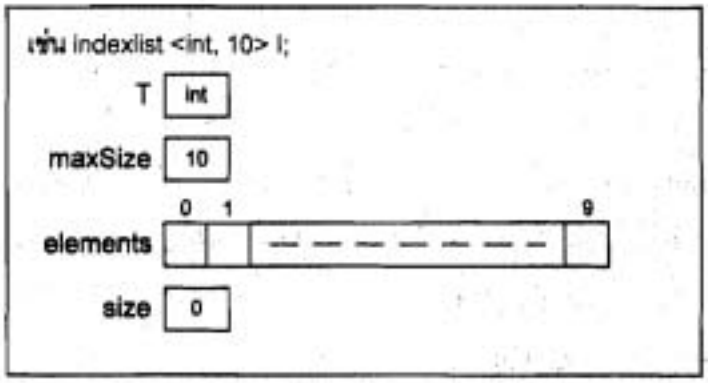
Member Functions for Indexed List Class

indexList	Constructor
append	Appends a new item to the indexed list (i.e., insert at the end).
insert	Inserts an item a specified index after moving the elements staring at that index to make room.
replace	Replaces an item at a specified index.
retrieve	Retrieves the value stored at a specified index.
remove	Deletes the value stored at a specified index.
findMin	Locates the smallest value in a portion of an indexed list.
findMax	Locates the largest value in a portion of an indexed list.
search	Searches for a target value in an indexed list.
sort	Sorts an indexed list.
read	Reads data into an indexed list from an input file or the keyboard, starting at element 0.
display	Displays the items in the indexed list.
getSize	Gets the size of the indexed list.

การกำหนดรายละเอียดในส่วนของ Attribute นั้นมีตัวแปร 2 ตัวโดยตัวแปร elements เป็นข้อมูลชนิดอาเรย์ 1 มิติ ในที่นี้เราสามารถกำหนดให้เก็บเป็นข้อมูลชนิดใดก็ได้ แต่ผู้ใช้งานต้องกำหนดขึ้น แต่เนื่องจากการกำหนดเนื้อที่ในการจัดเก็บของผู้ใช้มีขนาดไม่เท่ากัน การทำงานของคลาสนี้ผู้ใช้ต้องกำหนดขนาดสูงสุดของข้อมูลที่ระบบต้องจองด้วย ซึ่งการกำหนดค่าต่างๆเหล่านี้ผู้ใช้ต้องกำหนดในขณะที่มีการประกาศออบเจ็กต์ใหม่ที่เป็นสมาชิกของคลาสนี้ ดังมีรูปแบบดังนี้

<class T, int maxSize>

โดย T คือชนิดของข้อมูลที่กำหนดให้กระทำในคลาส ส่วน maxSize เป็นการกำหนดขนาดของข้อมูลที่ทำการจัดเก็บ



ดังนั้นการกำหนดทั้งชนิดของข้อมูลและขนาดของอาเรย์ที่สามารถเก็บข้อมูลได้สูงสุด โดยผู้ใช้นี้เองทำให้เกิดความหยืดหยุ่นเป็นอย่างมาก ถึงแม้ว่าออบเจ็กต์ต่างๆจะเป็นสมาชิกเหมือนกันอยู่ในคลาสเดียวกัน แต่การจองเนื้อที่ในการจัดเก็บรวมทั้งชนิดของข้อมูลแต่ละตัวมีขนาดไม่เท่ากัน แต่สามารถเรียกใช้ฟังก์ชันในคลาสเดียวกันได้

เราสามารถกำหนดต้นแบบได้โดยจัดเก็บในแฟ้ม indexList.h ดังนี้

```
// Class indexList header file
// File:  indexList.h
// Definition of indexed list template class

#ifndef  INDEXLIST_H
```

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

template <class T, int maxSize>
class indexList
{
   public:
    // Constructor
   indexList ( ) ;

    // Add an item to the end of an indexed list
    bool append (const T&); // IN: item appended

    // Replace an element at a specified index
    bool replace (int,          // IN: index
                  const T&);    // IN: item inserted

    // Insert an element at a specified index
    bool retrieve (int,         // IN: index
                   T&) const;   // OUT: value retrieved

    // Delete an element at a specified index
    bool remove(int);           // IN: index

    // Find index of smallest value in a sublist
     int findMin(int,           // IN:  start index
```

```
                int) const;    // OUT: end index

    // Find index of largest value in a sublist
    int findMax (int,          // IN:  start index
                 int) const;   // OUT: end index

    // Find index of a target item
    //   Returns -1 if target item not found
    int search (const T&) const;        // IN: target item

    // Sort an indexed list
    void selSort ( ) ;

    // Read data from an input stream into the list
    void display ( ) const;

    // Display the list contents
    void display ( ) const;

    // Get the current size
    int getSize ( ) const;

  private:
    T elements [maxSize];      // Storage for elements
    int size;                  // Count of elements in list
};
#endif  // INDEXLIST_H
```

สำหรับการใช้งานคลาส ในที่นี้เราจะกำหนดออบเจ็กต์เพื่อกำหนดชนิดของข้อมูลและขนาดของอาเรย์ 2 ตัวคือ

```
indexList<int, 10>      myIntData;    // list of ints
indexList<string, 5>    myStringData; // list of strings
```

ในการอ้างถึงฟังก์ชันต่างๆนั้นการทำงานของสองออบเจ็กต์นี้จะปฏิบัติงานได้เหมือนกันดังนี้

```
// Testing class indexList
// File: indexListTest.cpp
// Testing the indexed list template class
#include "indexList.h"
#include <iostream>
#include <string>
using namespace std;
int main ( )
{
   indexList<int, 10>      myIntData;        // list of ints
   indexList<string, 5>    myStringData;     // list of strings
   string aAtring;
   int anInt;
   bool aBool;

   // Store the integer data.
   myIntData.append (5);
   myIntData.append (0);
   myIntData.append (-5);
   myIntData.append (-10);
```

```
    // Store the string data.
    Cout << "Read a list of strings:" << endl;
    myStringData.read (cin);    // read from keyboard

    // Sort the indexed lists.
    myIntData.selSort ( ) ;
    myStringData.selSort ( ) ;

   // Retrieve and display the first value in each list.
    aBool = myIntData.retrieve (0, anInt);
    if (aBool)
       cout << "First integer value after sorting is"
            << aString << endl << endl;

    // Display each list size and contents
    cout << "The indexed list of integers contains"
         << myIntData.getSixe ( ) << "values." << endl;
    cout << "Its contents follows:" << endl;
    myIntData.display ( ) ;
    cout << endl << "The indexed list of strings contains"
    << myStringData.getSize ( ) << "values." << endl;
    cout << "Its contents follows:" << endl;
    myStringData.display ( ) ;

    return 0;
}
```

เมื่อนำโปรแกรมไปปฏิบัติงานโดยป้อนข้อมูลเพื่อทดสอบจะได้ผลลัพธ์ปรากฏดังนี้

```
Read a list of strings:
Enter number of list items to read: 3
Enter next item – Robin
Enter next item – Beth
Enter next item – Koffman
First integer value after sorting is –10
First string value after sorting is Beth
The indexed list of integers contains 4 values.
Its contents follows :
-10
-5
0
5
The indexed list of strings contains 3 values.
Its contents follows :
Beth Koffman
Robin
```

เราสามารถสร้างฟังก์ชันเพื่อปฏิบัติงานได้ตามตัวอย่างดังนี้

```
// Some member functions for class indexList
// File:  indexList.cpp
// Indexed list class implementation

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

```
template <class T, int maxSize>
indexList<T,  maxSize> : : indexList ( )          // constructor
{
     size = 0;  // list is empty
}

// Add an item to the end of an indexed list
// Pre:   item is defined
//  Post:  If size < maxSize, item is appended to list
// Returns:  true if item was appended; otherwise, false
template <class T, int maxSize>
bool indexList<T, maxSize> : : append (const T& item)
{
     bool result;
     // Add item to the end of the list if list is not full.
     if (size < maxSize)
     {
        elements [size] = item;
        size++;
        result = true;
      }
     else
     {
        cerr << "Array is filled – can't append!" << endl;
        result = false;
     }
```

```
    return result;
}

// Replace an item at a specified index.
// Pre:  item and index are defined
// Post:  item is placed a position index if valid
// Returns:  true if item was inserted; otherwise, false
template <class T, int maxSize>
bool indexList<T, maxSize> : : replace (int  index, const T& item)
{
    bool result;
    // Overwrite a list element if index is valid.
    if (index >= 0  &&  index < size)
    {
      elements [index] = item;
      result = true;
    }
    else
    {
      cerr  <<  "Index"  <<  index  <<  " not in filled part "
            <<  " – can't insert!"  >> endl;
      result = false;
    }
    return result;
}

// Retrieve an item at a specified index
```

```
// Pre:  item and index are defined
// Post:  if index is valid, elements [index] is returned
// Returns:  true if item was returned; otherwise, false
template <class T, int maxSize>
bool indexList<T, maxSize> : : retrieve (int index, T& item) const
{
    bool result;
    // Return a list element through item if index is valid.
    if ( index >= 0  &&  index < size)
     {
      item = elements [index];
      result = true;
    }
     else
    {
      cerr << "Index" << index << "not in filled part"
           << " – can't retrieve!" << endl;
      result = false;
    }
    return result;
}

// Delete an element at a specified index
// Pre:  index is defined
// Post:  if index is valid, elements[index] is deleted
//           and size is decremented.
// Returns:  true if item was deleted; otherwise, false
```

```
template <class T, int maxSize>
bool indexList<T, maxSize> : : remove(int index)
{
    int i;
    bool result;
   // Delete element at index i by moving elements up
    if (index >= 0  &&  index < size)
    {
      // Move each element up 1 position
      for  (i = index + 1;  i < size; i++)
           elements[i-1] = elements [i];
      size--;                    //    decrement size
      result = true;
    }
    else
    {
      cerr << "Index" << index << "not in filled part"
           << " – can't delete!" << endl;
      result = false;
    }
    return result;
}

// Read data from an input stream into the list
// Pre:  none
// Post:  All data items are stored in array elements
//        and size is the count of items
```

```
template <class T, int maxSize>
void indexList<T, maxSize>: :read (istream& ins)
{  int numItems;                    // input – number of items to read
   T nextItem;              // input – mext data item

   cout << "Enter number of list items to read: ";
   ins >> numItems;
   ins.ignore(80, '\n');   // skip newline

   // If numItems is valid, read each list element,
   // starting with first element.
   size = 0;                // The list is empty.
   if (numItems >= 0 && numItems <= maxSize)
      while (size < numItems)
     {
       cout << "Enter next item - ";
       ins >> nextItem;
       elements[size] = nextItem;
       size++;
     }
   else
       cerr << "Number of items" << numItems
            << " is invalid" << " – no data entry!" << endl;
}

// Display the list contents
// Pre: none
```

```
// Post:  Displays each item stored in the list
template <class T, int maxSize>
void indexList<T, maxSize>: :display ( ) const
{
    // Display each list element.
    for (int i = 0; i < size; i++)
        cout << elements [i]  <<  endl;
}

// Find index of a target item
// Pre:  none
// Post:  Returns the index of target if found;
//        otherwise, return –1.
template <class T, int maxSize>
int indexList<T,  maxSize>: :search
    (const T& target) const
{
    for (int i = 0; i < maxSize; i++)
      if  (elements[i] = = target)
          return i;                  // target found at position i
    // target not found
    return –1;
}

// Sort the indexed list
template <class T,  int maxSize>
void indexList<T,  maxSize>: :selSort ( )
```

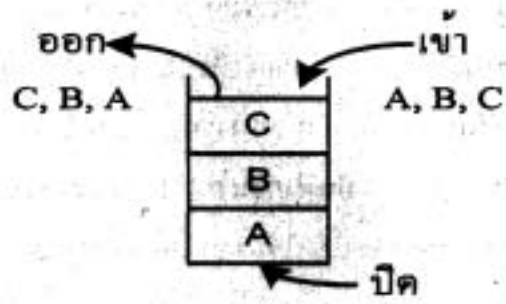
```
{
    // Selection sort stub -- do nothing
}

// Get the current size
template <class T, int maxSize>
int indexList<T, maxSize>: :getSize ( ) const
{
    return size;
}
```

สำหรับแฟ้ม indexList.cpp นี้ยังไม่สมบูรณ์นักศึกษาทดลองเขียนฟังก์ชันที่ยังขาดอยู่เพื่อให้โปรแกรมสมบูรณ์ และสามารถเรียกใช้งานได้

10.2 Template class stack

สแตก (stack) เป็นโครงสร้างข้อมูลที่มีการปฏิบัติงานในลักษณะของ LIFO (Last in-First out) เป็นการปฏิบัติงานที่สมาชิกตัวหลังสุดจะได้รับการบริการเป็นอันดับแรก และสมาชิกตัวแรกจะได้รับบริการเป็นลำดับสุดท้าย โครงสร้างชนิดนี้สามารถเปรียบเทียบกับการดำเนินกิจกรรมในชีวิตประจำวันของเราได้เช่น การดึงกระดาษจากกล่องทิชชูที่เป็นลักษณะเป็นสี่เหลี่ยมผืนผ้าโดยกระดาษชำระที่อยู่ด้านบนจะถูกดึงออกมาใช้งานก่อนส่วนกระดาษในแผ่นที่ดึงออกมาหลังสุดจะถูกนำเข้าไปเก็บไว้เป็นลำดับแรก หรือการใส่ถ่านไฟฉายในกระบอกไฟฉายที่ ถ่านไฟฉายก้อนที่ใส่หลังสุดจะถูกนำออกก่อน จะสังเกตุได้ว่ากิจกรรมที่นำข้อมูลหรือสมาชิกเข้าและออกนั้นจะกระทำที่ปลายข้างใดข้างหนึ่งโดยเฉพาะโดยมีปลายอีกด้านเป็นปลายปิด วิธีการในลักษณะที่กล่าวมานี้เราสามารถนำมาประยุกต์ใช้ในการเขียนหรือพัฒนาโปรแกรมในการแก้ปัญหาบางลักษณะได้ดี โดยข้อมูลที่เก็บในสแตกนั้นจะแทนด้วยกลุ่มของข้อมูลที่ต่อเนื่องหรือเชื่อมโยงกันเป็นลิสต์หรือแถวก็ได้ การนำข้อมูลเก็บเข้าไปในสแตกเรียกว่าการ push โดยกระทำที่ปลายด้านใดด้านหนึ่ง การนำข้อมูลออกจากสแตกเรียกว่า การ pop โดยจะกระทำได้ครั้งละ 1 สมาชิกและ

ปลายด้านเดียวกับการ push ด้วย โดยในบทนี้เราจะใช้โครงสร้างชนิดแถว 1 มิติในการอธิบายถึงการประยุกต์การปฏิบัติงานกับสแตก โดยทั่วไปเราจะมีตัวแปร top ทำหน้าที่ในการชี้ข้อมูลตัวสุดท้ายของสแตก ทำให้ทราบว่าการนำสมาชิกออกจะกระทำที่ตำแหน่งที่ top ชี้อยู่ หรือการนำเข้าต้องกระทำหลังจากตำแหน่งที่ top ชี้อยู่ การประยุกต์งานแก้ปัญหาที่เราคุ้นเคยกันเป็นอย่างดีคือการใช้โครงสร้างนี้ไปใช้แก้ปัญหางานในลักษณะของการเรียกตัวเอง หรือการเรียกฟังก์ชันซ้อนฟังก์ชัน จะเห็นได้ว่าฟังก์ชันใดเรียกก่อนจะส่งค่าผลลัพธ์กลับเป็นลำดับสุดท้าย แต่ถ้าฟังก์ชันใดเรียกทีหลังจะถูกปฏิบัติงานและส่งผลลัพธ์กลับมายังจุดเรียกใช้ก่อนเสมอ



สำหรับตัวอย่างต่อไปนี้ เราจะสร้างคลาสชื่อ stack เพื่อจำลองการปฏิบัติงานพื้นฐานที่สำคัญของโครงสร้างข้อมูลชนิดสแตก โดยผู้ใช้สามารถสร้างเป็น Template class ที่กำหนดชนิดข้อมูลที่กระทำในโครงสร้างนี้ได้

ตัวอย่างที่ 10.3 Template Class stack

เป็นการสร้างคลาส stack เพื่อผู้ใช้งานสามารถเรียกใช้ได้จากโปรแกรมภายนอก จากการพิจารณารายละเอียดการทำงานของสแตกแล้วเราสามารถออกแบบหรือระบุรายละเอียดของสแตกได้ดังนี้

Attribute ประกอบด้วย

1. int top; กำหนดตัวแปร top ให้ชี้ที่ข้อมูลตัวบนสุดของสแตก ในกรณีที่สแตกว่างค่าของ top จะมีค่าเท่ากับ -1

2. stackElType item[maxSize]; กำหนดตัวแปร item เป็นอาเรย์ 1 มิติที่เก็บเป็นข้อมูลเป็นชนิด stackElType ซึ่งสามารถกำหนดโดยผู้ใช้ภายนอกได้ รวมทั้งผู้ใช้ภายนอกสามารถกำหนดขนาดสูงสุดในการจัดเก็บข้อมูลได้อีกด้วย

Member Function ประกอบไปด้วย

1. stack(); เป็น constructor ที่กำหนดค่าเริ่มต้นให้แก่ตัวแปร top กำหนดให้มีค่าเริ่มต้นเท่ากับ -1 เมื่อมีการประกาศตัวแปรออบเจ็กต์ใหม่ ซึ่งหมายถึงสแตกว่างไม่มีข้อมูล
2. bool push (const stackElType& x); เป็นการรับค่า(INPUT)ที่ส่งมาจากการเรียกใช้ฟังก์ชั่น โดยนำข้อมูลที่ส่งมาเก็บในสแตก โดยค่าที่รับจะมีค่าของข้อมูลสอดคล้องกับชนิดข้อมูลที่กำหนด โดยมีการเลื่อน top ให้ชี้ที่ตำแหน่งข้อมูลตัวสุดท้าย
3. bool pop (stackElType& x); เป็นฟังก์ชันในการดึงข้อมูลตัวบนสุดของสแตกออกมาเก็บในตำแหน่งที่อ้างอิงจากจุดเรียกใช้ และให้ตัวแปร top เลื่อนลงมาชี้ที่ตำแหน่งลำดับถัดมา
4. bool peek (stackElType& x) const; เป็นการดึงข้อมูลตัวบนสุดของสแตก โดยไม่มีการเปลี่ยนแปลงค่าใดๆในสแตก โดย top และข้อมูลที่เก็บในอาเรย์ยังมีค่าคงเดิมทุกประการ
5. bool isEmpty() const; เป็นการตรวจสอบว่าสแตกว่างใช่หรือไม่ กรณีที่สแตกว่างจะส่งค่า true กลับ แต่ถ้าไม่ว่างจะส่งผ่านค่า false กลับ
6. bool isFull() const; เป็นการตรวจสอบข้อมูลในสแตกว่าเต็มหรือไม่ กรณีที่เต็มจะส่งผ่านค่า true กลับ มิฉะนั้นจะส่งค่า false กลับ

จากการระบุรายละเอียดคุณสมบัติของคลาส stack ดังกล่าวข้างต้น สามารถสร้างเป็นแฟ้มต้นแบบให้สามารถใช้ได้กับข้อมูลชนิดต่างๆที่กำหนดได้ดังนี้

```
// File: stack.h
// Definition for a template class stack
// Structure:    A stack consists of a collection of elements
//               that are all of the same type , stackElType.
```

```
//              The elements of a stack are ordered according
//              to when they were placed on the stack. Only
//              the element that was last inserted onto the
//              stack can be removed or examined. New elements
//              are inserted at the top of the stack. Space for
//              maxSize elements is allocated, default is 100.
# ifndef  STACK_H
# define  STACK_H
template <class stackElType, int maxSize = 100>
class stack
{
        public:
            // Member functions
            // constructor to create an empty stack
            stack();

            // Push an element onto the stack
            bool push
              (const stackElType& x);          // IN:  item to push onto stack

            // Pop an element off the stack
            bool pop
              (stackElType& x);                // OUT:  Element popped from stack

            // Retrieve top element from stack without popping
            bool peek
              (stackElType& x) const;          // OUT:  Value at top of stack
```

```
        // Test to see if stack is empty
        bool isEmpty()  const;
        // Test to see if stack is full
        bool isFull()  const;
    private:
        // The data members are explained in Section 11.9
        int top;                        // index of element at top of stack
        stackElType item[maxSize];// array storage for stack
};
#endif  //STACK_H
```

การกำหนดต้นแบบของ Class stack นี้มีการกำหนดชนิดของข้อมูลได้ตามผู้ใช้ สำหรับเนื้อที่ในการจัดเก็บข้อมูลนั้นกำหนดไว้สูงสุดเท่ากับ 100 ค่า สำหรับการสร้างฟังก์ชันเพื่อใช้งาน เราสามารถสร้างไว้ในชื่อแฟ้ม stack.cpp ดังรายละเอียดดังนี้

```
// Implementation of template class stack using an array
// File: stack.cpp
// Implementation of template class stack
# include  "stack.h"

// constructor to create an empty stack
template <class stackElType, int maxSize>
stack<stackElType, maxSize>::stack ()
{
   top = -1;                        // A value of –1 indicates empty
}
```

```
// Push an element onto the stack
// Pre:  The element x is defined.
// Post: If the stack is not full, the item is pushed onto
//       the stack and true is returned. Otherwise, the stack
//       is unchanged and false is returned.
template <class stackElType, int maxSize>
bool stack<stackElType, maxSize>::push
   (const stackElType& x)          // IN:  item to push onto stack
{
   bool success;                          // flag:  true indicates successful push
   if (top < maxSize-1)             // If there is room
   {
      top++;                        // increment top
      items[top] = x;               // insert x
      success = true;                    // indicate success
   }
   else
      success = false;                    // no room, indicate failure
   return success;
} // end push

// Pop an element off the stack
// Pre:  none
// Post: If the stack is not empty, the value at the top of
//       the stack is removed, its value is placed in x, and
//       true is returned. If the stack is empty, x is not
//       defined and false is returned.
```

```
template <class stackElType, int maxSize>
bool stack<stackElType, maxSize>::pop
    (stackElType& x)                    // OUT:  Element popped from stack
{
    bool success;                               // flag:  true indicates successful pop
    if (top = 0)                        // if not empty
    {
        x = items[top];                 // remove top element
        top--;                          // decrement top
        success = true;                         // indicate success
    }
    else
        success = false;                        // indicate failure
    return success;
} // end pop

// Access top element from stack without popping
// Pre:  none
// Post:  If the stack is not empty, the value at the top is
//        copied into x and true is returned. If the stack is
//        empty, x is not defined and false is returned. In either
//        case, the stack is not changed.
template <class stackElType, int maxSize>
bool stack<stackElType, maxSize>::peek
    (stackElType& x)  const             // OUT:  Value at top of stack
{
    bool success;                       // flag:  true indicates successful get
```

```
    if (top >= 0)                    // if not empty
    {
        x = item[top];               // retrieve top element, do not pop
        success = true;              // indicate success
    }
    else
        success = false;             // indicate failure
    return success;
} // end peek

// Test to see if stack is empty
// Pre:  none
// Post:  Returns true if the stack is empty: otherwise,
//        returns false.
template <class stackElType, int maxSize>
bool stack<stackElType, maxSize>::isEmpty()  const
{
    return top < 0;
}

// Test to see if stack is full
// Pre:  none
// Post:  Returns true if the stack is full: otherwise,
//        returns false.
template <class stackElType, int maxSize>
bool stack<stackElType, maxSize>::isFull()  const
{
```

```
    return top >= maxSize-1;
}
```

เมื่อเรากำหนด Template class stack เรียบร้อยแล้ว เราสามารถกำหนดออบเจ็กต์ที่เป็นสมาชิกในคลาสนี้ได้ สำหรับตัวอย่างต่อไปนี้เป็นการนำข้อมูลที่เป็น string มาปฏิบัติการในโครงสร้างของสแตก โดยจัดการนำข้อความต่างๆเก็บในสแตกเป็นลำดับ และนำข้อความต่างๆออกมาเป็นลำดับเช่นกัน ผลของการทำงานจะเป็นส่วนกลับของกันและกัน ดังนี้

```
// Using a stack of characters
// File : stackText.cpp
// Use a stack to store strings and display them in reverse order

# include "stack.h"                    // Uses stack template class
# include <string>
# include <iostream>
using namespace std;

typedef stack<string, 20> stringStack;

int fillStack(stringStack& s);
void displayStack(stringStack& s);

int mian()
{
    // Local data
    stringStack s;
    // Read data into the stack.
```

```
    fillStack(s);
    // Display the stack contents
    displayStack(s);
    return 0;
}
//      Reads data characters and pushes them onto stack s.
//      Pre  : s is an empty stack.
//      Post : s contains the strings read in reverse order.
//      Returns the number of Strings read not counting the sentinel.
int fillStack(stringStack& s)           // OUT: stack to fill
{
    // Local data
    string nextStr;                     // next string
    int numString;                      // count of strings read
    const string sentinel = "****";     // sentinel string

    // Read and push strings onto stack until done.
    numString = 0;
    cout << "Enter next string or" << sentinel << ">";
    cin >> nextStr;
    while ((nextStr != sentinel) && (!s.isFull()))
    {
        s.push (nextStr);                       // Push next string on stack S.
        numString++;
        cout << "Enter next string or" << sentinel << ">";
        cin >> nextStr;
    }
```

```
    return numStrings;
} // end fillStack
//        Pops each string from stack s and displays it.
//        Pre : Stack s is defined.
//        Post : Stack s is empty and all strings are displayed.
void displayStack(stringStack& s)      // IN:  stack to display
{
    // Local data
    string nextStr;

    // Pop and display strings until stack is empty.
    while (!s.isEmpty())
    {
        s.pop(nextStr);                // Pop next string off.
        cout << nextStr << endl;
    }
}   // end displayStack
```

เมื่อนำโปรแกรมนี้ไปทำงานและมีการป้อนข้อมูลดังนี้

```
Enter next string or "***> Here
Enter next string or "***> are
Enter next string or "***> strings!
Enter next string or "***> ***
```

ผลของการทำงานจะแสดงข้อความในลักษณะย้อนกลับ ดังนี้

```
strings!
are
Here.
```

สรุป

1. Template Classes เป็นการสร้างคลาสโดยให้ผู้ใช้สามารถกระทำการปฏิบัติการกับชนิดของข้อมูลที่กำหนดโดยผู้ใช้งานได้
2. รูปแบบของการกำหนด Template Classes

```
template <class T>
class class-name {
public:
```

- List of class variables, types, constants, etc. (if any) that may be accessed by name from outside the class
- Prototype for each function that may be accessed by name from outside the class

```
. . .
private:
```

- List of class variables, types, constants, etc., that are intended to be hidden from reference from outside the class
- Prototype for each function (if any) to be hiden from outside the class

```
. . .
};
```

โดย Class class-name คือ template class โดยมีพารามิเตอร์ T ซึ่ง T เป็นเนื้อที่ที่ผู้ใช้หรือระบบกำหนดขึ้นสำหรับเก็บชนิดข้อมูลที่ผู้ใช้กำหนด ต้องอยู่ในเครื่องหมาย < > เมื่อสังเกตให้ดีจะเห็นว่าสิ่งที่เปลี่ยนไปคือการกำหนด template <class T> ก่อนกำหนดต้นแบบหรือรายละเอียดของคลาสนั้นเอง

3. สำหรับการประกาศออบเจ็กต์ที่มีการใช้ Template Classes นี้จะมีการปรับเปลี่ยนโดยมีการส่งชนิดของข้อมูลเพื่อกระทำงานในคลาสมีรูปแบบดังนี้

การประกาศ Template Classes	
รูปแบบ:	class-name<type> an-object;
ตัวอย่าง	indexList<int> intList;

type ที่อยู่ในเครื่องหมาย < > จากรูปแบบการประกาศนี้อาจเป็นชนิดของข้อมูลอะไรก็ได้ที่ผู้ใช้ต้องการให้ปฏิบัติงานในคลาส ส่วน class-name เป็นชื่อของ Template class ที่กำหนดขึ้น สำหรับ an-object เป็นชื่อของออบเจ็กต์ที่สร้างขึ้นมาโดยจะมีความสอดคล้องกับชนิดของข้อมูลที่ได้ประกาศ ซึ่งเมื่อมีการเรียกใช้ฟังก์ชั่นทั้งหลายในคลาสจะมีการแทนชนิดของข้อมูลเพื่อให้สามารถใช้งานได้ตามที่ได้ตามที่ต้องการ

4. สแตก (stack) เป็นโครงสร้างข้อมูลที่มีการปฏิบัติงานในลักษณะของ LIFO (Last in-First out) เป็นการปฏิบัติงานที่สมาชิกตัวหลังสุดจะได้รับการบริการเป็นอันดับแรก และสมาชิกตัวแรกจะได้รับบริการเป็นลำดับสุดท้าย
5. การนำข้อมูลเก็บเข้าไปในสแตกเรียกว่าการ push โดยกระทำที่ปลายด้านใดด้านหนึ่ง การนำข้อมูลออกจากสแตกเรียกว่า การ pop โดยจะกระทำได้ครั้งละ 1 สมาชิกและปลายด้านเดียวกัน
6. โดยทั่วไปเราจะมีตัวแปร top ทำหน้าที่ในการชี้ข้อมูลตัวสุดท้ายของสแตก ทำให้ทราบว่าการนำสมาชิกออกจะกระทำที่ตำแหน่งที่ top ชี้อยู่ หรือการนำเข้าต้องกระทำหลังจากตำแหน่งที่ top ชี้อยู่

แบบฝึกหัด

1. จงเขียน Member Function ชื่อ popNextTop เพื่อปฏิบัติการกับนำสมาชิกก่อนหน้าของสมาชิกที่ top ชี้อยู่ออกจากสแตก
2. โครงสร้างสแตกแตกต่างจากโครงสร้างอาเรย์อย่างไร จงอธิบายพอเข้าใจ
3. บริษัทCT212 จำกัด ต้องการพัฒนาโปรแกรมเพื่อจัดการระบบการจ่ายเงินพนักงาน โดยมีรายละเอียดดังนี้ พนักงานแบ่งเป็น 2 ประเภท คือ

   ประเภทที่ 1 ลูกจ้างรายวัน มีการคิดค่าตอบแทนโดยนำจ่ายเงินเป็นรายสัปดาห์ โดยการนำอัตราค่าจ้างรายชั่วโมงคูณด้วยจำนวนชั่วโมงทำงานในสัปดาห์นั้น ลูกจ้างคนไหนทำงานเกินกว่า 40 ชั่วโมง ในสัปดาห์นั้นจะได้รับค่าทำงานล่วงเวลา โดยจำนวนชั่วโมงการทำงานที่เกินกว่า 40 ชั่วโมง จะได้รับค่าจ้างเป็น 2 เท่าของอัตราจ้างปกติ

   ประเภทที่ 2 พนักงานประจำจะได้รับเงินเดือนตามที่กำหนด แต่จะต้องถูกหักภาษีเงินได้ไว้ 5% และถ้าเงินเดือนต่ำกว่า 10,000 บาท จะได้รับเงินช่วยเหลือพิเศษเดือนละ 1000 บาท

   จงทำการวิเคราะห์และออกแบบโปรแกรมที่จะใช้งานนี้ โดยเขียนเป็นโปรแกรมภาษา C++ ที่ออกแบบโดยใช้แนวความคิดเชิงวัตถุ
4. จงเขียนโปรแกรมในการแปลงนิพจน์ Infix ให้เป็นนิพจน์ Postfix
   ตัวอย่างข้อมูลเข้าคือ A + B * C – D ผลลัพธ์คือ A B C * + D –
   โดยมีการเรียกใช้คลาสที่ชื่อว่า stack ทำงาน โปรแกรมจะทำการตรวจสอบตัวกระทำถ้าตัวกระทำใดทำงานก่อนจะมีการนำตัวกระทำไปไว้หลังตัวโอเปอร์แรนด์ทั้งสอง
5. จงเขียนสร้างคลาสที่ชื่อว่า employee ซึ่งมีการเก็บข้อมูลพื้นฐานประกอบด้วย รหัสพนักงาน (id) , อัตราค่าจ้างรายชั่วโมง(rate) และจำนวนชั่วโมงทำงาน (hours) มีกิจกรรมที่กระทำในคลาสประกอบด้วย
   5.1 totalgross เป็นการหารายได้ทั้งหมดที่พนักงานได้รับ
   5.2 searchid เป็นฟังก์ชันในการค้นหาข้อมูลของพนักงานโดยใช้รหัสพนักงานเป็นข้อมูลสืบค้น

5.3 searchMax เป็นฟังก์ชันในการแสดงข้อมูลของพนักงานที่ได้รับค่าจ้างหรือเงินเดือนมากที่สุด

โดยให้นักศึกษาออกแบบคลาสให้เหมาะสม และสามารถประกาศออบเจ็กต์ของ employee เพื่อทำงานตามฟังก์ชันต่างๆเหล่านี้ได้

6. เขียนโปรแกรมในการบวกเมตริกซ์ , ลบเมตริกซ์ , คูณเมตริกซ์ ใดๆ โดยการสร้างคลาสเมตริกซ์ใดๆขึ้นมา เพื่อให้สามารถทำงานกับกิจกรรมต่างๆตามที่กำหนดได้
7. เขียนโปรแกรมในการ Union , Interaction เซตใดๆ โดยสร้างคลาสเซตใดๆขึ้นมาเพื่อสามารถปฏิบัติงานตามที่ต้องการได้ โดยออกแบบตามความเหมาะสม