ปฏิบัติการที่ 10 Pointer with class & Virtual function

บทเรียนย่อย

- การสร้างวัตถุแบบ pointer
- Dynamic Variable
- Dynamic Arrays
- Classes and Pointers
- Virtual Functions

วัตถุประสงค์การเรียนรู้

- เรียนรู้ และทำความเข้าใจกับวัตถุ (Object) แบบ Pointer
- เรียนรู้ และทำความเข้าใจ Dynamic Variable
- เรียนรู้ และทำความเข้าใจ Dynamic Arrays
- เรียนรู้ และทำความเข้าใจการใช้งาน Class กับ Pointer
- เรียนรู้ และทำความเข้าใจการสืบทอดคุณสมบัติ และ Virtual Function

---

ให้นิสิตสร้าง Directory ชื่อว่า Lab10 สำหรับทดลองปฏิบัติการดังต่อไปนี้

ตอนที่ 1 การสร้างวัตถุแบบ pointer

วัตถุที่สร้างขึ้นจากคลาสใดๆ สามารถกำหนดให้เป็นตัวแปรแบบ pointer ได้เช่นเดียวกับตัวแปรของชนิดอื่น โดยมีรูปแบบดังนี้

className *objectname;

ตัวอย่างการใช้งาน

```
class studenType   {
    public:
       string name;
       double gpa;
       print();
};
studentType  student;
studentType  *studentPrt;
```

จากด้านบน student เป็นวัตถุที่สร้างจากคลาส studentType และ studentPrt เป็นตัวแปร pointer สำหรับชี้ไปยังวัตถุที่สร้างจากคลาส studentType

studentPrt = &student;

หลังจากการทำงานนี้ จะทำให้ตัวแปร pointer studentPrt เก็บค่าแอดเดรสของวัตถุ student ในคำสั่งต่อไปเป็นการกำหนดค่า 3.9 ให้กับแอตทริบิวต์ gpa ในวัตถุ student

(*studentPrt).gpa = 3.9;

สังเกตว่ามีวงเล็บครอบ *studentPrt ก่อน . (dot) เนื่องจากโดยปกติตัวดำเนินการ dot นั้นมีความสำคัญมากกว่าจะทำงานก่อนตัวดำเนินการ * จึงจำเป็นต้องใส่วงเล็บครอบ โดยตัวดำเนินการอีกตัวที่สามารถใช้งานในการเข้าถึงสมาชิกในคลาสแทนรูปแบบดังกล่าวได้คือ -> มีรูปแบบดังนี้

pointerVariableName->classMemberName

เช่น

(*studentPrt).gpa = 3.9;

มีผลเท่ากับ

studentPrt->gpa = 3.9;

ให้นิสิตศึกษาและทดสอบรันโปรแกรมในโฟลเดอร์ ex01_pointer

ตอนที่ 2 Dynamic Variable

จากหัวข้อที่ผ่านมา การประกาศตัวแปร pointer นั้นเป็นการประกาศที่ตัวที่เก็บแอดเดรสของวัตถุอื่นที่ถูกจองพื้นที่ในหน่วยความจำแล้วเท่านั้น เพียงลำพังตัวแปร pointer เองนั้นไม่สามารถกำหนดข้อมูล หรือใช้งานได้ เนื่องจากไม่มีการจองพื้นหน่วยความจำ ซึ่งในหัวข้อนี้จะเป็นหัวข้อสำหรับตัวดำเนินการ (operator) สำหรับการจองพื้นที่ในหน่วยความจำ และคืนพื้นที่ในหน่วยความจำของตัวแปร pointer ซึ่งตัวแปรที่สร้างจากการจองพื้นที่หน่วยความจำแบบนี้เรียกว่า dynamic variables

ตัวดำเนินการ (Operator) new

ตัวดำเนินการ new สามารถใช้ได้สองประเภทคือ การจองพื้นที่ให้กับตัวแปรเดี่ยวๆ และการจองพื้นที่ให้กับตัวแปรอาร์เรย์ (Array) ซึ่งมีรูปแบบดังนี้

new dataType; //to allocate a single variable

new dataType[intExp]; //to allocate an array of variables

โดยกำหนดให้ intExp คือจำนวนอาร์เรย์ที่ต้องการจองพื้นที่ ซึ่งต้องเป็นจำนวนเต็มบวก

ตัวดำเนินการ new จะทำการจองพื้นที่หน่วยความจำตามชนิดตัวแปรที่กำหนด (เช่นเดียวกันกับตัวแปรเป็นวัตถุที่สร้างจากคลาส) จากนั้นทำการส่งแอดเดรสกลับมายังตัวแปร pointer

ตัวอย่างการใช้งาน

```
int  *p;
char  *ch
studenType  *student;
studenType  *student2;
p  =  new  int;
ch  =  new  char[20];
student  =  new  studentType;
student2  =  new  studentType("Somchai");
```

สังเกตวัตถุ student2 เป็นการจองพื้นที่หน่วยความจำขนาดเท่ากับคลาส studentType โดยทำการเรียกใช้ constructor แบบส่งพารามิเตอร์ไปด้วย ซึ่งในส่วนของ student นั้นเป็นการเรียกใช้งาน default constructor

ตัวดำเนินการ (Operator) delete

ตัวดำเนินการ delete เป็นตัวดำเนินการสำหรับทำลาย dynamic variables นั้นคือเป็นการคืนพื้นที่หน่วยความจำให้แก่ระบบ มีรูปแบบดังนี้

delete pointerVariables; //to deallocate a single dynamic variable

delete [] pointerVariables; //to deallocate a dynamically created array

ตัวอย่างการใช้งาน

```
delete  p;
delete  []  ch;
delete  student;
```

ให้นิสิตศึกษา และรันโปรแกรมเพื่อดูผลลัพธ์ ไฟล์ ex02_allocate.cpp

ตอนที่ 3 Dynamic Arrays

จากเดิมการประกาศตัวแปรอเรย์นั้นจำเป็นต้องกำหนดขนาดของอาร์เรย์ไว้ล่วงหน้าก่อนใช้งานตัวแปรดังกล่าว นั้นได้
เนื่องจากระบบจะทำการจองพื้นที่หน่วยความจำให้แก่ตัวแปรอาร์เรย์ไว้ล่วงหน้าก่อนการรันโปรแกรม
ในหัวข้อนี้เราสามารถใช้ตัวดำเนินการสำหรับจัดการหน่วยความจำของตัวแปรอาร์เรย์แบบ dynamic array
ซึ่งการประกาศตัวแปรแบบนี้ระบบจะทำการจองพื้นที่หน่วยความจำให้กับตัวแปรในขณะที่ทำงานจริง (runtime)
การใช้งานอาร์เรย์ที่เป็นแบบ dynamic array มีรูปแบบคือ

```
int  *p;
p = new int[10];
```

Dynamic Two-Dimensional Array

หัวข้อข้างต้นเป็นการประกาศอาร์เรย์หนึ่งมิติเป็นแบบ dynamic array แล้ว
ในหัวข้อนี้ยกตัวอย่างการประกาศตัวแปรอเรย์สองมิติในแบบ dynamic array ซึ่งทำได้หลายรูปแบบคือ
พิจารณาตัวอย่างต่อไปนี้

```
int  *board[4];
```

ด้านบนเป็นการประกาศตัวแปร board ให้เป็นอาร์เรย์ของ pointer 4 ตัวด้วยกัน ซึ่งจะได้ board[0], board[1], board[2] และ board[3] เป็น pointer ดังนั้นเราสามารถใช้ pointer ทั้ง 4 นี้
เป็นจำนวนแถวของ อาร์เรย์สองมิติได้ สมมติเราต้องการให้แต่ละแถวนั้นมีจำนวนคอลัมน์เท่ากับ 6
สามารถใช้ลูป for ในการกำหนดจองพื้นที่หน่วยความจำได้ดังนี้

```
for (int row = 0; row < 4; row++)   {
   board[row]  = new  int[6];
}
```

ตัวดำเนินการ new int[6] เป็นการสร้างอาร์เรย์ความยาวหกตัว (จองพื้นที่หน่วยความจำ) และคืนค่าแอดเดรสกลับมายังตัวแปร board[row] ซึ่งจากซอร์สโค้ดนี้จะได้ตัวแปรอาร์เรย์สองมิติที่มีขนาด 4 แถว 6 คอลัมน์ จากตัวอย่างนี้เมื่อเราเปลี่ยนแปลงค่าจำนวนคอลัมน์จาก 6 เป็นค่าอื่น เช่น 10 ผลลัพธ์ก็จะได้อาร์เรย์สองมิติที่มีขนาดต่างออกไป นั้นคือเราสามารถเปลี่ยนแปลงจำนวนคอลัมน์ของตัวแปร board ได้ตอนโปรแกรมทำงาน แต่จะเห็นว่าจำนวนแถวจะถูกกำหนดอยู่ก่อนหน้าแล้ว ซึ่งเราสามารถให้เป็น dynamic two dimensional array อย่างสมบูรณ์ได้คือ

```
int  **board;
```

จากนั้นกำหนดขนาดอเรย์ได้ภายหลังได้จาก

```
board  =  new  int*  [10];
```

จากตัวอย่างเป็นการกำหนดให้ตัวแปร board เป็นตัวแปรอาร์เรย์ที่มีจำนวนแถวเท่ากับ 10 แถว
จากนั้น สามารถกำหนดจำนวนคอลัมน์ได้จาก

```
for (int row = 0; row < 10; row++)
        board[row]  =  new  int[15];
```

จากซอร์สโค้ดด้านบนเราจะได้ตัวแปร board เป็นตัวแปรอาร์เรย์ขนาด 10 แถว 15 คอลัมน์ซึ่งเป็นตัว แปรแบบ dynamic two-dimensional array

ให้นิสิตศึกษา และทดลองรันโปรแกรมจากไฟล์ ex03_dynamic.cpp

ตอนที่ 4 Classes and Pointers

สมาชิกภายในคลาสที่เป็นตัวแปรนั้น สามารถเป็นตัวแปรแบบ pointer ได้ ดังตัวอย่างต่อไปนี้

```
class ptrMemberVarType   {
    private:
        int  n;
        int  *p;
};
```

โหากในการทำงานของตัวแปรดังกล่าว เป็นแบบ dynamic variable เมื่อวัตถุที่สร้างจากคลาสดังกล่าวนี้ สิ้นสุดการทำงาน หรือถูกทำลายลง จำเป็นต้องคืนพื้นที่หน่วยความจำที่จองไว้ให้กับตัวแปรแบบ pointer นั้นด้วย การคืนพื้นที่หน่วยความจำนั้นสามารถทำได้ภายใน destructor ของการกำหนดการทำงานของคลาส (implement) โดยใช้ตัวดำเนินการ delete หากเราไม่ใช้ตัวดำเนินการ delete ในการคืนพื้นที่หน่วยความจำนั้น จะทำให้เมื่อวัตถุถูกทำลายไปแล้ว พื้นที่ของตัวแปรแบบ pointer จะยังคงถูกจองพื้นที่อยู่ ไม่สามารถนำมาใช้งานในระบบได้อีกจนกว่าจะจบโปรแกรม

ตัวอย่างการเขียน destructor สำหรับคืนพื้นที่หน่วยความจำ

```
ptrMemeberVarType::~ptrMemberVarType()   {
    delete [] p;
 }
```

และต้องไม่ลืมที่กำหนดไว้ยังโครงสร้างคลาสด้วยเช่นกัน

```
class ptrMemberVarType   {
    public:
      ~ptrMemberVarType();
    private:
  int  n;
      int  *p;
};
```

Assignment Operator

ในส่วนนี้จะอธิบายถึงข้อจำกัดบางส่วนในการกำหนดค่าให้กับวัตถุที่สร้างจากคลาสเดียวกัน จากตัวอย่างต่อไปนี้

```
ptrMemberVarType  objectOne;
```

ptrMemberVarType objectTwo;
objectOne = objectTwo;

จากการทำงานด้านบน เป็นการกำหนดค่าแอตทริบิวต์ของ objectOne ทุกตัวมีค่าเท่ากับแอตทริบิวต์ ของ ObjectTwo ดังนั้นเมื่อภายในคลาส ptrMemberVarType มีสมาชิกที่เป็นตัวแปรแบบ pointer จาก ตัวอย่างที่ผ่านมาคือ p จะทำให้ตัวแปร p ภายใน objectOne นั้นจะเก็บค่าแอดเดรสเดียวกันกับ objectTwo ซึ่งจะส่งผลให้เมื่อเปลี่ยนแปลงค่าที่แอดเดรสที่ตัวแปร p ใน objectTwo เก็บไว้ จะทำให้ค่าที่ตัว แปร p ใน objectOne เปลี่ยนแปลงตามไปด้วย

หากต้องการให้การกำหนดค่าของวัตถุเป็นการคัดลอกที่แยกกันนั้นสามารถทำได้โดยหลายวิธีหนึ่งในนั้นคือการสร้าง copy constructor ขึ้นมา คือการกำหนดให้มี constructor รับตัวแปรที่เป็นวัตถุที่สร้างจากคลาสเดียวกันเข้ามาเพื่อกำหนดค่าแอตทริบิวต์ที่เรากำหนดเอง ตามคำสั่งตัวอย่างต่อไปนี้

ptrMemberVarType objectThree(objectTwo);

ให้นิสิตศึกษาการเขียนโปรแกรม copy constructor และทดสอบรันโปรแกรม ภายในโฟลเดอร์ ex04_copyConstructor

ตอนที่ 5 Virtual Function

ใน C++ นั้นอนุญาตให้ผู้ใช้สามารถส่งพารามิเตอร์ที่เป็นวัตถุของคลาสลูก (derived class) ไปยัง function กำหนดชนิดของพารามิเตอร์ (formal parameter) ที่เป็นคลาสแม่ (base class) ได้ ตัวอย่างเช่น

```
class baseType
{
public:
   void print();
private:
   string name;
};
class derivedType: public baseType
{
};
```

สมมติให้เขียนรายละเอียดคลาสเรียบร้อย (Implement) และภายในไฟล์อื่นสร้างฟังก์ชันการใช้งานคลาส baseType ดังนี้

```
void callPrint(baseType& b)
{
b.print();
}
```

C++ อนุญาตให้สามารถส่งวัตถุที่สร้างจากคลาส derivedType ส่งค่าไปยังฟังก์ชัน callPrint ได้ เช่น

```
baseType bs;
derivedType dv;
callPrint(bs);
callPrint(dv);
```

ให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex01_virtual_01 และสังเกตผลการทดสอบ จะเห็นว่าสามารถเรียกใช้งานฟังก์ชัน callPrint โดยการส่งวัตถุที่สร้างจากคลาส petType และวัตถุที่สร้างจากคลาส dogType แต่ให้นิสิตสังเกตว่าภายในคลาส dogType นั้นมีการ overriding method ที่ชื่อว่า print ดังนั้น ภายในคลาส petType และ dogType ต่างมีสมาชิกที่เป็นฟังก์ชันที่ชื่อ print ของตนเอง ให้นิสิตสังเกตและตอบคำถามต่อไปนี้

รหัสนิสิต..... ชื่อ สกุล ..... กลุ่ม .....

1. เมื่อส่งวัตถุ (dog) ที่สร้างจากคลาส dogType ไปยังฟังก์ชัน callPrint ซึ่งภายในเรียกใช้ method print เป็นการเรียกใช้ print ของคลาสใด และข้อมูลของวัตถุใด

.....

.....

.....

การทำงานของ method print ภายในฟังก์ชัน callPrint นั้นจะถูกกำหนดการทำงานว่าทำงานแบบอะไรบ้าง ในตอบคอมไพล์โปรแกรมแล้ว ทำให้ในตอนคอมไพล์โปรแกรม callPrint จะเป็นเรียกการทำงานของคลาส petType หรือเรียกว่า compile-time binding (static binding) การทำงานดังกล่าวนี้ C++สามารถแก้ไขปัญหาให้สามารถเรียกใช้งาน method ที่ถูกต้องได้โดยการใช้กลไลที่เรียกว่า virtual functionsซึ่งจะทำให้การเรียกใช้ method ภายในคลาสเกิดขึ้นตอนรันโปรแกรม หรือเรียกว่า run-time binding (dynamic binding) โดยการเขียน virtual function สามารถทำได้โดยใช้ คำเฉพาะ virtual ไว้หน้า method ที่ต้องการภายในคลาสแม่ (base class) เช่น

```
class petType
{
public:
   virtual void print();
   petType(string n = "");
private:
   string name;
};
class dogType: public petType
{
public:
   void print();
   dogType(string n = "", string b = "");
private:
   string breed;
};
```

สังเกต method print ภายในคลาส petType เท่านั้น ให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex01_virtual_02 และตอบคำถามดังนี้

2. เมื่อส่งวัตถุ (dog) ที่สร้างจากคลาส dogType ไปยังฟังก์ชัน callPrint ซึ่งภายในเรียกใช้ method print เป็นการเรียกใช้ print ของคลาสใด และข้อมูลของวัตถุใด

.....

.....

ให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex01_virtual_03 เพื่อเห็นการใช้งาน dynamic variables กับ virtual function และการส่งค่าวัตถุไปยังฟังก์ชันและให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex01_virtual_04 เพื่อเห็นการใช้งาน virtual function กับการส่งพารามิเตอร์แบบ pass by value และตอบคำถามต่อไปนี้

3. เมื่อส่งวัตถุ (dog) ที่สร้างจากคลาส dogType ไปยังฟังก์ชัน callPrint ซึ่งภายในเรียกใช้ method print เป็นการเรียกใช้ print ของคลาสใด และข้อมูลของวัตถุใด

.....

.....

ให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex01_virtual_05 เพื่อเห็นการทำงานเมื่อกำหนดค่าแอดเดรสของวัตถุที่สร้างจากคลาสลูกให้กับวัตถุที่สร้างจากคลาสแม่

ตอนที่ 6 Abstract Class และ Pure Virtual Function

หากการสร้างคลาสแม่ (base class) ขึ้นมาเพื่อทำการรวมลักษณะ หรือการทำงานของหลายๆ คลาส เพื่อสร้างความสะดวกเมื่อสืบทอดคุณสมบัติไปยังคลาสลูก (derived class)
หลายคลาสซึ่งไม่จำเป็นต้องเขียนโปรแกรมการทำงานซ้ำซ้อนกัน เช่น คลาส shape เป็นคลาสแม่ สืบทอดไปยังคลาส rectangle และ คลาส circle เพื่อลดการเขียนโปรแกรมซ้ าซ้อนของข้อมูลความยาวด้าน หรือรัศมี
หรือรูปแบบการทำงานคล้ายคลึงกัน เช่น การหาพื้นที่ หรือการหาเส้นรอบรูปเลขาคณิต แต่อย่างไรก็ตาม อาจมีบาง method ที่มีการทำงานจริงเกิดขึ้นเฉพาะคลาสลูก (ภายในคลาสแม่ ไม่สามารถระบุการทำงานของ method นั้นได้ เพราะไม่สามารถทำให้ทำงานได้จริง) เช่น method draw() ที่ในคลาสแม่ คือ คลาส shape
ไม่สามารถวาดรูปได้ว่าเป็นรูปแบบใดแน่นอน แต่ภายในคลาสลูกทุกคลาสที่สืบทอดมาจำเป็นต้องมี
โดยการออกแบบคลาสลักษณะนี้ นิยมใช้ความสามารถของ Virtual functions ในการออกแบบ ซึ่งจะกำหนดให้เป็น Pure Virtual Functions นั้นคือการกำหนด method ในคลาสแม่เป็น Virtual Functions แต่ไม่มีการทำงานใดๆ (ไม่ต้อง Implement) โดยมีรูปแบบตัวอย่างดังนี้

```
virtual void draw() = 0;
virtual void move(double x, double y) = 0;
```

โดย method ที่เป็น Pure Virtual Functions นั้นไม่จำเป็นเขียนรายละเอียดการทำงาน (Implement) ในคลาสแม่ และไม่สามารถเรียกใช้งานได้ในกรณีที่สร้างวัตถุจากคลาสแม่นี้ แต่มีข้อกำหนดว่า คลาสลูกใดๆ
ที่สืบทอดไปนั้นจะต้องทำการ overriding method ที่เป็น Pure Virtual Functions
เสมอและเมื่อภายในคลาสแม่ใดที่มีสมาชิกที่เป็น function (method) เป็น Pure Virtual Functions
หนึ่งหรือมากกว่า จะเรียกคลาสนั้นว่าเป็น Abstract class

ให้นิสิตศึกษา และรันโปรแกรมภายในโฟลเดอร์ ex02_abstract เพื่อทำความเข้าใจการสร้าง Abstract class

แบบฝึกหัด

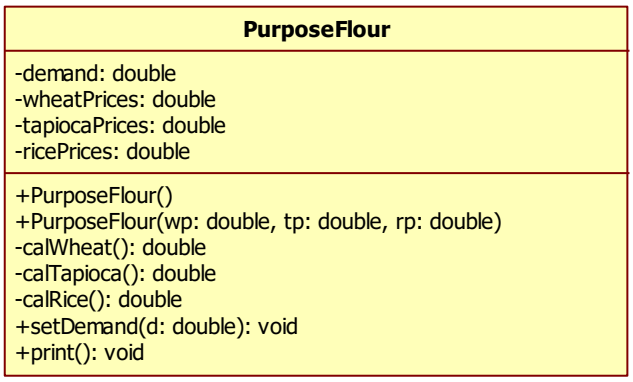
1. ให้นิสิตออกแบบ และเขียนคลาส PurposeFlour เป็นคลาสเกี่ยวกับส่วนผสมของแป้งทอดกรอบ ที่มีรายละเอียดดังนี้
    - ส่วนประกอบของแป้งทอดกรอบ คือ
        - แป้งสาลี (wheat) 50%
        - แป้งมัน (tapioca) 25%
        - แป้งข้าวเจ้า (rice) 25%
    - มีแอตทริบิวต์สำหรับเก็บจำนวนแป้งทอดกรอบ (demand) ที่ต้องการผสมออกมาได้ (กก.)
    - มีแอตทริบิวต์สำหรับเก็บราคาแป้งสาลีต่อกิโลกรัม (wheatPrices)
    - มีแอตทริบิวต์สำหรับเก็บราคาแป้งมันต่อกิโลกรัม (tapiocaPrices)
    - มีแอตทริบิวต์สำหรับเก็บราคาแป้งข้าวเจ้าต่อกิโลกรัม (ricePrices)
    - มี constructor สำหรับรับค่าราคาแป้งส่วนผสมทั้งสามชนิด

- มีเมธอดสำหรับรับค่าจำนวนแป้งทอดกรอบที่ต้องการผสมเป็นกิโลกรัม (methotd ชื่อ setDemand)
- มีเมธอดสำหรับคำนวณจำนวนแป้งสาลีที่ต้องใช้เพื่อให้ได้จำนวนแป้งทอดกรอบที่ต้องการ (methotd ชื่อ calWheat)
- มีเมธอดสำหรับคำนวณจำนวนแป้งมันที่ต้องใช้เพื่อให้ได้จำนวนแป้งทอดกรอบที่ต้องการ (methotd ชื่อ calTapioca)
- มีเมธอดสำหรับคำนวณจำนวนแป้งข้าวเจ้าที่ต้องใช้เพื่อให้ได้จำนวนแป้งทอดกรอบที่ต้องการ (methotd ชื่อ calRice)
- มีเมธอดสำหรับพิมพ์ข้อมูลต่างๆ ดังนี้ จำนวนแป้งทอดกรอบที่ต้องการได้ จำนวนส่วนผสมของแป้งแต่ละชนิดที่ต้องใช้ และราคาจากจำนวนที่ต้องใช้นั้น ออกทางหน้าจอ (methotd ชื่อ print)

Note:

การหาจำนวนแป้ง ให้คิดคำนวณแบบเทียบบัญญัติไตรยางค์ และคำนวณออกมาเลขทศนิยมแล้วค่อยนำไปคูณกับค่าตัวเลขที่ต้องการ

กำหนด Class Diagram ดังนี้



จากนั้นให้นิสิตเขียนโปรแกรมเพื่อทดสอบการใช้งาน โดยต้องสร้างวัตถุขึ้นจากคลาส PurposeFlour โดยกำหนดให้เป็น dynamic variables

โดยโปรแกรมจะทำรับ input ค่าจำนวนแป้งทอดกรอบที่ต้องการ และราคาของแป้งสาลี แป้งมัน แป้งข้าวเจ้าตามลำดับ แล้วให้เขียนโปรแกรมคำนวนจำนวนแป้งแต่ละชนิดและค่าใช้จ่ายในการซื้อทั้งหมด

ตัวอย่าง

Input ต้องการแป้งทอดกรอบจำนวน 200 kg มีราคา แป้งสาลี แป้งมัน แป้งข้าวเจ้า คือ 15 20 10 บาทตามลำดับ

Output คือ ใช้แป้งสาลีจำนวน 100 kg แป้งมัน 50 kg แป้งข้าวเจ้า 50 kg และค่าใช้จ่ายทั้งหมด 3000 บาท

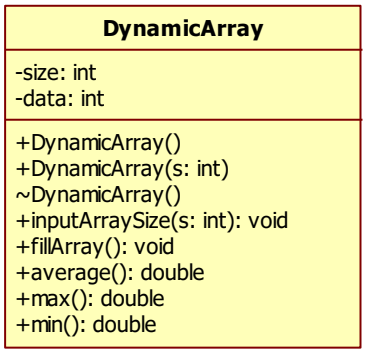
รหัสนิสิต..... ชื่อ สกุล ..... กลุ่ม .....

2. ให้นิสิตออกแบบ และเขียนคลาส DynamicArray ที่มีรายละเอียดดังนี้

- มีแอตทริบิวต์ที่ใช้เก็บค่าจำนวนความยาวของอาร์เรย์ (ตัวแปร size)
- มีแอตทริบิวต์ที่เป็นตัวแปร dynamic array ชนิด int (ตัวแปร data)
- มีเมธอดสำหรับรับค่าความยาวของ array จากทางหน้าจอ (methotd ชื่อ inputArraySize)
- มีเมธอดสำหรับรับค่าของ int แต่ละตัวจากทางหน้าจอ (methotd ชื่อ fillArray) ตามจำนวนความยาว รับมาจากเมธอดก่อนหน้า (inputArraySize)
- มีเมธอดสำหรับคืนค่าเฉลี่ยของค่าใน array ทั้งหมด (methotd ชื่อ average)
- มีเมธอดสำหรับคืนค่ามากที่สุดใน array ออกมา (methotd ชื่อ max)
- มีเมธอดสำหรับคืนค่าน้อยที่สุดใน array ออกมา (methotd ชื่อ min)

จากนั้นให้นิสิตเขียนโปรแกรม main.cpp เพื่อทดสอบการใช้งานในทุกเมธอด

กำหนด Class Diagram ดังนี้



ตัวอย่าง

Input

บรรทัดแรก ความยาวอาร์เรย์

บรรทัดที่สองจนถึงบรรทัดสุดท้าย ข้อมูลของอารเรย์ตามความยาวอาร์เรย์

Output

บรรทัดแรก ค่าเฉลี่ย (แสดงเลขทศนิยม 2 ตำแหน่ง)

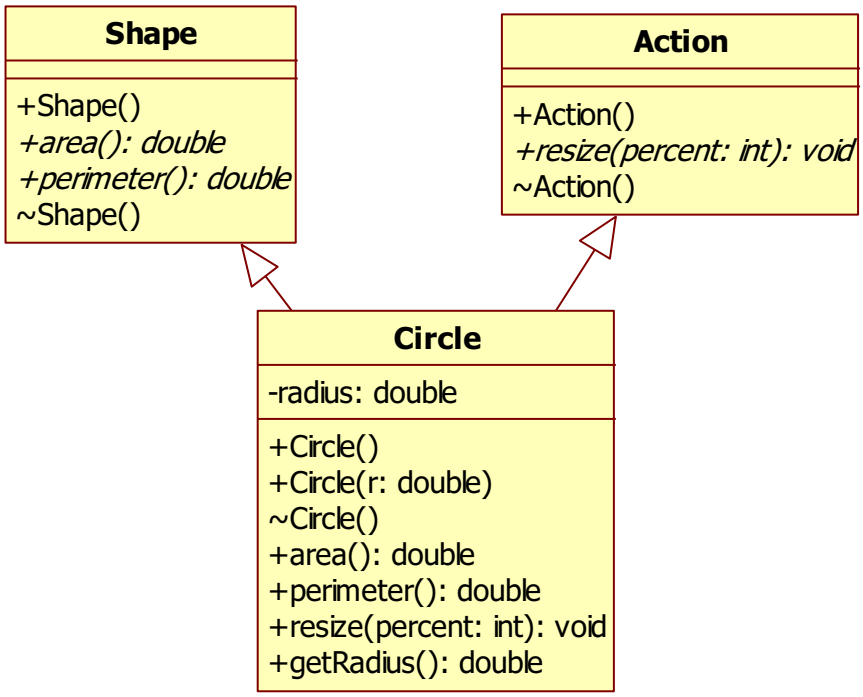
บรรทัดที่สอง ค่าต่ำสุด (แสดงเลขจำนวนเต็ม)

บรรทัดที่สาม ค่าสูงสุด (แสดงเลขจำนวนเต็ม)

Input	Output
10 1 2 3 4 5 6 7	5.80 1 11

9 10 11	

3.คลาสรูปร่าง (Shape) เป็นคลาสนามธรรม (Abstract Class) ที่ประกอบด้วยเมธอดสำหรับหาพื้นที่ (area) และเส้นรอบรูป (perimeter) ให้นิสิตสร้างคลาสวงกลม (Circle) ที่สืบทอดมาจากคลาสรูปร่าง และสืบทอดมาจากคลาส Action ที่ประกอบด้วยเมธอดสำหรับสำหรับเปลี่ยนแปลงขนาด (resize) ที่ทำการลดหรือเพิ่มขนาดในลักษณะเปอร์เซ็นต์ของรูปร่างเดิม



หลังจากนิสิตเขียนคลาสต่างๆ เรียบร้อยแล้วให้ใช้ main program ต่อไปนี้ในการรันโปรแกรม

```
int main()
{
    Shape *s = new Circle(10);
    cout << "Circle has radius = " << ((Circle*)s)->getRadius() << endl;
    cout << "area : " << s->area() << endl;
    cout << "perimeter : " << s->perimeter() << endl;

    cout << "----------------------" << endl;
    cout << "resize 50%" << endl;
    ((Circle*)s)->resize(50);
```

```
    cout << "----------------------" << endl;

    cout << "Circle has radius = " << ((Circle*)s)->getRadius() << endl;
    cout << "area : " << s->area() << endl;
    cout << "perimeter : " << s->perimeter() << endl;

    return 0;
}
```

โดยผลลัพธ์ของโปรแกรมควรจะเป็นดังนี้

```
Circle has radius = 10
area : 314
perimeter : 62.8
----------------------
resize 50%
----------------------
Circle has radius = 5
area : 78.5
     perimeter : 31.4
```