UEE1303 S18: Object-Oriented Programming

Advanced Topics of Class

What you will learn from Lab 5

In this laboratory session you will:

1. learn the advance topics of object-oriented programming using class.
2. learn how to use operator overloading, which are important functionality provided by C++.

LAB 5-1: STATIC MEMBER

✓ static member can be taken as a global member for this class and all objects own the same copy (or value) of the member.

```
// FILE::lab5-1.cpp
#include <iostream>
class Point2D
{
public:
	Point2D();
	void assignPoint2D(int x, int y);
	void displayPoint2D();
	static void setValue(double v);
	// only static member function can access static member
private:
	int x;
	int y;
	static const double limit = 10.0; // const static member can be init.
	static double value; // indicates that all object's value are the same
};
Point2D::Point2D()
{
	x = 0;
	y = 0;
}
void Point2D::assignPoint2D(int n1, int n2)
{
	x = n1;
	y = n2;
}
void Point2D::displayPoint2D()
{
	std::cout << "(" << x << "," << y << ") = ";
	std::cout << value << std::endl;
}
void Point2D::setValue(double v)
{
	if (v < limit)
		value = v;
```

```
    else
        value = limit;
}
double Point2D::value = 0.0;
// It needs to initialize static member
int main()
{
    Point2D ptArray[10];
    ptArray[0].setValue(1.1);
    // modify the static member by static member function
    for (int i = 0;i < 10; i++)
    {
        ptArray[i].assignPoint2D(i,i+2);
        ptArray[i].displayPoint2D();
    }
    return 0;
}
```

- ✓ Remark the line double Point2D::value = 0.0; and compiler the program again. Try to explain the error message.
- ✓ Remove static in static const double limit = 10.0; and compiler the program again.
- ✓ Remove const in static const double limit = 10.0; and compiler the program again.
- ✓ Try to modify ptArray[0].setValue(1.1); as ptArray[0].setValue(30.1); and execute the program again.

LAB 5-2: THIS POINTER

- ✓ this pointer is an implicit private member to store the address of the object for a class.

```
// original version in lab4-2.cpp
PointND::PointND()
{
    value = 0.0;
    coord = new int [num];
    for (int i = 0; i < num; i++) coord[i] = 0;
}
```

```
// modify version in lab5-2.cpp
PointND::PointND()
{
this->value = 0.0;
this->coord = new int [num];
for (int i = 0; i < num; i++) this->coord[i] = 0;
}
```

- ✓ this pointer includes the address of the object, so it can be used to compare the addresses between different objects.

```
// lab5-2.cpp
#include <iostream>
/* class PointND declares and defines in lab 5-2 with copy constructor*/
/* add declaration of member function: copyPoint2D() to class PointND */
void PointND::copyPointND(const PointND &pt)
{
	if (this != &pt)
	{
		value = pt.value;
		coord = new int [num];
		for (int i = 0; i < num; i++) coord[i] = pt.coord[i];
	}
}
int main()
{
	int *vec = new int [num];
	for (int i = 0; i < num; i++) vec[i] = i;

	PointND pt1;
	pt1.assignValue(4.3);
	pt1.assignCoord(vec, num);
	pt1.displayPointND();
	PointND pt2;
	pt2.copyPointND(pt1);
	pt2.displayPointND();
	PointND pt3;
	pt3.copyPointND(pt3);
	pt3.displayPointND();
	delete []vec;

	return 0;
}
```

LAB 5-3: Overloaded Functions as Member Functions

✓ In this example, there are three overloaded constructors and two overloaded member functions.

```
// lab6-3.cpp
#include <iostream>
class Point2D
{
public:
	Point2D();
	Point2D(int n1, int n2);
	Point2D(int n1, int n2, double v);
	void assignPoint2D(int n1, int n2);
	void assignPoint2D(int n1, int n2, double v);
	void displayPoint2D() const;
private:
	int x;
	int y;
	double value;
};
```

```
Point2D::Point2D()
{
	x = 0;
	y = 0;
	value = 0;
}
Point2D::Point2D(int n1, int n2)
{
	assignPoint2D(n1, n2, 0.0);
}
Point2D::Point2D(int n1, int n2, double v)
{
	assignPoint2D(n1, n2, v);
}
void Point2D::assignPoint2D(int n1, int n2)
{
	assignPoint2D(n1, n2, value);
}
void Point2D::assignPoint2D(int n1, int n2, double v)
{
	x = n1;
	y = n2;
	value = v;
}
void Point2D::displayPoint2D() const
{
	std::cout << "(" << x << "," << y << ") = ";
	std::cout << value << std::endl;
}
int main()
{
	Point2D pt1(3,4,3.9);
	Point2D pt2;
	pt1.displayPoint2D();
	pt2.displayPoint2D();
	std::cout << "after assignment " << std::endl;
	pt1.assignPoint2D(1,3);
	pt2.assignPoint2D(2,3,1.1);
	pt1.displayPoint2D();
	pt2.displayPoint2D();
	return 0;
}
```

LAB 5-4: Overloaded Functions as Friend Functions

✓ Friend functions can be also overloaded.

```
// lab5-4.cpp
#include <iostream>
#include <cmath>
/* class Point2D declares and defines in lab5-3 */
/* add two overloaded functions as friend */
double distPoint2D(const Point2D &pt1, const Point2D &pt2)
```

```
{
	return sqrt((pt1.x - pt2.x)*(pt1.x - pt2.x) + (pt1.y - pt2.y)*(pt1.y - pt2.y));
}
double distPoint2D(const Point2D &pt1, const Point2D &pt2, const Point2D &pt3)
{
	double n1 = distPoint2D(pt1, pt2);
	double n2 = distPoint2D(pt1, pt3);
	double n3 = distPoint2D(pt2, pt3);
	return (n1 + n2 + n3);
}
int main()
{
	Point2D pt1(3,0);
	Point2D pt2(0,4);
	Point2D pt3(0,0);
	std::cout << "total distance: " << distPoint2D(pt1,pt2,pt3) << std::endl;
	return 0;
}
```

- ✓ How to choose the type of function? Is it better to set distPoint2D() as a common function or a member of the class object?
- ✓ There is no absolute answer. The type of function is depended on the functionality of this function. For example, Point2D is a class object, but distPoint2D is operated on the object Point2D instead of the object's property.

EXERCISE:5-1(DATE)

- ✓ Write a class Data to have the following capabilities.
  - Output the date in multiple formats such as
    DDD YYYY
    MM/DD/YY
    June 14, 1992
  - Use overloaded constructors to create Date objects initialized with dates of the formats as described.
  - Create a Date constructor that reads the system date, using the standard library functions of the `<ctime>` header, and sets the Date members.
- ✓ Hints:
  - There are four constructors for this class: a default constructor that sets the date to the current date, using `<ctime>`; a constructor that takes a date in the form (DDD, YYYY); where DDD represents the day of the year, a constructor that takes a date in the form (MM, DD, YY) and a constructor which takes the month name, day and year. Use a `char*` and two `int`s for the last constructor.
  - In addition to the four constructors, include functions for setting the month, day and year. No other data members are necessary.
  - Write three different printing member functions. You may find it necessary to implement helper member functions that perform the following tasks:

- Return the name of a month (as a `char*`).
- Return the number of days in a month.
- Test for a leap year. A year is a leap year if it is divisible 400 or divisible by four and not by 100.
- Return the name of a month.
- Convert DDD to MM DD.
- Convert MM DD to DDD.
- Convert from month name to MM.

✓ The main function is defined as follows

```
// ex5-1.cpp
// Driver program for class Date.
#include <iostream>
using std::cout;
using std::endl;
#include "Date.h" // include Date class definition
int main()
{
	Date date1( 256, 1999 ); // initialize using ddd yyyy format
	Date date2( 3, 25, 04 ); // initialize using mm/dd/yy format
	Date date3( "September", 1, 2000 ); // "month" dd, yyyy format
	Date date4; // initialize to current date with default constructor
	// print Date objects in default format
	date1.print();
	date2.print();
	date3.print();
	date4.print();
	cout << endl;
	// print Date objects in 'ddd yyyy' format
	date1.printDDDYYYY();
	date2.printDDDYYYY();
	date3.printDDDYYYY();
	date4.printDDDYYYY();
	cout << endl;
	// print Date objects in 'mm/dd/yy' format
	date1.printMMDDYY();
	date2.printMMDDYY();
	date3.printMMDDYY();
	date4.printMMDDYY();
	cout << endl;
	// print Date objects in '"month" d, yyyy' format
	date1.printMonthDDYYYY();
	date2.printMonthDDYYYY();
	date3.printMonthDDYYYY();
	date4.printMonthDDYYYY();
	cout << endl;
	return 0;
} // end main
```

✓ The sample output is

```
9/13/1999
3/25/2004
9/1/2000
4/10/2013

256 1999
85 2004
245 2000
100 2013

09/13/99
03/25/04
09/01/00
04/10/13

September 13, 1999
March 25, 2004
September 1, 2000
April 10, 2013

Date object destructor for date 4/10/2013
Date object destructor for date 9/1/2000
Date object destructor for date 3/25/2004
Date object destructor for date 9/13/1999
```

- ✓ The class Data is defined as

```
// Date.h
#ifndef DATE_H
#define DATE_H
#include <string>
using std::string;
class Date
{
public:
    Date(); // default constructor uses <ctime> functions to set date
    Date( int, int ); // constructor using ddd yyyy format
    Date( int, int, int ); // constructor using dd/mm/yy format
    Date( string, int, int ); // constructor using Month dd, yyyy format
    void setDay( int ); // set the day
    void setMonth( int ); // set the month
    void print() const; // print date in month/day/year format
    void printDDDYYYY() const; // print date in ddd yyyy format
    void printMMDDYY() const; // print date in mm/dd/yy format
    void printMonthDDYYYY() const; // print date in Month dd, yyyy format
    ~Date(); // provided to confirm destruction order
private:
    int month; // 1-12 (January-December)
    int day; // 1-31 based on month
    int year; // any year
    int checkDay( int ) const; // check if day is proper for month and year
    int daysInMonth( int ) const; // returns number of days in given month
    bool isLeapYear() const; // indicates whether date is in a leap year
```

```
    int convertDDToDDD() const; // get 3-digit day based on month and day
    void setMMDDFromDDD( int ); // set month and day based on 3-digit day
    string convertMMToMonth( int ) const; // convert mm to month name
    void setMMFromMonth( string ); // convert month name to mm
    int convertYYYYToYY() const; // get 2-digit year based on 4-digit
    void setYYYYFromYY( int ); // set year based on 2-digit year
}; // end class Date
#endif
```