C++ 프로그래밍 실습

Visual Studio 2015

Contents

- Classes
- Exercise
- 시험공지

- **Practice1 – Classes**

Practice 1-1 : Class Declarations

- A method can be defined either
  - *Inside* of the class declaration, or
  - *Outside* of the class declaration

```
class Person {
 public:
   // methods defined inside
   //   of class declaration
   void setAge(unsigned n)
        { age = n;  };
   unsigned getAge()
        { return age; };
 private:
   unsigned age;
};
```

```
class Person {
 public:
   void setAge(unsigned n);
   unsigned getAge();
 private:
   unsigned age;
};

// methods defined outside
void Person::setAge(unsigned n)
{ age = n; }

unsigned Person::getAge()
{ return age; }
```

Practice 1-1 : Class Declarations

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

class Date {
private:
   int year;
   int month;
   int day;

public:
   void setDate(int yy, int mm, int dd);
   void display();
};

void Date::setDate(int yy, int mm, int dd) {
   year = yy;
   month = mm;
   day = dd;
}
void Date::display() {
   cout << year << "." << month << "." << day << endl;
}

int main() {
   Date birthday;
   birthday.setDate(1999, 11, 22);
   birthday.display();
}
```

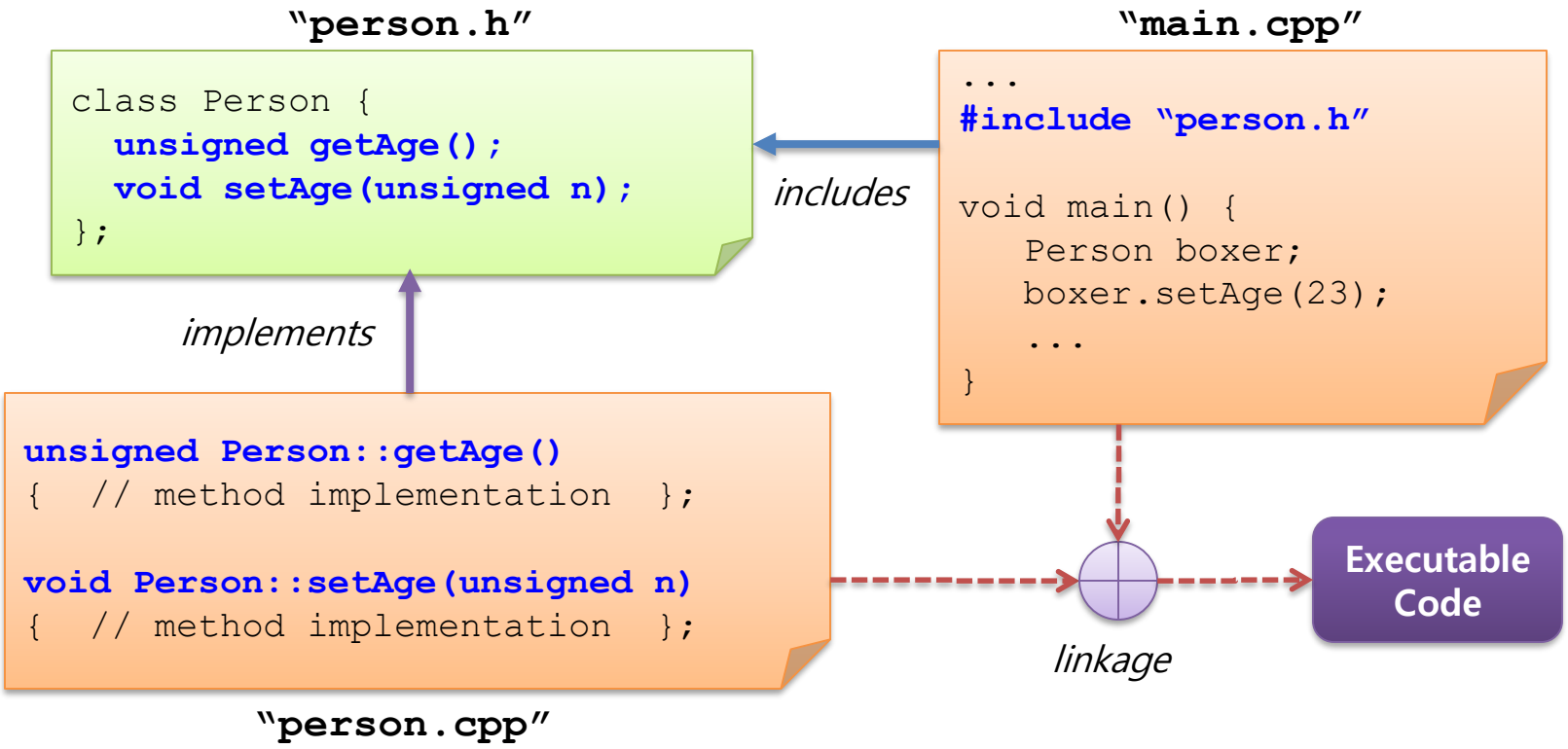
methods defined outside

Execution Result:

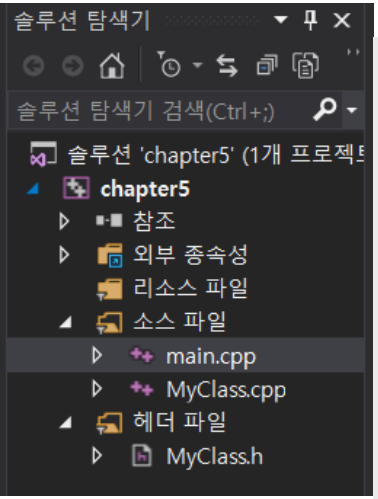
```
1999.11.22
```

Practice 1-2 : Writing Classes into Files

- Normally, class declarations are placed in header files and then included whenever needed
  - Method definitions are placed in separate CPP files (note **`inline`** methods can be defined in header files)



Practice 1-2 : Writing Classes into Files




```
#include<iostream>
#include<string>

class MyClass {
private:
char c;
public:
void MyFunc(int i);
};
```

```
#include "MyClass.h"

using namespace std;

void MyClass::MyFunc(int i) {
    if (i > 127) throw 1;
    if (i < -128) throw'm';
    c = i;
    cout << "Value of char c = "
<< (int)c << endl;
}
```

MyClass.h

Declarations

implements

MyClass.cpp

Definitions

Practice 1-2 Writing Classes into Files

main.cpp includes

```
#include<iostream>
#include<string>
#include"MyClass.h"

using namespace std;

int main() {
   MyClass c;
   try
   {
   c.MyFunc(-228);
   }
   catch (int i)
   {
   if (i == 1) cout << "MyFunc argment is too large. " << endl;
   }
   catch (char c)
   {
   cout << "MyFunc argment is too small. " << endl;
   return -1;
   }
return 0;
}
```

Execution Result:

```
MyFunc argment is too small.
```

Practice 1-3 : Pointers to Objects

- Pointers to objects are frequently used in C++ applications in the following context
  - Dynamically allocated objects by **`new`** and **`new[]`** operators
    - E.g.)
```
Person* boxer = new Person;
Person* boxers = new Person[10];
```
  - Call or return by reference using object pointers

- Access to an object's members through a pointer
  - Using the class indirection operator "->" instead of the member selector operator "."
    - Cf., to access object members through references, the member selector operator "." is used

Practice 1-3 Pointers to Objects

```
#include<iostream>

using namespace std;

class Circle {
private:
   int radius;
public:
   Circle() { radius = 1; }
   Circle(int r) { radius = r; }
   double getArea();
};

double Circle::getArea() {
   return 3.14 * radius *radius;
}

int main() {
   Circle donut;
   Circle pizza(30);

   //객체 이름으로 멤버 접근
   cout << donut.getArea() << endl;

   //객체 포인터로 멤버 접근
   Circle *p;
   p = &donut;
   cout << p->getArea() << endl;
   cout << (*p).getArea() << endl;

   p = &pizza;
   cout << p->getArea() << endl;
   cout << (*p).getArea() << endl;
}
```

Execution Result:

```
3.14
3.14
3.14
2826
2826
```

Practice 1-4 this

- The pointer constant **this** can be used inside a method to access the object associated with the method's invocation
  - I.e., **this** points to the object containing the method itself

```
class C {
 private:
   int x;
     ...
 public:
   m() { x = 0; }
     ...
}
```

=

```
class C {
 private:
   int x;
     ...
 public:
   m() { this->x = 0; }
     ...
}
```

Practice 1-4 this

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

class MyClass
{
private:
   int num1;
   int num2;
public:
   MyClass(int num1, int num2)
   {
   this->num1 = num1;
   this->num2 = num2;
   }
   void getInfo()
   {
   cout << "num1: " << num1 << endl;
   cout << "num2: " << num2 << endl;
   }
};

int main()
{
   MyClass mc(10, 20);

   mc.getInfo();
   return 0;
}
```

Execution Result:

```
num1: 10
num2: 20
```

Practice 1-5 Passing and Returning Objects by Reference

- Like other variables, objects can be passed to or returned from functions or methods
  - Call/return by value (default)
  - Call/return by reference (pointer or reference type)
- Objects required to be passed or returned by reference
  - Passing or returning objects by value can be inefficient due to copying objects, which may be large (i.e., waste of storage and time)
  - Using reference is easier at syntax level than using pointer to objects (i.e., no dereferencing is required)
- When returning local objects
  - The returned objects must be **`static`**, otherwise, the invoker may receive a nonexistent object
  - Note that once initialized, **`static`** objects last the program's whole execution time (similar to global objects except for naming scope)

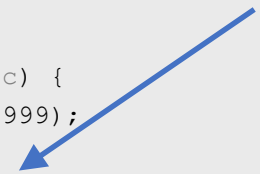
Practice 1-5 Passing and Returning Objects by Reference

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

class C {
private:
   int num;
public:
   void set(int n) {
      num = n;
}
   int get(){
      return num;
   }
};

void f(C& c) {
   c.set(-999);
}
C& g1() {
   static C c;
   c.set(123);
   return c;
}

int main() {
   C c1, c2;
   f(c1);
   cout << c1.get() << endl;
   c2 = g1();
   cout << c2.get() << endl;
}
```



storage for c will remain even after return

Execution Result:

```
-999
123
```

Practice 1-6 : Constructors

- A constructor is a method whose name is **the same as the class name** and has **no return type**
  - Can be overloaded: the most suitable constructor is invoked automatically whenever an object of the class is created
  - Types of constructors
    - Default constructors (without parameters)
      - Automatically generated in the absence of explicit constructors
    - Parameterized constructors (with parameters)

```
class Person {
public:
   Person();                          // constructor (default)
   void Person();                     // error: no return type required
   Person(const string& n);           // constructor (parameterized)
   Person(const char* n);             // constructor (parameterized)
   ...
   void setName(const string& n);
private:
   string name;
};
```

Practice 1-6 : Constructors

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

class Circle {
private:
   int radius;
public:
   Circle(); //기본 생성자
   Circle(int r); //매개 변수 있는 생성자
   double getArea();
};

Circle::Circle() {
   radius = 1; //반지름 값 초기화
   cout << "반지름: " << radius << endl;
}

Circle::Circle(int r) {
   radius = r; //반지름 값 초기화
   cout << "반지름: " << radius << endl;
}

double Circle::getArea(){
   return 3.14 *radius*radius;
}
```

Practice 1-6 : Constructors

```
int main() {
   Circle donut; //매개 변수 없는 생성자 호출
   double area = donut.getArea();
   cout << "donut의 면적은 " << area << endl;

   Circle pizza(30); // 매개 변수 있는 생성자 호출. 30이 r에 전달됨
   area = pizza.getArea();
   cout << "pizza의 면적은 " << area << endl;
}
```

Execution Result:

```
반지름: 1
donut의 면적은 3.14
반지름: 30
pizza의 면적은 2826
```

Practice 1-7 : Constructor Initializers

- Data members of a class can be initialized in
  - the body of the constructor using the assignment operator "=", or
  - the initialization section of the constructor using the ***constructor initializers***

```
class C {
 private:
  int x;
  const int c;

 public:
  C() {
     x = -1;
     c = 0; // error: c is const
  }
};
```

```
class C {
 private:
  int x;    // initialized first
  const int c;

 public:
  C() : c(0), x(-1) {
     /* empty */
  }
};
```

❖ Note that
  - Operator "=" can not be applied to **`const`** members, but the initializer can
  - Initialization occurs in the order in which the members are declared

Practice 1-7 Constructor Initializers

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

class MyClass
{
private:
   int num1;
   int num2;
public:
   MyClass(int num1, int num2) : num1(num1), num2(num2)
   {
   /* empty */
   }
   void getInfo()
   {
   cout << "num1: " << num1 << endl;
   cout << "num2: " << num2 << endl;
   }
};

int main()
   {
   MyClass mc(10, 20);

   mc.getInfo();
   return 0;
   }
```

Practice 1-8 : Destructors

- A method automatically invoked whenever objects are destroyed
  - For example,
    - when local objects go out of scope, or
    - dynamically allocated objects are deleted
  - Destructor prototype of a class ***C***

```
~C();
```

    - No return type and no parameters
    - Only one destructor per a class

Practice 1-8 : Destructors

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

class C {
private:
   string name;

public:
   C() {
   name = "anonymous";
   cout << name << " constructing" << endl;
   }
   C(string n) {
   name = n;
   cout << name << " constructing" << endl;
   }
   ~C() {
   cout << name << " destructing" << endl;
   }
};

int main() {
   C c0("bar");
   {
   C c1;
   C c2("foo");
   }
   C* ptr = new C();
   delete ptr;
   return 0;
}
```

destructor

c1 and c2 are destroyed

ptr is destroyed

c0 is destroyed

Execution Result:

```
bar constructing
anonymous constructing
foo constructing
foo destructing
anonymous destructing
anonymous constructing
anonymous destructing
bar destructing
```

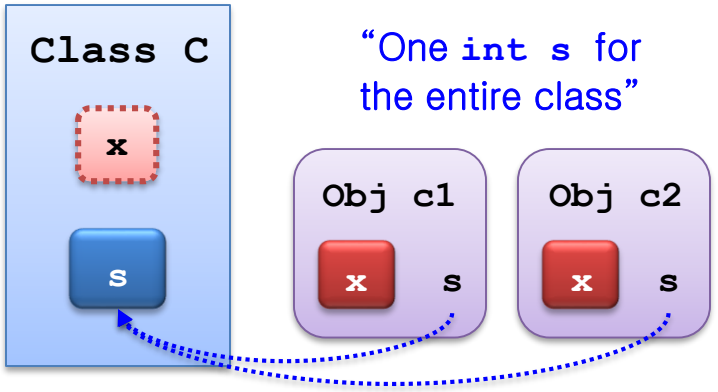
Practice 1-9 : Class members

- So far, we have seen data members and methods associated with individual objects
  - I.e., when we create two objects, each has its own data members and methods (called *object members* or *instance members*)
- C++ supports members associated with the class itself rather than its objects
  - They are called ***class members*** as opposed to object members
  - The keyword **`static`** is used to create class members

```
class C {
   int x;  // object data member

   static int s;
           // class data member
   ...
};

C c1, c2;
```



Practice 1-9 Static class members

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

class C {
private:
    static int num;
    int x;
public:
    C() {
        x = 0;
    }
    void setValues(int n) {
        num = n;
        x = n;
        }
    void addValues() {
        num++;
        x++;
    }
    void printValues() {
        cout << "num= " << num << " x= " << x << endl;
    }
};
int C::num = 0;
```

need to initialize the object num within the global scope

Practice 1-9 Static class members

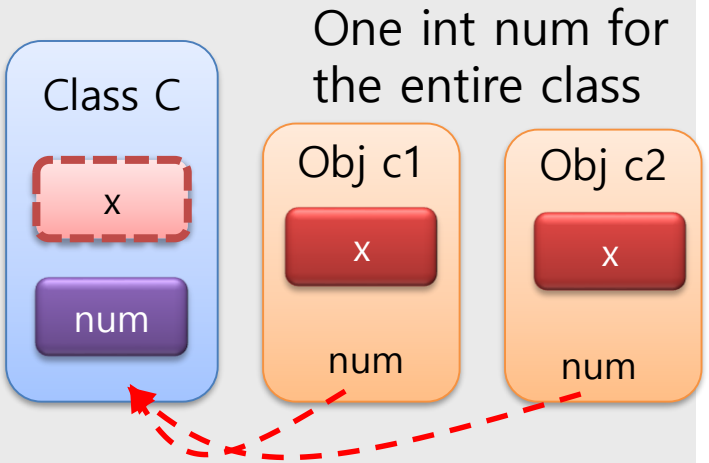
```
int main() {
    C c1, c2, c3;
    c1.setValues(5);

    c1.addValues();
    c2.addValues();
    c3.addValues();
    c2.addValues();

    cout << "c1: ";
    c1.printValues();

    cout << "c2: ";
    c2.printValues();

    cout << "c3: ";
    c3.printValues();
}
```



Execution Result:

```
c1: num= 9 x= 6
c2: num= 9 x= 2
c3: num= 9 x= 1
```

Exercise

- 자신만의 개성 있는 Class를 설계하시오

  - 클래스 선언부와 구현부를 헤더 파일과 cpp파일로 나누어서 프로그램을 작성하시오
  - Constructors를 사용하시오.
  - Pointers to Objects 사용하시오.

- 주석은 필수

중간고사 공지

o 시험범위: 시험날짜 전주에 실습한 내용까지

o 시험장소: 수업하는 강의실

o 시험일정:

- C++프로그래밍: 10월 17일 (수) 13:00~15:00

[주의사항]

- 강의자료만 지참 가능 (강의자료 이외의 어떠한 자료도 보면 안되고 적발시 0점 처리)

- 강의자료 출력물에 필기한 내용은 참조 가능함. 그러나 강의자료를 정리한 별도의 노트는 지참 불가

- 개인 노트북 사용 가능. 강의자료는 파일 형태로 봐도 무방(태블릿 가능, 스마트폰 불가)

- 시험중에 어떠한 형태로든 인터넷에 접속하는 경우 즉시 0점 처리함

- 시험답안은 이메일로 제출하며, 제출시 발생하는 모든 문제에 대해서는 제출자 본인의 책임이므로 반드시 제출 이상여부 확인후 퇴실할 것

Course Homepage

- How to access
  - URL: sclab.konkuk.ac.kr
- Downloading class material
  - Students can download syllabus and lecture notes in PDF format
- Class announcement
  - About homework and project
  - Exam schedule and result
  - And so on

Submit

- Teaching assistant: 장성수
  Office: 신공학관 1216호 (대학원 SCLab 연구실)
  Email: pik1100@naver.com

- Title of the email : [2018][Practice#]_student# _ student _ name

- Ex) [2018][Practice04]_201700000_장성수

- Create zip file. (C++ project folder)

- 주의 : 메일 양식이 잘못될 경우 채점이 되지 않을 수 있음.

- 질문 메일 : pik1100@naver.com : 장성수

끝