프로그래밍 언어 (2)
실습

참고 문헌 : IT강좌 프론티어 (www.freelec.com)

오버라이딩

- **ex1)**

```
#include <iostream>
#include <string>
using namespace std;
class Parent {
  string s;
 public:
  Parent() : s("부모") { cout << "부모 클래스" << endl; }
  void what() { cout << s << endl; }
};
class Child : public Parent {
  string s;
 public:
  Child() : s("자식"), Parent() { cout << "자식 클래스" << endl; }
  void what() { cout << s << endl; }
};
int main() {
  cout << " === 부모 클래스 생성 ===" << endl;
  Parent p;
  p.what();
  cout << " === 자식 클래스 생성 ===" << endl;
  Child c;
  c.what();

  return 0;
}
```

오버라이딩

- ex2)

```
#include <iostream>
using namespace std;
class First{
public:
    void MyFunc()
    {
        cout << "First::MyFunc()" << endl;
    }
    void Func1()
    {
        cout << "First::Func1()" << endl;
    }
};

class Second : public First{
public:
    void MyFunc() // 함수 오버라이딩 (First 클래스에 MyFunc()가 있음)
    {
        cout << "Second::MyFunc()" << endl;
    }
    void Func2()
    {
        cout << "Second::Func2()" << endl;
    }
};
class Third : public Second{
public:
    void MyFunc() // 함수 오버라이딩 (Second 클래스에 MyFunc()가 있음)
    {
        cout << "Third::MyFunc()" << endl;
    }
    void Func3()
    {
        cout << "Third::Func3()" << endl;
    }
};
```

오버라이딩

- **ex2)계속**

```
int main(){
    Third* third = new Third;
    Second* second = third; // Ok, third를 만들때 이미 Second, First 객체를 묵시적으로 만들었음
    First* first = second;  // Ok, second를 만들때 이미 First 객체를 묵시적으로 만들었음

    // 하지만 맴버 함수를 접근할 때에는 해당 클래스의 함수만 접근 가능
    first->MyFunc();
    second->MyFunc();
    third->MyFunc();
    cout << endl;

    first->Func1(); // Ok
    //first->Func2(); // 에러: First 클래스에는 Func2() 맴버 함수가 없다.
    //first->Func3(); // 에러: First 클래스에는 Func3() 맴버 함수가 없다.
    cout << endl;
    second->Func1(); // Ok, First 클래스를 상속받았기에 Func1()에 접근 가능
    second->Func2(); // Ok, 자기 자신의 멤버함수
    //second->Func3(); // 에러: Second 클래스에는 Func3() 맴버 함수가 없다.
    cout << endl;
    third->Func1(); // Ok, Third 클래스는 Second 클래스를 통하여 결국 First 클래스를 상속 받았음
    third->Func2(); // Ok, Third 클래스는 Second 클래스를 상속 받았음
    third->Func3(); // Ok, 자기 자신의 멤버함수

    system("pause");    // VC++ 에서만
    return 0;
}
```

가상함수

- **ex3)**

```
#include <iostream>
#include <string>
using namespace std;
class Parent {
  string s;
 public:
  Parent() : s("부모") { cout << "부모 클래스" << endl; }
  virtual void what() { cout << s << endl; }
};
class Child : public Parent {
  string s;
 public:
  Child() : s("자식"), Parent() { cout << "자식 클래스" << endl; }
  void what() { cout << s << endl; }
};

int main() {
  Parent p;
  Child c;
  Parent* p_c = &c;
  Parent* p_p = &p;

  cout << " == 실제 객체는 Parent == " << endl;
  p_p->what();
  cout << " == 실제 객체는 Child == " << endl;
  p_c->what();

  return 0;
}
```

가상함수

- **ex4)**

```
#include <iostream>
#include <string>
using namespace std;
class First{
public:
    virtual void MyFunc() // 가상 함수
    {
        cout << "First::MyFunc()" << endl;
    }
};

class Second : public First{
public:
    virtual void MyFunc() // 가상 함수
    {
        cout << "Second::MyFunc()" << endl;
    }
};

class Third : public Second{
public:
    virtual void MyFunc() // 가상 함수
    {
        cout << "Third::MyFunc()" << endl;
    }
};
```

가상함수

- **ex4)계속**

```
int main()
{
    Third* third = new Third;
    Second* second = third; // Ok, third를 만들때 이미 Second, First 객체를 묵시적으로 만들었음
    First* first = second;  // Ok, second를 만들때 이미 First 객체를 묵시적으로 만들었음

    cout << "address of third (즉, thrid 안의 내용  = 주소) " << third << endl;
    cout << "address of second(즉, second 안의 내용 = 주소) " << second << endl;
    cout << "address of first (즉, first 안의 내용  = 주소) " << first << endl;

    // 가상함수로 선언된 맴버 함수를 접근할 때에는 실제 객체의 맴버 함수 호출
    // 가상함수로 선언된 third = second = first 인 상황이므로 third의 맴버 함수 호출
    first->MyFunc();
    second->MyFunc();
    third->MyFunc();

    system("pause");    // VC++ 에서만
    return 0;
}
```

연산자 오버로딩

- **ex5)**

```
#include <iostream>
using namespace std;
class Vector
{
private:
    double x, y;
public:
    Vector(double x, double y){
        this->x = x;
        this->y = y;
    }
    friend Vector operator+(const Vector& v1, const Vector& v2);
    void display()
    {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};
Vector operator+(const Vector& v1, const Vector& v2)
{
    Vector v(0.0, 0.0);
    v.x = v1.x + v2.x;
    v.y = v1.y + v2.y;
    return v;
}
```

```
int main()
{
    Vector v1(1, 2), v2(3, 4);
    Vector v3 = v1 + v2;
    v3.display();

    return 0;
}
```

연산자 오버로딩

- **ex6)**

```
#include <iostream>
using namespace std;
class Point {
private:
    int x, y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y){}
    void ShowPosition();
    Point operator+(const Point& p);
};
void Point::ShowPosition(){
    cout<<x<<" "<<y<<endl;
}
Point Point::operator+(const Point& p){
    Point temp(x+p.x, y+p.y);
    return temp;
}
int main(void){
    Point p1(1, 2);
    Point p2(2, 1);
    Point p3=p1+p2;
    p3.ShowPosition();
    return 0;
}
```

연산자 오버로딩

- EX7)

```
#include <iostream>
using namespace std;
class Position
{
private:
    double x_;
    double y_;
public:
    Position(double x, double y);   // 생성자
    void Display();
    Position operator-(const Position& other);  // - 연산자 함수
};
int main(void)
{
    Position pos1 = Position(3.3, 12.5);
    Position pos2 = Position(-14.4, 7.8);
    Position pos3 = pos1 - pos2;

    pos3.Display();
    return 0;
}
```

연산자 오버로딩

- EX7)계속

```
Position::Position(double x, double y)
{
    x_ = x;
    y_ = y;
}
Position Position::operator-(const Position& other)
{
    return Position((x_ + other.x_)/2, (y_ + other.y_)/2);
}
void Position::Display()
{
    cout << "두 지점의 중간 지점의 좌표는 x좌표가 " << this->x_ << "이고, y좌표가 " << this->y_ << "입니다." << endl;
}
```

참고문헌

- https://modoocode.com/210
- https://wikidocs.net/22468
- http://algamza.blogspot.com/2016/03/c-operator-overloading.html
- http://tcpschool.com/cpp/cpp_operatorOverloading_intro

Q & A

참고 문헌 : IT강좌 프론티어 (www.freelec.com)