dongguk UNIVERSITY

심화프로그래밍

실습강의 11주차

실습강의 소개

- **실습 진행 방법**
  - 간단한 이론 복습 및 해당주차 실습과제 설명
  - 실습 후 보고서와 소스코드를 압축하여 **수요일 자정(23:59)까지**

    꼭!! 이클래스 제출(**이메일 제출 불가, 반드시 이클래스를 통해 제출**)
  - 실습과제 제출기한 엄수(제출기한 이후로는 0점 처리)
- **Q & A**
  - 이클래스 및 실습조교 이메일을 통해 질의응답
  - **이메일 제목 : [심화프로그래밍_홍길동] *본인 과목명과 성명 꼭 작성!!**
  - **실습조교 메일 주소 : 0hae@dgu.ac.kr, wundermilch@dgu.ac.kr**

실습 보고서 작성 방법 [1/2]

- **실습 보고서**
  - **문제 분석: 실습 문제에 대한 요구 사항 파악, 해결 방법 등 기술**
  - **프로그램 설계 및 알고리즘**
    - 해결 방법에 따라 프로그램 설계 및 알고리즘 등 기술
    - e.g.) 문제 해결 과정 및 핵심 알고리즘 기술
  - **소스코드 및 주석**
    - 소스코드와 그에 해당하는 주석 첨부
    - 각각의 함수가 수행하는 작업, 매개변수, 반환 값 등을 명시
    - 소스코드 전체 첨부**(소스코드 화면 캡처X, 소스코드는 복사/붙여넣기로 첨부)**
  - **결과 및 결과 분석**
    - 결과 화면을 캡쳐 하여 첨부, 해당 결과가 도출된 이유와 타당성 분석
  - **소감**
    - 실습 문제를 통해 습득할 수 있었던 지식, 느낀 점 등을 기술

실습 보고서 작성 방법 [2/2]

- **제출 방법**
  - 보고서, 소스코드를 1개의 파일로 압축하여 e-class "과제" 메뉴를 통해 제출
    - **"이름학번실습주차.zip" 형태로 제출(e.g. :김동국19919876실습11.zip)**
    - 파일명에 공백, 특수 문자 등 사용 금지
- **유의 사항**
  - 보고서의 표지에는 학과, 학번, 이름, 담당 교수님, 제출일자 반드시 작성
  - 정해진 기한내 제출
    - 기한 넘기면 **0점** 처리
    - 이클래스가 과제 제출 마지막 날 오류로 동작하지 않을 수 있으므로, 최소 1~2일전에 제출
    - **과제 제출 당일 이클래스 오류로 인한 미제출은 불인정**
  - 소스코드, 보고서를 자신이 작성하지 않은 경우 **실습 전체 점수 0점 처리**
  - **Visual Studio 2019 또는 Sharstra 웹 IDE 기반 학습 프로그램 사용하여 실습 진행**

바인딩 [1/2]

- **바인딩(Binding)**
  - 프로그램 소스에 쓰인 각종 내부 요소, 이름 식별자들에 대해 값 또는 속성을 확정하는 과정임
  - 바인딩 종류로는 정적 바인딩(Static Binding), 동적 바인딩(Dynamic Binding)가 있음
- **정적 바인딩**
  - 컴파일 시점에 이루어지며, 소스 코드 상에서 명시적으로 타입과 그 타입의 변수명을 선언함

```
1. int main() {
2.     int a = 0;
3. }
```

- **동적 바인딩**
  - 프로그램 실행 도중에 이루어지며, 일반적으로 가상함수, 상속, 포인터와 함께 사용됨
  - 즉, 포인터가 가리키는 객체에 따라 호출되는 함수가 변경되는 것임

바인딩 [2/2]

- **동적 바인딩 예제 소스**
  - print()라는 함수는 클래스 A에 가상함수(virtual: 부모 클래스에서 상속받을 클래스에서 재정의할 것으로 기대하고 정의해놓은 함수)로 선언됨, 클래스 B는 클래스 A를 상속받아 print() 함수를 재정의함
  - 이 때 main() 함수에서 클래스 A를 선언하여 print()를 호출할 경우, 클래스 A의 print() 함수가 호출되며, a 객체를 B클래스 객체의 포인터로 변경할 경우, B 클래스의 print() 함수가 호출됨.
  - 즉, 이 과정은 실행 도중 print() 함수가 A 클래스의 print() 함수에서 B 클래스의 print() 함수로 변경됨

```
1. class A {
2.   virtual void print() {
3.     std::cout << "A::print()" << std::endl;
4.   }
5. };
6.
7. class B : public A {
8.   void print() override {
9.     std::cout << "B::print()" << std::endl;
10.  }
11.};
12.
13.int main() {
14.  A* a = new A();
15.  a->print();
16.
17.  B* b = new B();
18.  a = b;
19.  a->print();
20.}
```

override는 가상함수를 가르키기 위한 키워드

```
A::print()
B::print()
```

오버로딩과 함수 재정의, 오버라이딩 비교

- **C++ 오버라이딩의 특징**
  - 메소드 재정의: 자식 클래스에서 부모 클래스의 메소드를 **재정의(override)**, 이 때, 메소드의 시그니처(이름, 매개변수 타입 및 개수)는 같아야 함.
  - 다형성 구현: 오버라이딩은 다형성을 가능하게 하는 핵심 메커니즘, 부모 클래스 타입의 **참조변수**로 자식 클래스의 오버라이딩된 메소드에 접근할 수 있음.
  - 가상 함수 사용: C++에서 오버라이딩은 주로 **가상 함수(virtual 키워드 사용)**를 통해 이루어짐. 부모 클래스에서 메소드를 virtual로 선언하면, 자식 클래스에서 해당 메소드를 오버라이딩할 수 있음
  - 동적 바인딩: 오버라이딩된 메소드는 **실행 시간에 (런타임에)** 결정됩니다. 이는 컴파일 시간이 아닌 런타임에 어떤 메소드가 호출될지 결정된다는 의미입니다.
- **오버로딩과 함수 재정의, 오버라이딩 비교**

비교 요소	오버로딩	함수 재정의(가상 함수가 아닌 멤버에 대해)	오버라이딩
정의	매개 변수 타입이나 개수가 다르지만, 이름이 같은 함수들이 중복 작성되는 것	기본 클래스의 멤버 함수를 파생 클래스에서 이름, 매개 변수 타입과 개수, 리턴 타입까지 완벽히 같은 원형으로 재작성하는 것	기본 클래스의 가상 함수를 파생 클래스에서 이름, 매개 변수 타입과 개수, 리턴 타입까지 완벽히 같은 원형으로 재작성하는 것
존재	클래스의 멤버들 사이, 외부 함수들 사이, 그리고 기본 클래스와 파생 클래스 사이에 존재 가능	상속 관계	상속 관계
목적	이름이 같은 여러 개의 함수를 중복 작성하여 사용의 편의성 향상	기본 클래스의 멤버 함수와 별도로 파생 클래스에서 필요하여 재작성	기본 클래스에 구현된 가상 함수를 무시하고, 파생 클래스에서 새로운 기능으로 재작성하고자 함
바인딩	정적 바인딩. 컴파일 시에 중복된 함수들의 호출 구분	정적 바인딩. 컴파일 시에 함수의 호출 구분	동적 바인딩. 실행 시간에 오버라이딩된 함수를 찾아 실행
객체 지향 특성	컴파일 시간 다형성	컴파일 시간 다형성	실행 시간 다형성

[연습 1] 파생 클래스에서 함수를 재정의하는 사례

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Base {
5.  public:
6.      void f() { cout << "Base::f() called" << endl; }
7.  };
8.
9.  class Derived : public Base {
10. public:
11.     void f() { cout << "Derived::f() called" << endl; }
12. };
13.
14. void main() {
15.     Derived d, *pDer;
16.     pDer = &d;
17.     pDer->f(); // Derived::f() 호출
18.
19.     Base* pBase;
20.     pBase = pDer; // 업캐스팅
21.     pBase->f(); // Base::f() 호출
22. }
```

```
Derived::f() called
Base::f() called
```

[연습 2] 오버라이딩과 가상 함수 호출

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Base {
5.  public:
6.      virtual void f() { cout << "Base::f() called" << endl; }
7.  };
8.
9.  class Derived : public Base {
10. public:
11.     virtual void f() { cout << "Derived::f() called" << endl; }
12. };
13.
14. int main() {
15.     Derived d, *pDer;
16.     pDer = &d;
17.     pDer->f(); // Derived::f() 호출
18.
19.     Base * pBase;
20.     pBase = pDer; // 업 캐스팅
21.     pBase->f(); // 동적 바인딩 발생!! Derived::f() 실행
22. }
```

```
Derived::f() called
Derived::f() called
```

[연습 3] 상속이 반복되는 경우 가상 함수 호출

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Base {
5.  public:
6.      virtual void f() { cout << "Base::f() called" << endl; }
7.  };
8.
9.  class Derived : public Base {
10. public:
11.     void f() { cout << "Derived::f() called" << endl; }
12. };
13.
14. class GrandDerived  : public Derived {
15. public:
16.     void f() { cout << "GrandDerived::f() called" << endl; }
17. };
18.
19. int main() {
20.     GrandDerived g;
21.     Base *bp;
22.     Derived *dp;
23.     GrandDerived *gp;
24.
25.     bp = dp = gp = &g;
26.
27.     bp->f();
28.     dp->f();
29.     gp->f();
30. }
```

```
GrandDerived::f() called
GrandDerived::f() called
GrandDerived::f() called
```

[연습 4] 범위 지정 연산자(::)를 이용한 가상 함수 호출

```
--Shape--Circle
--Shape--
```

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Shape {
5.  public:
6.      virtual void draw() {
7.          cout << "--Shape--";
8.      }
9.  };
10.
11. class Circle : public Shape {
12. public:
13.     virtual void draw() {
14.         Shape::draw(); // 기본 클래스의 draw() 호출
15.         cout << "Circle" << endl;
16.     }
17. };
18.
19. int main() {
20.     Circle circle;
21.     Shape * pShape = &circle;
22.
23.     pShape->draw();
24.     pShape->Shape::draw();
25. }
```

[연습 5] 소멸자를 가상 함수로 선언

```
1.#include <iostream>
2.using namespace std;
3.
4.class Base {
5.public:
6.    virtual ~Base() { cout << "~Base()" << endl; }
7.};
8.
9.class Derived: public Base {
10.public:
11.    virtual ~Derived() { cout << "~Derived()" << endl; }
12.};
13.
14.int main() {
15.    Derived *dp  = new Derived();
16.    Base *bp = new Derived();
17.
18.    delete dp; // Derived의 포인터로 소멸
19.    delete bp; // Base의 포인터로 소멸
20.}
```

```
~Derived()
~Base()
~Derived()
~Base()
```

연습문제(9장)

1. 호출하는 함수의 결정을 실행 시간에 하도록 컴파일러에게 지시하는 키워드는?
   ① virtual ② static ③ public ④ extern

2. 기본 클래스의 가상 함수와 동일한 타입의 함수를 파생 클래스에서 작성하는 것을 무엇이라고 하는가?
   ① overloading ② overriding ③ virtual ④ dynamic binding

3. 다음 중 의미가 다른 하나는?
   ① dynamic binding ② late binding
   ③ compile-time binding ④ run-time binding

4. 오버로딩, 오버라이딩, 연산자 중복 등은 C++ 언어의 어떤 객체 지향 특성인가?

연습문제(9장)

5. 다음 코드에 대해 물음에 답하여라.

```
class Base {
public:
   void func() { f(); }
   void f() { cout << "Base::f() called" << endl; }
};
class Derived : public Base {
public:
   void f() { cout << "Derived::f() called" << endl; }
};
```

(1) 기본 클래스와 파생 클래스는 무엇인가?

* 다음 코드가 실행될 때 화면에 출력되는 내용은?

```
Derived der;
der.f();   // (2)
Base base;
Base* p = &base;
p->f();    // (3)
p = &der;
p->f();    // (4)
p->func();    // (5)
```

연습문제(9장)

6. 다음 코드에 대해 물음에 답하여라.

```
class A {
public:
   void func() { f(); }
   virtual void f() { cout << "A::f() called" << endl; }
};
class B : public A {
public:
   void f() { cout << "B::f() called" << endl; }
};
class C : public B {
public:
   void f() { cout << "C::f() called" << endl; }
};
```

(1) 다음 함수 중 가상 함수를 모두 골라라.

① A의 f()　② B의 f()　③ C의 f()　④ A의 func()

* 다음 코드가 실행될 때 출력되는 결과는 무엇인가?

```
C c;
c.f();  // (2)
A* pa;
B* pb;
pa = pb = &c;
pb->f();   // (3)
pa->f();   // (4)
pa->func();  // (5)
```

연습문제(9장)

7. 다음 빈칸에 적절한 단어를 보기에서 골라 삽입하라.

동일한 이름의 변수나 함수가 여러 곳에 선언되었을 때, 가장 _____ 범위에 선언된 이름을 사용하는 규칙을 컴퓨터 언어 이론에서 _____(이)라고 한다. _____(을)를 사용하면 클래스 멤버와 동일한 이름의 외부 함수를 클래스 내에서 호출할 수 있다.

보기

생명 주기, 범위 규칙, 가까운, 먼, 전역, 지역, 범위 지정 연산자, 괄호 연산자, `virtual` 키워드, 상속, 동적 바인딩

연습문제(9장)

8. 각 문항에 따라 함수 g()의 빈칸에 적절한 코드는?

```
void f() {
   cout << "f() called" << endl;
}
class A {
public:
   virtual void f() { cout << "A::f() called" << endl; }
};
class B : public A {
public:
   void g() { _________ }
   void f() { cout << "B::f() called" << endl; }
};
```

(1) 함수 g()가 외부 함수 f()를 호출한다.
(2) 함수 g()가 클래스 A의 멤버 함수 f()를 호출한다.
(3) 함수 g()가 자신의 멤버 함수 f()를 호출한다.

연습문제(9장)

9. 생성자와 소멸자를 가상 함수로 선언하는 것에 대한 설명 중 맞는 것은?

① 소멸자는 동적 바인딩 되지 않는다.

② 소멸자를 가상 함수로 선언하는 것이 바람직하다.

③ 소멸자를 가상 함수로 선언해도 동적 바인딩이 일어나지 않는다.

④ 생성자를 가상 함수로 선언하는 것이 바람직하다.

연습문제(9장)

10. 다음은 Person 클래스와 파생 클래스 Student를 작성한 사례이다.

```
class Person {
   int id;
public:
   Person(int id=0) { this->id = id; }
   ~Person() { cout << "id=" << id << endl; }
};
class Student : public Person {
   char* name;
public:
   Student(int id, const char* name) : Person(id) {
      int len = strlen(name);
      this->name = new char [len + 1];
      strcpy(this->name, name);
   }
   ~Student() {
      cout << "name=" << name << endl;
      delete [] name;
   }
};
int main() {
   Person *p = new Student(10, "손연재");
   delete p;
}
```

(1) 다음 코드의 실행 결과는 무엇인가? 실행 결과에 대해 어떤 문제가 있다고 생각되는가?

(2) Person 클래스나 Student 클래스를 수정하여 문제점을 해결하라.

연습문제(9장)

11. 다음 중 순수 가상 함수는?

```
class Shape {
public:
   void draw()=0;          // ①
   virtual void draw();    // ②
   virtual void draw()=0;  // ③
   virtual void draw() { } // ④
};
```

12. 순수 가상 함수에 대해 잘못 말한 것은?

① 순수 가상 함수를 가진 클래스는 무조건 추상 클래스이다.

② 순수 가상 함수는 실행이 목적이 아니라, 파생 클래스가 구현해야 할 함수를 알려 주기 위한 것이다.

③ 외부 함수도 순수 가상 함수로 선언 가능하다.

④ 순수 가상 함수가 호출되면 동적 바인딩이 일어난다.

13. 다음에서 추상 클래스는?

①
```
class Shape {
   void draw()=0;
};
```

②
```
class Shape {
   virtual void draw()=0;
};
```

③
```
class Shape {
   virtual void draw() {}=0;
};
```

④
```
abstract class Shape {
   virtual void draw()=0;
};
```

연습문제(9장)

14. 다음 코드에 대해 물음에 답하여라.

```
class Shape {
public:
   void paint() { draw(); }
   virtual void draw()=0;
};
class Circle : public Shape {
   int radius;
public:
   Circle(int radius=1) { this->radius = radius; }
   double getArea() { return 3.14*radius*radius; }
};
```

(1) 다음 중 오류가 발생하는 것을 있는 대로 골라라.

① Shape shape;

② Shape* p;

③ Circle circle(10);

④ Circle *pCircle;

(2) 다음 코드의 실행 결과 "반지름=10인 원"이 출력되도록 Shape이나 Circle 클래스를 수정하라.

```
Circle *p = new Circle(10);
p->paint();
```

실행 결과

```
반지름=10인 원
```

연습문제(9장)

15. 다음 중 컴파일 시에 바인딩되지 않는 것은?
① 중복된 함수 중에서 구분하여 호출
② 중복된 연산자 중에서 구분하여 호출
③ 범위 지정 연산자(::)를 이용한 함수 호출
④ 순수 가상 함수 호출

실습과제(9장)

다음은 단위를 변환하는 추상 클래스 Converter이다.

```
class Converter {
protected:
    double ratio;
    virtual double convert(double src)=0; // src를 다른 단위로 변환한다.
    virtual string getSourceString()=0; // src 단위 명칭
    virtual string getDestString()=0; // dest 단위 명칭
public:
    Converter (double ratio) { this->ratio = ratio; }
    void run () {
        double src;
        cout << getSourceString() << "을 " << getDestString() << "로 바꿉니다. ";
        cout << getSourceString() << "을 입력하세요>> ";
        cin >> src;
        cout << "변환 결과 :" << convert(src) << getDestString() << endl ;
    }
};
```

실습과제(9장)

1. Converter 클래스를 상속받아 km를 mile(마일)로 변환하는 KmToMile 클래스를 작성하라. main() 함수와 실행 결과는 다음과 같다.

```
int main () {
    KmToMile toMile(1.609344); // 1 mile은 1.609344 Km
    toMile.run();
}
```

```
Km을 Mile로 바꿉니다. Km을 입력하세요>>  25
변환 결과 : 15.5343Mile
```

실습과제(9장)

다음 추상 클래스 LoopAdder가 있다.

```
class LoopAdder { // 추상 클래스
    string name; // 루프의 이름
    int x, y, sum; // x에서 y까지의 합은 sum
    void read(); // x, y 값을 읽어 들이는 함수
    void write(); // sum을 출력하는 함수
protected:
    LoopAdder (string name="") { // 루프의 이름을 받는다. 초깃값은 ""
        this->name = name;
    }
    int getX() { return x; }
    int getY() { return y; }
    virtual int calculate() = 0; // 순수 가상 함수. 루프를 돌며 합을 구하는 함수
public:
    void run(); // 연산을 진행하는 함수
};

void LoopAdder::read() { // x, y 입력
    cout << name << ":" << endl;
    cout << "처음 수에서 두번째 수까지 더합니다. 두 수를 입력하세요 >>";
    cin >> x >> y;
}
void LoopAdder::write() { // 결과 sum 출력
    cout << x << "에서 " << y << "까지의 합 =" << sum << " 입니다"<< endl;
}

void LoopAdder::run () {
    read(); // x, y를 읽는다.
    sum = calculate(); // 루프를 돌면서 계산한다.
    write(); // 결과 sum을 출력한다.
}
```


실습과제(9장)

2. LoopAdder 클래스를 상속받아 다음 main() 함수와 실행 결과처럼 되도록 WhileLoopAdder, DoWhileLoopAdder 클래스를 작성하라. while 문, do-while 문을 이용하여 합을 구하도록 calculate() 함수를 각각 작성하면 된다.

```
int main() {
    WhileLoopAdder whileLoop("While Loop");
    DoWhileLoopAdder doWhileLoop("Do while Loop");

    whileLoop.run();
    doWhileLoop.run();
}
```

```
While Loop:
처음 수에서 두번째 수까지 더합니다. 두 수를 입력하세요 >> 3 5
3에서 5까지의 합 = 12 입니다
Do while Loop:
처음 수에서 두번째 수까지 더합니다. 두 수를 입력하세요 >> 10 20
10에서 20까지의 합 = 165 입니다
```

실습과제(9장)

3. 다음 AbstractStack은 정수 스택 클래스로서 추상 클래스이다.

```
class AbstractStack {
public:
    virtual bool push(int n) = 0; // 스택에 n을 푸시한다. 스택이 full이면 false 리턴
    virtual bool pop(int& n) =0; // 스택에서 팝한 정수를 n에 저장하고 스택이 empty이면 false 리턴
    virtual int size() = 0; // 현재 스택에 저장된 정수의 개수 리턴
};
```

이를 상속받아 정수를 푸시, 팝하는 IntStack 클래스를 만들고 사용 사례를 보여라.

실습과제(9장)

4. 간단한 그래픽 편집기를 콘솔 바탕으로 만들어보자. 그래픽 편집기의 기능은 "삽입", "삭제", "모두보기", "종료" 의 4가지이고, 실행 과정은 다음과 같다.

```
그래픽 에디터입니다.
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 2
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 3
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 3
0: Line
1: Circle
2: Rectangle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 2
삭제하고자 하는 도형의 인덱스 >> 1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 3
0: Line
1: Rectangle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 4
```

실습과제(9장)

4. (앞 슬라이드 계속)힌트

Shape과 이를 상속받은 Circle, Line, Rect 클래스는 [그림 9-13]을 이용하고 필요한 클래스와 main() 함수를 작성하라. 전체 프로그램은 대략 아래와 같이 구성된다.