dongguk UNIVERSITY

심화프로그래밍

실습강의 6주차

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. :김동국19919876실습6.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**

함수의 인자 전달 방식

- **값에 의한 호출, Call by Value**
  - C++에서 포인터가 아닌 인수는 **값**으로 전달됨
  - 인수가 값으로 전달되면 인수의 값은 해당 함수 매개 변수의 값으로 **복사**됨
- **주소에 의한 호출, Call by Address**
  - 함수에 변수를 전달할 수 있는 또 다른 방법 중 하나로 **주소를 사용하는 것임**
  - 인수가 주소이기 때문에 함수 매개 변수는 **포인터**를 사용함
  - 함수는 가리키는 값에 **접근**하거나 **변경**하기 위해 포인터를 **역참조**할 수 있음
- **참조에 의한 호출, Call by Reference**
  - 큰 구조체 또는 클래스를 함수에 전달할 때 큰 비용과 성능 저하가 발생할 수 있음
  - 함수의 반환 값을 사용하는 것도 좋지만, 함수에서 **인수를 수정하는 것**이 더 명확하고 효율적임
  - 참조를 사용하면 위의 두 가지 문제(큰 비용, 성능 저하)를 해결할 수 있다.

[예시 1] 값에 의한 호출, Call by value

- **값에 의한 호출 예시**
  - foo(5)가 호출되면 변수 y가 만들어지고 값 5가 y로 복사됨
    - 변수 y는 foo() 함수가 종료되면 소멸함
  - foo(x) 호출에서 인수는 변수 x, x는 값 6으로 평가됨
    - 변수 y가 다시 만들어지고 값 6이 복사됨, 마찬가지로 foo() 함수가 종료되면 소멸함
  - foo(x + 1) 호출에서 인수는 값 7로 평가되며, 변수 y로 복사되고 foo() 함수가 끝나면 소멸됨

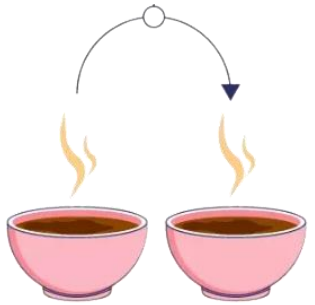
```
1.  void foo(int y) {
2.      cout << "y = " << y << '\n';
3.  }
4.
5.  int main() {
6.      foo(5);   // first call
7.
8.      int x = 6;
9.      foo(x);   // second call
10.     foo(x + 1); // third call
11.
12.     return 0;
13. }
```

```
y = 5
y = 6
y = 7
```

[예시 2] 주소에 의한 호출, Call by address

- **주소에 의한 호출 예시**
  - 함수 foo()는 **포인터 매개 변수 ptr**을 통해 인수의 값을 변경함

```
1.  void foo(int* ptr){
2.      *ptr = 6;
3.  }
4.
5.  int main() {
6.      int value = 5;
7.
8.      cout << "value = " << value << '\n';
9.      foo(&value);
10.     cout << "value = " << value << '\n';
11.     return 0;
12. }
```



```
value = 5
value = 6
```

  - 주소에 의한 호출은 주로 **배열**과 함께 사용한다.

```
1. void printArray(int* array, int length) {
2.     for (int index = 0; index < length; ++index)
3.         cout << array[index] << ' ';
4. }
5.
6. int main() {
7.     int array[6] = { 6, 5, 4, 3, 2, 1 };
8.     printArray(array, 6);
9. }
```

```
6 5 4 3 2 1
```

객체 치환 및 객체 리턴

- **객체 치환(Assignment)**
  - 연산자를 이용하여 새로운 객체에 원본 객체를 대입하여 사용 가능함
  - 객체의 모든 데이터가 비트 단위로 복사되며, 동일한 클래스 타입에 대해서만 적용됨

```
1. BankAccount m1(123456);
2. BankAccount m2(500);
3. m1 = m2; // c1의 값을 500으로 변경
```

- **함수의 객체 리턴**
  - 생성된 복사본을 반환값으로 넘겨줌

```
1. BankAccount getBankAccount(){
2.     BankAccount tmp(999);
3.         return tmp;
4. }
5.
6. BankAccount m1; // c의 반지름은 10
7. m = getBankAccount(); // tmp 객체의 복사본이 c로 치환, c의 값은 999
```

참조 변수 [1/2]

- **일반적인 변수 타입**
  - 일반 변수(Normal variable): 직접 값을 보유함
  - 포인터(Pointer): 다른 값의 주소(또는 null)를 보유함
- **참조 변수(Reference variable)**
  - 참조 변수는 다른 객체 또는 값의 별칭으로 사용되는 C++ 타입임
  - 자료형 뒤에 **앰퍼샌드(&)**를 사용하여 선언함 → **자료형& 별명 = 기존 변수명;**
  - &는 주소(Address)를 의미하지 않고 **참조(Reference)**를 의미한다

```
1. int main() {
2.     int value = 5; // normal integer
3.     int& ref = value; // reference to variable value
4.
5.     value = 6; // value is now 6
6.     ref = 7;   // value is now 7
7.
8.     cout << value << endl; // prints 7
9.     ++ref;
10.    cout << value << endl; // prints 8
11.
12.    cout << &value << endl;
13.    cout << &ref << endl;
14.
15.    return 0;
16.}
```

```
7
8
005FFDD4
005FFDD4
```

참조 변수 [2/2]

- **함수 매개 변수**
  - 참조형은 **함수 매개 변수**로 가장 많이 사용된다.

```
1. // ref is a reference to the argument passed in, not a copy
2. void changeN(int& ref) {
3.     ref = 6;
4. }
5.
6. int main() {
7.     int n = 5;
8.
9.     cout << n << '\n';
10.    changeN(n); // note that this argument does not need to be a reference
11.    cout << n << '\n';
12.    return 0;
13.}
```

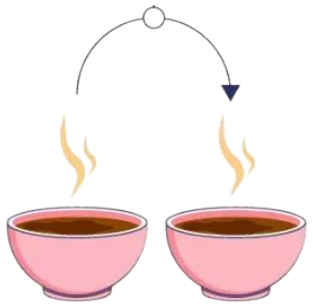
```
5
6
```

  - 인수 n이 함수에 전달되면 함수 매개변수 ref가 인수 n에 대한 참조로 설정된다
  - 이것은 함수가 ref를 통해 n의 값을 변경할 수 있게 한다.
  - 변수 n 자체가 참조형일 필요는 없다.

[예시 3] 참조에 의한 호출, Call by reference

- **참조에 의한 호출 예시**
  - 변수를 참조로 전달하려면 함수 매개 변수를 일반 변수가 아닌 **참조로 선언**해야 한다.
  - 참조는 변수 자체와 똑같이 취급되므로 참조에 대한 모든 변경 사항은 **인수에도 적용**된다.

```
1. void addOne(int& y) { // 함수가 호출되면 y는 인수에 대한 참조
2.     y = y + 1;
3. }
4.
5. void foo(int& value) {
6.     value = 6;
7. }
8.
9. int main() {
10.     int value = 5;
11.
12.     cout << "value = " << value << '\n';
13.     foo(value);
14.     cout << "value = " << value << '\n';
15.     return 0;
16.}
```



```
value = 5
value = 6
```

참조 리턴

- **C 언어의 함수 리턴**
  - C언어에서 함수가 리턴하도록 허용된 것은 오직 **값**뿐
  - 값에는 void를 포함하여 정수(int), 문자(char), 실수(float) 등의 기본 타입의 값과 주소(포인터)
- **C++의 함수 리턴**
  - 함수는 값 외에 참조 리턴이 가능함
  - 참조 리턴: 변수 등과 같이 현존하는 공간에 대한 참조 리턴
  - 변수의 값을 리턴하는 것이 아님

```
1.  char c = 'a';
2.
3.  char get() {  // char 값 리턴
4.      return c; // 변수 c의 값 리턴
5.  }
6.
7.  char a = get(); // a = 'a'가 됨
8.
9.  get() = 'b' // 컴파일 오류
```

문자 값을 리턴하는 get()

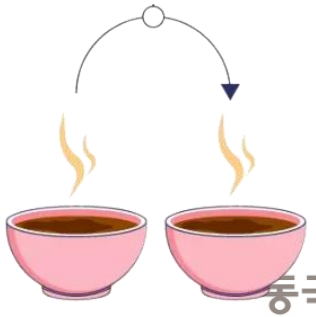
```
1. char c = 'a';
2.
3. char& find() { // char 타입의 참조 리턴
4.     return c; // 변수 c에 대한 참조 리턴
5. }
6.
7. char a = find(); // a = 'a'가 됨
8.
9. char &ref = find() // ref는 c에 대한 참조
10.ref = 'M': // c
11.find() = 'b';
```

char 타입의 참조를 리턴하는 find()

세 가지 호출 차이점 비교

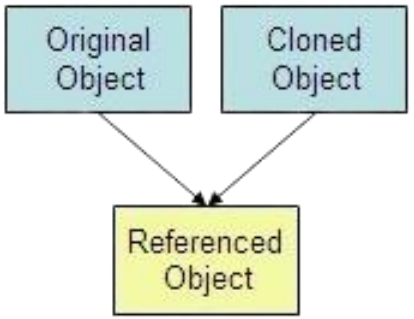
- **객체 전달 정리**
  - Call by Address는 Call by Value와 유사하지만 "Value"가 주소임
  - Call by Value 형태지만 Value가 주소이므로 Call by Address로 지칭함
  - 큰 맥락으론 Call by Reference와 Call by Value로 구분함
- **차이점 정리**

	Call by Value	Call by Reference
1	복사한 값을 전달함 → 안전함	원본 주소를 전달함 → 빠름
2	함수에서 변경한 내용이 다른 함수에 적용되지 않음	함수에서 변경한 내용이 다른 함수에도 적용됨
3	인자를 다루는 메모리 위치가 다름	인자를 다루는 메모리 위치가 같음
4	메모리 사용량이 크게 늘어날 수 있음	직접 참조하기 때문에 원본 값이 영향을 받음

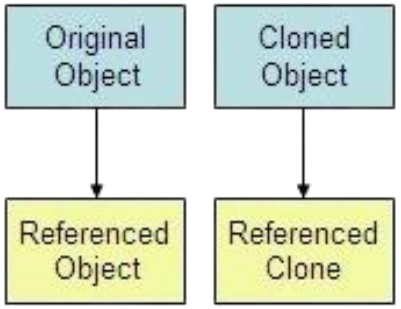


얕은 복사와 깊은 복사

- **얕은 복사(Shallow Copy)**
  - 객체 복사 시, 객체의 멤버를 1:1로 복사
  - 객체의 멤버 변수에 동적 메모리가 할당된 경우
    - 사본은 원본 객체가 할당 받은 메모리를 공유하는 문제 발생



- **깊은 복사(Deep copy)**
  - 객체 복사 시, 모든 객체의 멤버를 1:1로 복사
  - 객체의 멤버 변수에 동적 메모리가 할당된 경우
    - 사본은 원본이 가진 메모리 크기 만큼 별도로 동적 할당
    - 원본의 동적 메모리에 있는 내용을 사본에 복사
  - 완전한 형태의 복사
    - 사본과 원본은 메모리를 공유하는 문제 없음



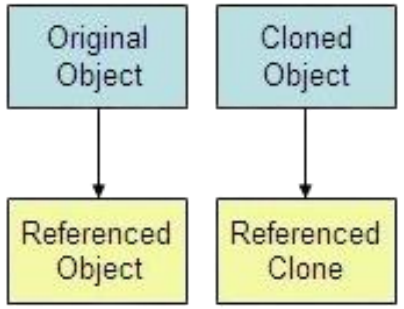
➢ 깊은 복사로 객체를 복사하면 **완전한 독립성**을 가질 수 있음

복사 생성자(Copy Constructor)

- **복사 생성자**
  - 한 객체의 내용을 다른 객체로 복사하여 생성된 생성자
  - 일반적인 생성자와 동일한데 그 내용이 다른 객체를 복사하여 생성된 것임
- **주요 특징**
  - 자신과 같은 클래스 타입의 객체를 인자로 받음
  - 복사 생성자가 정의되어 있지 않다면 디폴트 복사 생성자가 자동으로 생성됨
  - 매개 변수는 오직 하나로 자기 클래스에 대한 참조로 선언됨
- **선언 및 실행**

```
1. class Dongguk {
2.     Dongguk(const Dongguk& d); // 복사 생성자 선언
3.     ...
4. };
5.
6. Dongguk::Dongguk(const Dongguk& d) { // 복사 생성자 구현
7.     ...
8. }
9. ...
10.Dongguk src(159); // 보통 생성자 호출
11.Dongguk dest(src); // src 객체를 복사하여 dest 객체 생성, 복사 생성자 호출
```

자기 클래스에 대한 참조 매개 변수

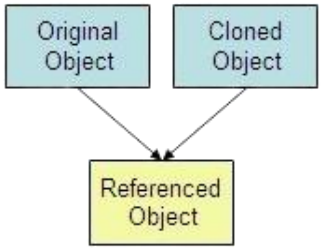


디폴트 복사 생성자(Default Copy Consructor)

- **디폴트 복사 생성자**
  - 복사 생성자가 정의되어 있지 않다면, 컴파일러는 오류로 처리하는 대신에

    다음과 같은 디폴트 복사 생성자를 묵시적으로 삽입하고 생성자를 호출하도록 컴파일함

```
1. class Dongguk {
2.     int elephant;
3. public:
4.     Dongguk(int n);
5.     double getNum();
6. };
7. ...
8. Dongguk dest(src); // 복사 생성자. Dongguk(const Dongguk&) 호출?
```




```
1. Dongguk::Dongguk(const Dongguk& d) {
2.     this->elephant = d.elephant;
3.     // 원본 객체 d의 각 멤버를 사본(this)에 복사한다.
4.     cout << "복사 생성자 실행 elephant = " << elephant << endl;
5. }
```

  - 컴파일러가 삽입하는 디폴트 복사 생성자 코드는 얕은 복사를 실행하도록 만들어진 코드임
    - 사본은 원본 객체가 할당 받은 메모리를 공유하는 문제가 발생함

연습문제(5장)

1. C++의 함수 인자 전달 방식이 아닌 것은?
   ① 값에 의한 호출 ② 주소에 의한 호출
   ③ 참조에 의한 호출 ④ 묵시에 의한 호출

2. 일반적으로 함수 호출 시 가장 비용(cost) 부담이 큰 것은?
   ① 값에 의한 호출 ② 주소에 의한 호출
   ③ 참조에 의한 호출 ④ 묵시에 의한 호출

3. 다음에서 f() 함수가 호출될 때 사용되는 인자 전달 방식은 무엇인가?

```
void f(int n[]);
int main() {
   int m[3]={1,2,3};
   f(m);
}
```

연습문제(5장)

4. 다음 두 함수 선언은 같은 것인가?

(1) `void f(int p[]);`와 `void f(int *p);`

(2) `void f(int *p);`와 `void f(int &p);`

5. 다음 프로그램의 실행 결과는 무엇인가?

(1)

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

void square(int n) {
   n = n*n;
}

int main() {
   int m = 5;
   square(m);
   cout << m;
}
```

(2)

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

void square(int &n) {
   n = n*n;
}

int main() {
   int m = 5;
   square(m);
   cout << m;
}
```

연습문제(5장)

6. 다음 프로그램의 실행 결과는 무엇인가?

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

void square(int n[], int size) {
   for(int i=0; i<size; i++) n[i] = n[i]*n[i];
}

int main() {
   int m[3] = {1,2,3};
   square(m, 3);
   for(int i=0; i<3; i++) cout << m[i] << ' ';
}
```

7. char 형 변수 c가 선언되어 있을 때, 참조 변수 r의 선언 중 틀린 것은?

① char & r = c; ② char r & = c; ③ char& r = c; ④ char &r = c;

연습문제(5장)

8. 변수 c에 'a' 문자를 기록하지 못하는 것은?

```
char c;
char *p = &c;
char q = c;
char &r = c;
```

① c = 'a';　② q = 'a';　③ r = 'a';　④ *p = 'a';

9. 다음 중 컴파일 오류가 발생하는 문장은?

```
int n = 10;
int &refn;          // ①
refn = n;           // ②
refn++;             // ③
int & m = refn;     // ④
```

10. 다음의 각 문제가 별도로 실행될 때 array 배열은 어떻게 되는가?

```
int array[]={0,2,4,6,8,10,12,14,16,18};
int& f(int n) {
   return array[n];
}
```

(1) f(9) = 100;　(2) for(int i=1; i<9; i++) f(i) = f(i) + 2;

(3) int v = f(0); v=100;　(4) f(f(2)) = 0;

연습문제(5장)

11. 다음 copy() 함수는 src 값을 dest에 복사하는 함수이다.

```
void copy(int dest, int src) {
   dest = src;
}
```

copy()를 이용하여 b 값을 a에 복사하고자 하지만, b 값이 a에 복사되지 않는다.

```
int a=4, b=5;
copy(a, b); // b 값을 a에 복사
```

복사되지 않는 이유가 무엇인지 설명하고, 복사가 잘 되도록 copy() 함수만 고쳐라.

연습문제(5장)

12. 비슷하게 생긴 다음 두 함수가 있다.

```
int& big1(int a, int b) {
   if(a > b) return a;
   else return b;
}
int& big2(int& a, int& b) {
   if(a > b) return a;
   else return b;
}
```

다음 코드를 실행하였을 때, x, y의 값이 어떻게 변하는지 예측하고, 그 이유를 설명하라.

```
int x=1, y=2;
int& z = big1(x, y);
z = 100;
int& w = big2(x, y);
w = 100;
```

연습문제(5장)

13. MyClass 클래스의 기본 생성자(디폴트 생성자)와 복사 생성자의 원형은 무엇인가?

14. 클래스 MyClass가 있다고 할 때, 복사 생성자가 필요한 경우가 아닌 것은?

① `MyClass a = f(); // f()가 MyClass 객체를 리턴하는 경우`

② `void f(MyClass *p);`

③ `MyClass b = a; // a는 MyClass 타입`

④ `MyClass b(a); // a는 MyClass 타입`

연습문제(5장)

15. 다음 클래스에 대해 물음에 답하여라.

```
class MyClass {
   int size;
   int *element;
public:
   MyClass(int size) {
      this->size = size;
      element = new int [size];
      for(int i=0; i<size; i++) element[i] = 0;
   }
};
```

(1) 적절한 소멸자를 작성하라.

(2) 컴파일러가 삽입하는 디폴트 복사 생성자 코드는 무엇인가?

(3) MyClass에 깊은 복사를 실행하는 복사 생성자 코드를 작성하라.

연습문제(5장)

16. 복사 생성자에 대해 설명한 것 중 틀린 것은?

① 복사 생성자는 중복 가능하여 필요에 따라 여러 개 선언될 수 있다.
② 복사 생성자가 선언되어 있지 않는 경우, 컴파일러가 디폴트 복사 생성자를 삽입한다.
③ 디폴트 복사 생성자는 얕은 복사를 실행한다.
④ 포인터 멤버가 없는 경우 디폴트 복사 생성자는 거의 문제가 되지 않는다.

17. 다음 클래스에서 컴파일러가 삽입하는 디폴트 복사 생성자는 무엇인가?

```
class Student {
   string name;
   string id;
   double grade;
};
```

연습문제(5장)

18. 다음 클래스에서 컴파일러가 삽입하는 디폴트 복사 생성자는 무엇인가?

```
class Student {
   string *pName;
   string *pId;
   double grade;
};
```

19. 문제 15의 클래스에 대해 다음 치환문이 있다면 어떤 문제가 발생하는가?

```
int main() {
   MyClass a(5), b(5);
   a = b; // 여기
}
```

실습과제(5장)

1. 두 개의 Circle 객체를 교환하는 swap()함수를 '참조에 의한 호출'이 되도록 작성하고 호출하는 프로그램을 작성하라.

2. 다음 main()함수와 실행 결과를 참고하여 half()함수를 작성하라.

```
int main() {
    double n=20;
    half(n); // n의 반값을 구해 n을 바꾼다.
    cout << n; // 10이 출력된다.
}
```

```
10
```

3. 다음과 같이 작동하도록 combine() 함수를 작성하라.

```
int main () {
    string text1("I love you"), text2("very much");
    string text3; // 비어 있는 문자열
    combine(text1, text2, text3); 1/ text1과 " ", 그리고 text2를 덧붙여 text3 만들기
    cout << text3; // "I love you very much" 출력
}
```

```
I love you very much
```

실습과제(5장)

4. 아래와 같이 원형이 주어진 bigger()를 작성하고 사용자로부터 2개의 정수를 입력받아 큰 값을 출력하는 main()을 작성하라. bigger()는 인자로 주어진 a, b가 같으면 true, 아니면 false를 리턴하고 큰 수는 big에 전달한다.

```
bool bigger(int a, int b, int& big);
```

5. 다음 Circle 클래스가 있다.

```
class Circle {
    int radius;
public:
    Circle(int r) {radius = r;}
    int getRadius () { return radius; }
    void setRadius (int r) { radius = r; }
    void show() { cout <<"반지름이 "<< radius << "인 원" << endl; }
}
```

실습과제(5장)

5. (앞 슬라이드에 이어) Circle 객체 b를 a에 더하여 a를 키우고자 다음 함수를 작성하였다.

```
void increaseBy (Circle a, Circle b) {
    int r = a.getRadius() + b.getRadius();
    a.setRadius(r);
}
```

다음 코드를 실행하면 increaseBy() 함수는 목적대로 실행되는가?

```
int main () {
    Circle x(10), y(5);
    increaseBy(x, y); // x의 반지름이 15인 원을 만들고자 한다.
    x.show(); // "반지름이 15인 원"을 출력한다.
}
```

main() 함수의 목적을 달성하도록 increaseBy() 함수를 수정하라.

실습과제(5장)

6. find() 함수의 원형은 다음과 같다. 문자열 a에서 문자 c를 찾아, 문자 c가 있는 공간에 대한 참조를 리턴한다. 만일 문자 c를 찾을 수 없다면 success 참조 매개 변수에 false를 설정한다. 물론 찾게 되면 success에 true를 설정한다.

```
char& find (char a[], char c, bool& success);
```

다음 main()이 잘 실행되도록 find()를 작성하라.

```
int main() {
    char s[] = "Mike";
    bool b = false;
    char& loc = find (s, 'M', b);
    if(b == false) {
        cout <<"M을 발견할 수 없다"<< endl;
       return 0;
    }
    loc = 'm'; // 'M' 위치에 'm' 기록
    cout << s << endl; // "mike"가 출력됨
}
```

```
mike
```

실습과제(5장)

7. 다음과 같이 선언된 정수를 저장하는 스택 클래스 MyIntStack을 구현하라. MyIntStack 스택에 저장할 수 있는 정수의 최대 개수는 10이다.

```
class MyIntStack {
    int p[10]; // 최대 10개의 정수 저장
    int tos; // 스택의 꼭대기를 가리키는 인덱스
public:
    MyIntStack();
    bool push(int n); // 정수 n 푸시. 꽉 차 있으면 false, 아니면 true 리턴
    bool pop(int &n); // 팝하여 n에 저장. 스택이 비어 있으면 false, 아니면 true 리턴
};
```

MyIntStack 클래스를 활용하는 코드와 실행 결과는 다음과 같다.

```
int main () {
    MyIntStack a;
    for(int i=0; i<11; i++) {    // 11개를 푸시하면, 마지막에는 stack full이 된다.
        if(a.push(i)) cout << i << ' '; // 푸시된 값 에코
        else cout << endl << i+1 << " 번 째 stack full" << endl;
    }
    int n;
    for(int i=0; i<11; i++) {   // 11개를 팝하면, 마지막에는 stack 가 된다.
        if(a.pop(n)) cout << n <<  ' '; // 팝 한 값 출력
        else cout << endl << i+1 << " 번 째 stack empty";
    }
    cout << endl;
}
```

```
0 1 2 3 4 5 6 7 8 9

11 번 째 stack full

9 8 7 6 5 4 3 2 1 0

11번 째 stack empty
```

실습과제(5장)

8. 문제 7번의 MyIntStack를 수정하여 다음과 같이 선언하였다. 스택에 저장할 수 있는 정수의 최대 개수는 생성자에서 주어지고 size 멤버에 유지한다. MyIntStack 클래스를 작성하라.

```
class MyIntStack {
    int *p; // 스택 메모리로 사용할 포인터
    int size; // 스택의 최대 크기
    int tos; // 스택의 탑을 가리키는 인덱스
public:
    MyIntStack();
    MyIntStack(int size);
    MyIntStack(const MyIntStack& s); // 복사 생성자
    ~MyIntStack();
    bool push(int n); // 정수 n을 스택에 푸시한다.
                      // 스택이 꽉 차 있으면 false를, 아니면 true 리턴
    bool pop(int &n); // 스택의 탑에 있는 값을 n에 팝한다.
                      // 만일 스택이 비어 있으면 false를, 아니면 true 리턴
};
```

실습과제(5장)

8. (앞 슬라이드에 이어)MyIntStack 클래스를 활용하는 코드와 실행 결과는 다음과 같다.

```
int main () {
    MyIntStack a(10);
    a. push (10);
    a. push (20);
    MyIntstack b = a; // 복사 생성
    b.push (30);

    int n;
    a.pop(n);   // 스택 a 팝
    cout < "스택 a에서 팝한 값 "<< n << endl;
    b.pop(n); // 스택 b 팝
    cout << "스택 b에서 팝한 값 "<< n << endl;
}
```

```
스택 a에서 팝한 값 20
스택 b에서 팝한 값 30
```

실습과제(5장)

9. 클래스 Accumulator는 add() 함수를 통해 계속 값을 누적하는 클래스로서, 다음과 같이 선언된다. Accumulator 클래스를 구현하라.

```
class Accumulator {
    int value;
public:
    Accumulator(int value); // 매개 변수 value로 멤버 value를 초기화한다.
    Accumulator& add(int n); // value에 n을 더해 값을 누적한다.
    int get(); // 누적된 값 value를 리턴한다.
};
```

Accumulator는 다음과 같이 main() 함수에 의해 활용된다.

```
int main () {
    Accumulator acc (10);
    acc.add(5).add(6).add(7); // acc의 value 멤버가 280| 된다.
    cout << acc.get(); // 28 출력
}
```

```
28
```

실습과제(5장)

10. 참조를 리턴하는 코드를 작성해보자. 다음 코드와 실행 결과를 참고하여 append() 함수를 작성하고 전체 프로그램을 완성하라. append()는 Buffer 객체에 문자열을 추가하고 Buffer 객체에 대한 참조를 반환하는 함수이다.

```
class Buffer {
    string text;
public:
    Buffer (string text) { this->text = text; }
    void add(string next) { text += next; } // text에 next 문자열 덧붙이기
    void print() { cout << text << endl; }
};
int main() {
    Buffer buf ("Hello");
    Buffer& temp = append (buf, "Guys");  // buf의 문자열에 "Guys" 덧붙임
    temp.print(); // "HelloGuys" 출력
    buf.print(); // "HelloGuys" 출력
}
```

```
HelloGuys
HelloGuys
```

실습과제(5장)

11) 책의 이름과 가격을 저장하는 다음 Book 클래스에 대해 물음에 답하여라.

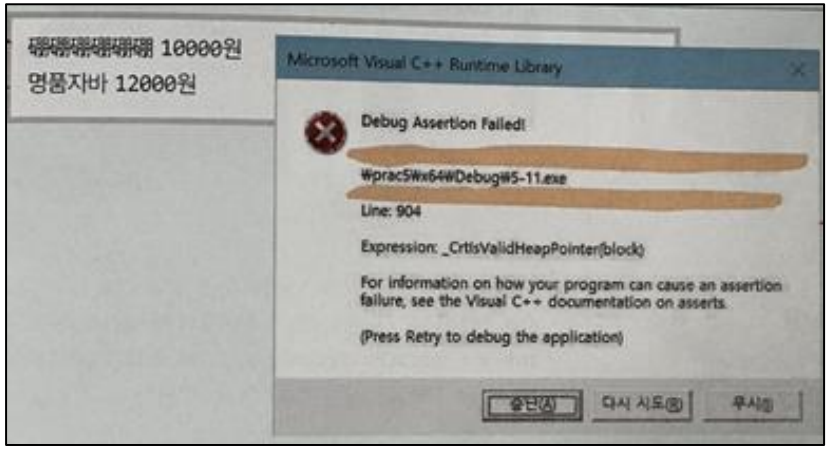
```
class Book {
    char *title; // 제목 문자열
    int price; // 가격
public:
    Book(const char* title, int price);
    ~Book();
    void set(char* title, int price);
    void show() { cout << title << ' ' << price << "원" << endl; }
};
```

1) Book 클래스의 생성자, 소멸자, set() 함수를 작성하라. set() 함수는 멤버 변수 title에 할당된 메모리가 있으면 먼저 반환한다. 그러고 나서 새로운 메모리를 할당받고 이곳에 매개 변수로 전달받은 책이름을 저장한다.
2) 컴파일러가 삽입하는 디폴트 복사 생성자 코드는 무엇인가?
3) 디폴트 복사 생성자만 있을 때 아래 main() 함수는 실행 오류가 발생한다.

```
int main ( ) {
    Book cpp("명품C++", 10000);
    Book java = cpp; // 복사 생성자 호출됨
    java.set ("명품자바", 12000);
    cpp. show);
    java. show();
}
```


실습과제(5장)

(앞 슬라이드에 이어)



다음과 같이 실행 오류가 발생하지 않도록 깊은 복사 생성자를 작성하라.

명품C++ 10000원
명품자바 12000원

정상적인 실행

4) 문제 3)에서 실행 오류가 발생하는 원인은 Book 클래스에서 C-스트링(char* title) 방식으로 문자열을 다루었기 때문이다. 복사 생성자를 작성하지 말고 문자열을 string 클래스를 사용하여, 문제 3)의 실행 오류가 발생하지 않도록 Book 클래스를 수정하라. 이 문제를 풀고 나면 문자열을 다룰 때, string을 사용해야하는 이유를 명확히 알게 될 것이다.

실습과제(5장)

12. 다음은 학과를 나타내는 Dept 클래스와 이를 활용하는 main()을 보여 준다.

```
class Dept {
    int size; // Scores 배열의 크기
    int* scores; // 동적 할당 받을 정수 배열의 주소
public:
    Dept(int size) { // 생성자
        this->size = size;
        scores = new int[size];
    }
    Dept(const Dept& dept); // 복사 생성자
    ~Dept(); // 소멸자
    int getSize() { return size; }
    void read(); // size 만큼 키보드에서 정수를 읽어 scores 배열에 저장
    bool isOver60(int index); // index의 학생의 성적이 60보다 크면 true 리턴
};
```

실습과제(5장)

(앞 슬라이드에 이어)

```
int countPass(Dept dept) { // dept 학과에 60점 이상으로 통과하는 학생의 수 리턴
    int count = 0;
    for (int i = 0; i < dept. getSize(); i++) {
        if (dept.isOver60(i)) count++;
    }
    return count;
}

int main () {
    Dept com(10); // 총 10명이 있는 학과 com
    com.read(); // 총 10명의 학생들의 성적을 키보드로부터 읽어 scores 배열에 저장
    int n= countPass(com); // com 학과에 60점 이상으로 통과한 학생의 수를 리턴
    cout <<"60점 이상은 "<< n << "명" ;
}
```

1) main()의 실행 결과가 다음과 같이 되도록 Dept 클래스에 멤버들을 모두 구현하고, 전체 프로그램을 완성하라.

```
10개 점수 입력>> 10 20 30 40 50 60 70 80 90 100
60점 이상은 4명
```

실습과제(5장)

(앞 슬라이드에 이어)

2) Dept 클래스에 복사 생성자 Dept(const Dept& dept);가 작성되어 있지 않은 경우, 컴파일은 되지만 프로그램 실행 끝에 실행 시간 오류가 발생한다 (복사 생성자를 뺀 채 실행해보라). 위의 코드 어느 부분이 실행될 때 복사 생성자가 호출되는지 설명하고, 복사 생성자가 없으면 왜 실행 오류가 발생하는지 설명하라.

3) Dept 클래스에 복사 생성자를 제거하라. 복사 생성자가 없는 상황에서도 실행 오류가 발생하지 않게 하려면 어느 부분을 수정하면 될까? 극히 일부분의 수정으로 해결된다. 코드를 수정해보라.