기초 C++ 프로그래밍 #2

More about OOP

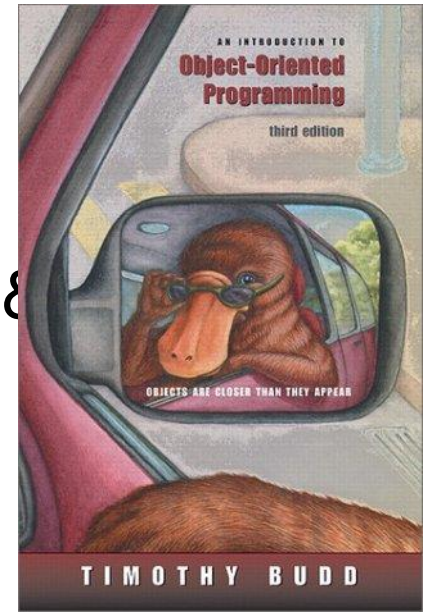
이전에 다루지 않았던 OOP에 대한 새로운 개념을 공부한다.

추상화(Abstraction)

재정의(Overriding)

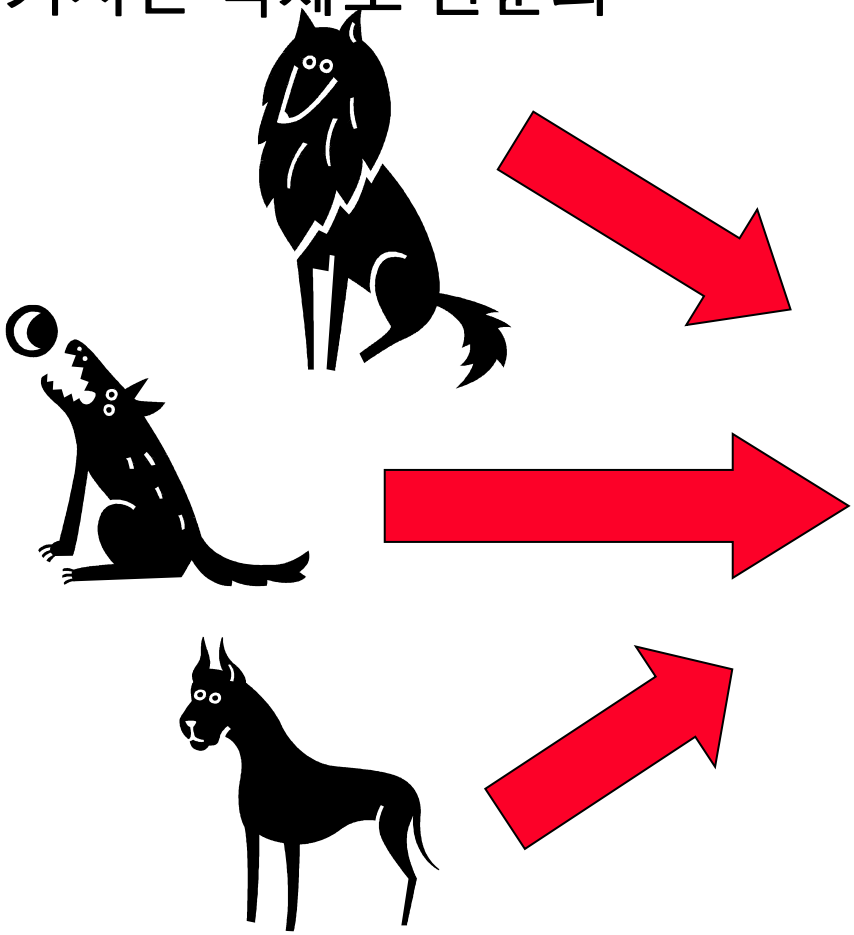
다형성(Polymorphism)

캡슐화 & 정보 은폐 & 인터페이스
(Encapsulation & Information Hiding &



추상화

실세계의 복잡한 객체의 형태를 몇몇 상태와 행동을 가지는 객체로 단순화



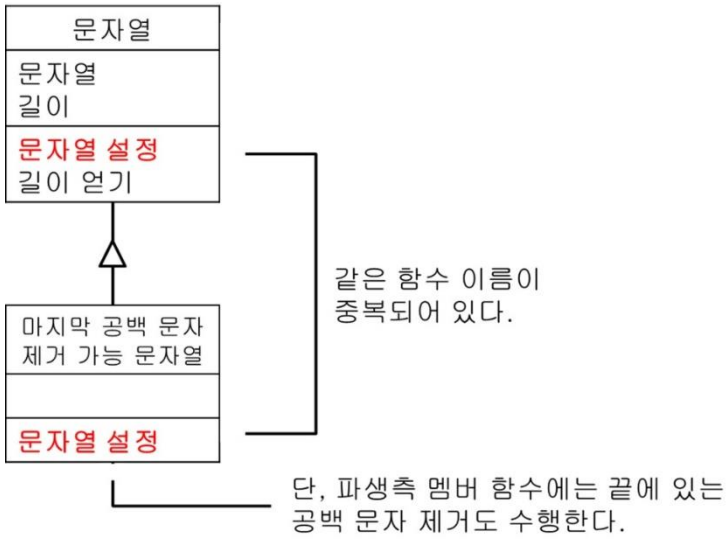
```
class Dog
{
  private:
    leg leg1, leg2, leg3, leg4;
    fur furs;

  public:
    void bark();
    void run();
    void sleep();
};
```

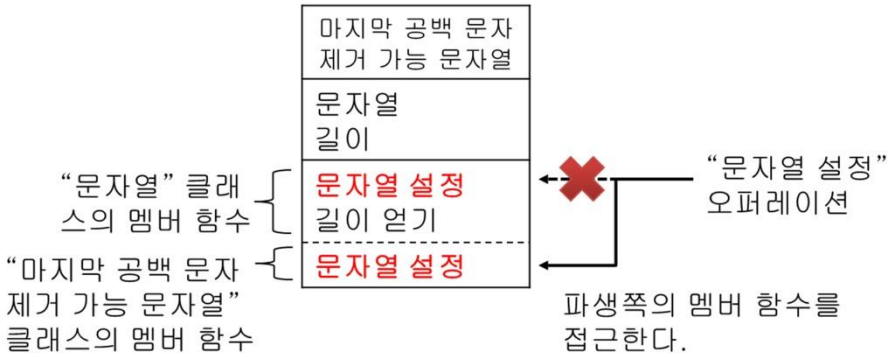
재정의

상속 시, 기존 클래스의 함수 처리 내용을 변경 – 커스터마이즈(Customize) 효과

기반 클래스의 멤버 함수와 완전히 같은 이름의 멤버 함수를 파생 클래스에서 만듦으로써 재정의



함수 재정의의 예



파생 클래스에서 재정의한 함수의 동작

다형성

Polymorphism = Poly(많은) + Morph(형태) + ~ism

하나의 인터페이스를 사용하여 여러 형태의 데이터 타입, 또는 함수들을 사용할 수 있게 하는 특징

C++은 프로그래밍 언어에서의 다음 세 가지 다형성을 모두 제공

- 파라미터적 다형성(Parametric Polymorphism)
- 서브타입 다형성(Subtype Polymorphism)
- 애드혹 다형성(Ad-hoc Polymorphism)

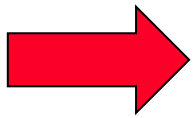
OOP의 다형성은 보통 서브타입 다형성을 의미함

파라미터적 다형성

파라미터적 다형성은 자료형을 고려할 필요 없이 사용할 수 있는 일반화된 코드를 작성할 수 있게 해 준다.

자료형만 달라서 모든 코드를 다시 작성해야 하는 문제를 해결해 준다.

```
class Stack_Int{
public:
    void push(int item);
    int pop();
    ...
};
```



```
template <class T>
class Stack{
public:
    void push(T item);
    T pop();
    ...
};
```

int형만 저장 가능

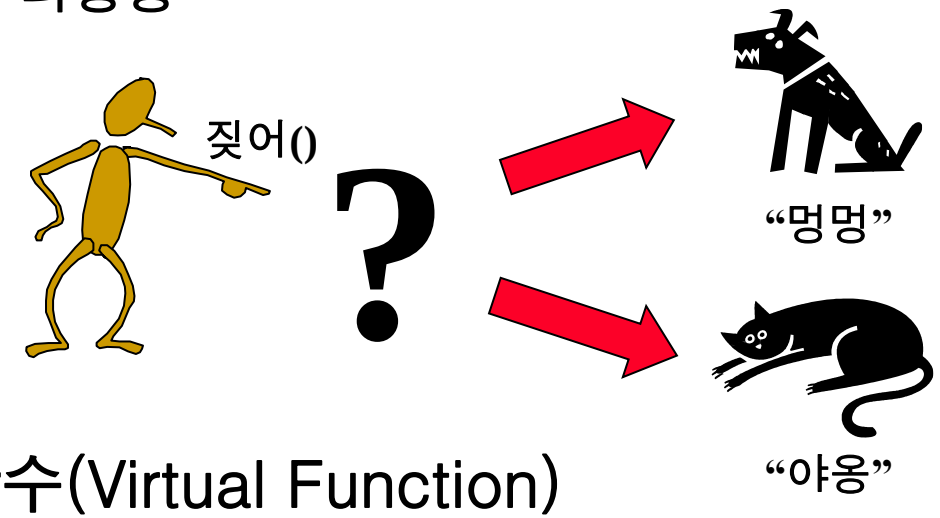
T에 자료형을 지정함으로써
다양한 자료형에 대해 사용 가능

서브타입 다형성

일반적으로 OOP에서의 다형성을 뜻하는 개념

같은 메시지에 대해서도 객체가 자신의 실체와 상태에 따라 스스로 다른 행위를 하는 것이 기본 개념

상속 관계에서 기반 클래스를 통해 파생 클래스를 사용함으로써 얻어지는 다형성



가상 함수(Virtual Function)

포인터의 정적 타입(포인터의 자료형)이 아닌 동적 타입(포인터가 가리키는 객체의 자료형)을 따르는 함수

서브타입 다형성

```
class Animal{
public
   void talk(){};
};
class Cat:public Animal{
public:
   void talk(){cout<<“Meow”<<endl;};
};
class Dog:public Animal{
public:
   void talk(){cout<<“Woof”<<endl;};
};
int main(){
   Animal *a=new Cat();
   a->talk(); // Animal::talk() 호출
   a=new Dog();
   a->talk(); // Animal::talk() 호출
   return 0;
}
```

출력:

```
class Animal{
public
   virtual string talk(){}; // talk()는 가상 함수
};
class Cat:public Animal{
public:
   virtual void talk(){cout<<“Meow”<<endl;};
};
class Dog:public Animal{
public:
   virtual void talk(){cout<<“Woof”<<endl;};
};
int main(){
   Animal *a=new Cat();
   a->talk(); // Cat::talk() 호출
   a=new Dog();
   a->talk(); // Dog::talk() 호출
   return 0;
}
```

출력:
Meow
Woof

애드혹 다형성

약한 다형성: 하나의 인터페이스(개체)가 여러 가지로 사용되는 것이 아니라, 미리 준비된 다수의 함수들(이름은 같고 인수만 다름) 중 하나를 선택하여 호출하는 방식으로 작동

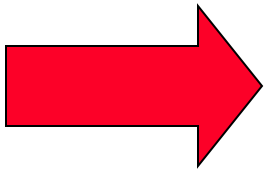
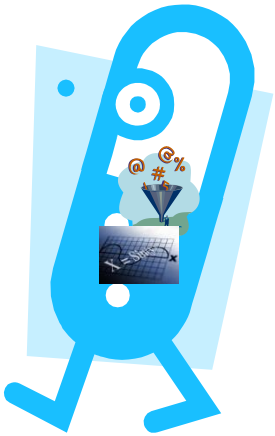
C++의 다중정의(Overloading)으로 구현됨

- 함수 다중정의(Function Overloading)
- 연산자 다중정의(Operator Overloading)
  - 지난 시간에 공부한 내용

캡슐화 & 정보 은폐 & 인터페이스

캡슐화란, 데이터와 해당 데이터를 조작할 수 있는 함수를 하나로 묶는 것을 의미한다. 이때 객체는 두 가지를 감싸는 캡슐이 된다.

C++에서는 클래스를 이용하여 구현

OOP의 기본 철학은 '겉만 보고 속은 숨긴다'이다.

즉, 객체의 내부의 구성을 몰라도, 공개된 인터페이스만으로도 해당 객체를 사용할 수 있게 하여 모듈성(modularity)과 재사용성(reusability)를 극대화

C++에서는 접근 지정자로 이를 구현

public : 외부에서 자유롭게 접근 가능

private : 해당 클래스의 멤버 함수만이 접근가능

protected : private와 같지만, 상속된 클래스의 멤버함수도 접근가능

4주차 실습 안내
(C++ Programming #2)

4주차 실습

CPP-2 다형성의 이해 부분 해결

OOP의 개념 중 1)파라미터적 다형성, 2)서브타입 다형성을 구현해본다

LinkedList와 Stack을 구현해본다.

CPP-2: 다형성의 이해

OOP의 여러 개념 중 앞에서 배운 파라미터적 다형성, 서브타입 다형성을 실습

실습 개요

파라미터적 다형성은 템플릿 클래스를 통해 달성되고, 서브타입 다형성은 상속 관계에서 기반 클래스를 통해 파생 클래스를 접근함으로써 구현할 수 있다.

따라서 제공되는 LinkedList 클래스(소스 코드 제공)를 템플릿을 사용하여 확장하고, 또한 LinkedList 클래스를 기반 클래스로 하는 파생 클래스 Stack을 구현하여 서브타입 다형성 예제를 실습한다.

LinkedList 클래스의 Print() 함수는 실행 예와 같이 작동할 수 있도록 직접 구현하여 본다. (이 함수만 코드가 제공되지 않음)

CPP-2: 다형성의 이해

먼저 다음 슬라이드에서 주어질 LinkedList 클래스를 파라미터적 다형성을 지원하게 하기 위해 템플릿(Template) 클래스로 확장

함수 또는 클래스에서 임의의 자료형이 사용될 때, 그 앞에
template <class 자료형이름>
을 명시하여 준다.

템플릿 함수와 클래스는 헤더 파일에 모두 기술하여야 한다.

```
template <class T>
T add(T a, T b)
{
        return a+b;
}
```

Ex) 템플릿 함수

```
template <class T>
class Stack{
public:
   void push(T item);
   T pop();
   ...
};

template <class T>
void push(T item){
   함수 본체;
}
```

Ex) 템플릿 클래스

CPP-2: 다형성의 이해

먼저 템플릿 선언 template <typename T>

T라는 타입에 대해 템플릿을 선언한다는 뜻

여러 타입에 대한 템플릿을 만들고 싶으면 template <typename T1, typename T2 …>

T는 모든 타입을 대변하는 이름이다. myFunc()을 호출하면 호출부의 인수의 타입을 읽어 그 타입에 맞는 함수를 (컴파일시 컴파일러가) 자동으로 작성한다.

```
1  #include <iostream>
2  using namespace std;
3
4
5  template <typename T>
6  T myFunc(T num1, T num2) {
7      return num1 + num2;
8  }
9
10 int main() {
11     cout << myFunc(1,3) << endl;
12     cout << myFunc(1.45, 3.5) << endl;
13
14 }
15
```

상속 받는 경우 ex) template <class T>

class 상속 받는 함수명 : public 상속하는 함수명<T> {…};

LinkedList 클래스 (Queue와 동일하게 작동함)

```
// Linked List Node
class Node{
public:
    int data;
    Node *link;
    Node(int element){
        data = element;
        link = 0; }
};

// Linked List Class
class LinkedList
{
protected:
    Node *first;
    int current_size;
public:
    LinkedList(){
        first = 0;
        current_size = 0; };
    int GetSize() { return current_size; }; // 노드 개수를 리턴
    void Insert(int element); // 맨 앞에 원소를 삽입
    virtual bool Delete(int &element); // 맨 뒤의 원소를 삭제
    void Print(); // 리스트를 출력
};
```

```
void LinkedList::Insert(int element){ // 새 노드를 맨 앞에 붙임
    Node *newnode = new Node(element);
    newnode->link = first;
    first = newnode;
    current_size++;
}
bool LinkedList::Delete(int &element){
    // 마지막 노드의 값을 리턴하면서, 메모리에서 할당 해제
    if(first == 0) return false;
    Node *current = first, *previous = 0;
    while(1){    // 마지막 노드까지 찾아가는 반복문
        if(current->link == 0)    // find end node
        {
            if(previous) previous->link = current->link;
            else first = first->link;
            break;
        }
        previous = current;
        current = current->link;
    }
    element = current->data;
    delete current;
    current_size--;
    return true;
}
```

LinkedList 클래스를 템플릿 클래스로 변경

앞 슬라이드에서 주어진 원본 클래스는 int형만을 지원함

앞의 클래스 구현에서 Print() 함수는 직접 구현하여야 함

아래의 코드를 실행하여, 다음 슬라이드와 같은 출력이 나와야 함

```
int main(){
    double dVal;
    string strVal;
    LinkedList<double> dList;
    LinkedList<string> strList;

    dList.Insert(3.14);
    dList.Insert(123456);
    dList.Insert(-0.987654);
    dList.Print();
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Print();
    dList.Insert(777.777);
    dList.Print();
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Print();
    dList.Delete(dVal);
    cout<<"삭제된 마지막 원소: "<<dVal<<endl;
    dList.Print();

    strList.Insert("This");
    strList.Insert("is a");
    strList.Insert("Template");
    strList.Insert("Example");
    strList.Print();
    strList.Delete(strVal);
    cout<<"삭제된 마지막 원소: "<<strVal<<endl;
    strList.Insert("Class");
    strList.Print();

    return 0;
}
```

LinkedList 클래스를 템플릿 클래스로 변경

출력 결과

```
[1|-0.987654]->[2|123456]->[3|3.14]
삭제된 마지막 원소: 3.14
[1|-0.987654]->[2|123456]
[1|777.777]->[2|-0.987654]->[3|123456]
삭제된 마지막 원소: 123456
삭제된 마지막 원소: -0.987654
[1|777.777]
삭제된 마지막 원소: 777.777
[1|Example]->[2|Template]->[3|is a]->[4|This]
삭제된 마지막 원소: This
[1|Class]->[2|Example]->[3|Template]->[4|is a]
```

Stack 클래스의 작성

앞에서 확장한 템플릿 기반의 LinkedList 클래스를 상속하여 Stack 클래스를 구현한다.

변경하여야 할 부분: Delete() 함수만
재정의(Overriding)하여, 앞의 클래스에서 맨 뒤의 데이터 원소를 삭제하는 것 대신 맨 앞의 데이터 원소를 삭제하도록 하면 됨

Stack 클래스가 잘 구현되었을 때, 다음 테스트 코드를 수행하여 C++에서 구현되는 파라미터적 다형성 및 서브타입 다형성을 확인할 수 있다.

Stack 클래스의 작성 – this

기본적으로 this는 class의 멤버변수에 접근하기 위한 것

c++에서 class를 작성할때 class 멤버 변수와 method의 파라미터의 변수의 이름이 같은 경우, method 구현시 변수를 사용하면 class 멤버 변수를 사용하는건지 파라미터의 변수를 사용하는 건지 의미가 모호해 질 수 있다.

이럴 경우 this 포인터를 사용하여 모호성을 없앤다.

```
class test_class{
public:
    int value;
    void set_value(int value){
    this->value = value;
    }
};
```

this 포인터를 사용하면 내부의 변수를 사용한다는 의미를 갖고

사용하지 않으면 파라미터의 변수를 사용한다는 의미가 된다.

위의 코드에서 this -> value는 위에서 선언된 class 멤버 변수 value를 가리키고 값으로 받는 value는 set_value의 파라미터인 value값을 받는다.

최종 테스트 코드

```
void prnMenu(){

cout<<"*****************************************"<<endl;
    cout<<"* 1. 삽입    2. 삭제    3. 출력    4. 종료 *"<<endl;

cout<<"*****************************************"<<endl;
    cout<<endl;
    cout<<"원하시는 메뉴를 골라주세요: ";
}

int main(){
    // 스택 및 연결 리스트 테스트용 코드
    int mode, selectNumber, tmpItem;
    LinkedList<int> *p;
    bool flag = false;

    cout<<"자료구조 선택(1: Stack, Other: Linked List): ";
    cin>>mode;

    // 기반 클래스의 포인터를 사용하여 기반 클래스 뿐만 아니라
    // 파생 클래스의 인스턴스 또한 접근할 수 있다.
    if(mode == 1)
        p = new Stack<int>();    // 정수를 저장하는 스택
    else
        p = new LinkedList<int>();    // 정수를 저장하는 연결 리스트
```

서브타입 다형성을 위해 기반 클래스의 포인터에
파생 클래스 인스턴스의 주소를 저장할 수 있게 한다.
mode가 1일 경우 이 서브타입다형성이 구현됨.

```
// 처리 부분
    do{
        prnMenu();
        cin>>selectNumber;
        switch(selectNumber){
        case 1:
            cout<<"원하시는 값을 입력해주세요: ";
            cin>>tmpItem;    p->Insert(tmpItem);
            cout<<tmpItem<<"가 삽입되었습니다."<<endl;
            break;
        case 2:
            if(p->Delete(tmpItem)==true)
                cout<<tmpItem<<"가 삭제되었습니다."<<endl;
            else cout<<"비어있습니다. 삭제 실패"<<endl;
            break;
        case 3:
            cout<<"크기: "<<p->GetSize()<<endl;
            p->Print();
            break;
        case 4:
            flag = true;    break;
        default:
            cout<<"잘못 입력하셨습니다."<<endl;
            break;
        }
        if(flag) break;
    } while(1);
    return 0;
}
```

최종 테스트 코드 수행 예: Stack

```
자료구조 선택(1: Stack, Other: Linked List): 1
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 33
33가 삽입되었습니다.
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 44
44가 삽입되었습니다.
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 55
55가 삽입되었습니다.
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************
```

```
원하시는 메뉴를 골라주세요: 3
크기: 3
[1|55]->[2|44]->[3|33]
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 2
55가 삭제되었습니다.
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 66
66가 삽입되었습니다.
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요: 3
크기: 3
[1|66]->[2|44]->[3|33]
*****************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*****************************************

원하시는 메뉴를 골라주세요:
```

최종 테스트 코드 수행 예: LinkedList

```
자료구조 선택(1: Stack, Other: Linked List): 2
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 11
11가 삽입되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 22
22가 삽입되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 3
크기: 2
[1|22]->[2|11]
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 33
33가 삽입되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 1
원하시는 값을 입력해주세요: 44
44가 삽입되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 2
11가 삭제되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 2
22가 삭제되었습니다.
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요: 3
크기: 2
[1|44]->[2|33]
*******************************************
* 1. 삽입   2. 삭제   3. 출력   4. 종료 *
*******************************************

원하시는 메뉴를 골라주세요:
```

예비 레포트

PDF 294page, 4-1 예비보고서에 있는 항목을 작성해서 제출한다.

제출은 사이버 캠퍼스 과제란을 활용한다.

실습 결과 레포트

PDF 294page, 4-3 결과 보고서에 있는 항목에 대해서 기술한다.

제출은 사이버 캠퍼스 과제란을 활용한다.