객체프로그래밍 정리노트

202204103 ict융합공학부 이승훈
202204023 ict융합공학부 김보민

[제네릭]
- 함수나 클래스를 일반화시키고, 매개 변수 타입을 지정하여 틀에서 찍어내듯이 함수나 클래스의 코드를 생산하는 기법

[템플릿]
- 함수나 클래스를 일반화하는 도구
- template 키워드로 함수나 클래스 선언
- 제네릭 타입 => 일반화를 위한 데이터 타입

[구체화]
- 템플릿의 제네릭 타입에 구체적인 타입 지정
(템플릿 함수로부터 구체화된 함수의 소스 코드 생성)

[예제 10-1]

```
//보민 , ->승훈

#include <iostream>
using namespace std;
class Circle {
	int radius;
public:
	Circle(int radius = 1) { this->radius = radius; }
	int getRadius() { return radius; }
};
template <class T> // myswap이라는 템플릿 함수를 정의하고 이 함수는 T타입의 두 변수를
                   참조 받아서 값을 교환해줘. 그리고 아래를 보면 tmp라는 T타입의 임시
                   변수를 사용하는 것을 알 수 있지.
void myswap(T& a, T& b) {
	T tmp;
	tmp = a;
	a = b;
	b = tmp;
}
```

```
int main() {
	int a = 4, b = 5;
	myswap(a, b); // myswap이라는 함수는 템플릿이기 때문에 정수, 실수, Circle 클래스
                      의 객체에 대해 값 교환을 수행하는 코드로 보면 될 것 같아. 근데
                      다른 종류의 제네릭 타입을 한 번에 구체화하진 못하나 ?
// 그게 우리가 구체화를 시킬 때 주의해야 할 점이야. 두 매개 변수 a,b의 제네릭 타입이 동일해야해. 두 개의 매개 변수 타입이 다를 경우에는 컴파일 오류가 나게 돼
	cout << "a=" << a << ", " << "b=" << b << endl;
	double c = 0.3, d = 12.5;
	myswap(c, d);
	cout << "c=" << c << ", " << "d=" << d << endl;
	Circle donut(5), pizza(20);
	myswap(donut, pizza);
	cout << "donut반지름=" << donut.getRadius() << ", ";
	cout << "pizza반지름=" << pizza.getRadius() << endl;
}
```

[템플릿 장점]

- 함수 코드의 제사용 (높은 소프트웨어의 생산성과 유용성)

[템플릿 단점]

- 포팅에 취약 (컴파일러에 따라 지원하지 않을 수 있음)
- 컴파일 오류 메시지 빈약, 디버깅 어려움

[제네릭 프로그래밍]

- 즉, 일반화 프로그래밍
- 제네릭 함수나 제네릭 클래스를 활용
- c++에서는 STL 제공/ 활용
- 보편화 추세 (JAVA, C# 등 활용)

[예제 10-2]
(메인함수 보고 제네릭 함수 만들기)

- 메인함수

```
int main() {
	int a = 20, b = 50;
	char c = 'a', d = 'z';
	cout << "bigger(20, 50)의 결과는 " << bigger(a, b) << endl;
	cout << "bigger('a', 'z')의 결과는 " << bigger(c, d) << endl;
}
```

- 제네릭 함수

```
template <class T>
T bigger(T a, T b) {
        if (a > b)
                return a;
        else
                return b;
}
```

[예제 10-3]

```
#include <iostream>
using namespace std;
template <class T>
T add(T data[], int n) { -> 배열 data에서 n개의 원소를 합한 결과를 리턴
        T sum = 0;
        for (int i = 0; i < n; i++) {
                sum += data[i];
        }
        return sum;
}
int main() {
        int x[] = { 1,2,3,4,5 };
        double d[] = { 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 };
        cout << "sum of x[] = " << add(x, 5) << endl;
        cout << "sum of d[] = " << add(d, 6) << endl;
}
```

[예제 10-6]

```
#include <iostream>
using namespace std;
template <class T>
class MyStack {
        int tos; // 코드를 보니 일반변수는 아닌 것 같고 이게 어떤 변수일까 ?
                -> 검색해보자. 아 이건 스택의 최상단을 나타내는 변수라고 나오네.
```

```
        T data[100];
public:
        MyStack();
        void push(T element); //element를 data[]에 삽입하는 코드로 볼 수 있고,
        T pop();              스택의 탑에 있는 데이터를 data[]에서 리턴해준다고 볼 수 있어.
};
template <class T>
MyStack<T>::MyStack() {
        tos = -1; //  tos를 초기화 해주어 스택이 비어있는 상태로 설정해 준 거야.
}
template <class T>
void MyStack<T>::push(T element) {
        if (tos == 99) {
                cout << "stack full";
                return;
        }
        tos++;
        data[tos] = element;
}
// 그 다음 push함수를 이용하여 스택에 요소를 추가할 수 있게 해줘
template <class T>
T MyStack<T>::pop() {
        T retData;
        if (tos == -1) {
                cout << "stack empty";
                return 0;
        }
        retData = data[tos--];
        return retData;
}
//pop함수는 스택의 최상단 데이터를 반환하고 스택에서 제거가 가능하도록 만들어줬어,
int main() {
        MyStack<int> iStack;
        iStack.push(3);
        cout << iStack.pop() << endl;
        MyStack<double> dStack;
        dStack.push(3.5);
        cout << dStack.pop() << endl;
        MyStack<char>* p = new MyStack<char>();
        p->push('a');
        cout << p->pop() << endl;
```

```
        delete p;
}
```
// main함수를 보면 각각의 스택을 생성하고 정수, 실수, 문자를 push/pop 하여 결과를 출력해주는 코드라고 볼 수 있어. 동적으로 할당된 MyStack<char> 객체에 대해서는 delete 연산자를 사용하여 메모리를 해제해주고 있는 것을 볼 수 있어.
-> 이제 코드를 스스로 이해할 수 있을 거 같아, 다음 코드는 내가 설명해볼게

[예제 10-8]

```
#include <iostream>
using namespace std;
template <class T1, class T2>
class GClass {
        T1 data1;
        T2 data2;
public:
        GClass();
        void set(T1 a, T2 b);
        void get(T1& a, T2& b);
};
```
-> GClass 클래스는 두 개의 템플릿 매개변수 T1, T2를 가지면서 data1과 data2 두 개의 데이터 멤버를 포함하고 있는 식을 나타내고 있는 것 같아.
```
template <class T1, class T2>
GClass<T1, T2>::GClass() {
        data1 = 0; data2 = 0;
}
```
-> 각 데이터 멤버를 0으로 초기화 하고 있으며
```
template <class T1, class T2>
void GClass<T1, T2>::set(T1 a, T2 b) {
        data1 = a; data2 = b;
}
```
-> set은 두 개의 매개변수를 받고 있는 것은 알겠는데 정확한 역할을 잘 모르겠어.
// 두 개의 매개변수를 받아서 클래스의 멤버 데이터 값을 설정해주고 있어. 이 코드를 작성한 이유는 이 코드를 통해서 객체의 상태를 외부에서 지정한 값으로 초기화가 가능해지게 되지.
-> 그렇구나.
```
template <class T1, class T2>
void GClass<T1, T2>::get(T1& a, T2& b) {
        a = data1; b = data2;
}
```
-> 클래스의 멤버 데이터 값을 변수에 반환시키도록 하는 코드 맞지 ?

```
// 맞아. 정확히 말하면 매개변수로 전달된 변수에 반환해줘.
int main() {
        int a;
        double b;
        GClass<int, double> x;
        x.set(2, 0.5);
        x.get(a, b);
        cout << "a=" << a << '\t' << "b=" << b << endl;
        char c;
        float d;
        GClass<char, float> y;
        y.set('m', 12.5);
        y.get(c, d);
        cout << "c=" << c << '\t' << "d=" << d << endl;
}
```

-> 메인에서는 GClass를 타입에 따라 인스턴스화 하여 사용하는 것을 볼 수 있어.
두 객체에 대해 값을 설정하면서 get 메서드를 사용하여 값을 가져와 출력하기에 다음과 같은 결과값이 나오고 있음을 알 수 있지.
// 맞게 잘 설명한 거 같아.

[vector 컨테이너]

- 가변 길이 배열을 구현한 제네릭 클래스
(개발자가 벡터의 길이를 고려할 필요 없음)
- 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원
- 백터에 저장된 원소는 인덱스로 접근 가능
(인덱스는 0부터 시작)

[예제 10-9]

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
        vector<int> v; // 정수를 저장하는 vector 객체 v를 선언해주면 이 객체는
                       비어있게 될 거야.
        v.push_back(1); // 그 다음 push_back라는 함수를 사용하여서 vector에 값을
                        추가해줬어
        v.push_back(2);
        v.push_back(3);
        for (int i = 0; i < v.size(); i++)
```

```
            cout << v[i] << " "; // 이를 통해서 첫 번째 줄에 1,2,3이 출력된 거 같아
        cout << endl;
        v[0] = 10;  // 여기서는 vector의 첫 번째 요소를 10으로 바꿔줘서 지금 vector의
                    값은 10,2,3이 되어 있을 거야.
        int n = v[2]; // 세 번째 요소 값을 변수 n에 복사하였기 때문에 지금 n은
                    3이겠지?
        v.at(2) = 5;  // 다음으로 at 함수를 사용하여 vector의 두 번째 요소를 5로 바꿔줘
                    그럼 지금 10,2,5가 된 걸 알 수 있어.
        for (int i = 0; i < v.size(); i++)
            cout << v[i] << " "; // 바뀐vector을 순서대로 출력하기 때문에
                                    출력값은 10,2,5가 되는 것을 볼 수 있지.
                    -> 처음 코드를 봤을 땐 어떻게 해석할지 막막했는데 막상
                    한줄한줄 해석하다보니 할만한 것 같아. 다음 건 내가 해볼게

        cout << endl;
}
```

[iterator 사용]

- 반복자라고 부름
- 컨데이너의 원소를 가리키는 포인터
- iterator의 변수 선언 => 구체적인 컨테이너를 지정하여 반복자 변수 생성

[예제 10-11]

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
        vector<int> v; -> vector 객체를 선언해주고
        v.push_back(1);
        v.push_back(2);
        v.push_back(3); -> vector에 값을 추가해줘
        vector<int>::iterator it;  -> vector의 원소에 대한 iterator it을 선언해줬어.
        for (it = v.begin(); it != v.end(); it++) { -> 벡터의 시작점과 끝 지점에 함수를
                                                    각각 선언해주었고, iterator을 사용하여
                                                    벡터를 순회시켜줬어.
            int n = *it;  -> 이 부분의 의미가 정확히 뭘까 ?
                        //이 부분은 iterator가 현재 가리키는 값을 n에 복사해준
                        것을 의미해
            n = n * 2; ->이 부분에서 n을 두 배로 업데이트 해주고
            *it = n; -> 업데이트 된 값을 n에 저장했다고 볼 수 있지.
```

```
        }
        for (it = v.begin(); it != v.end(); it++) ->여기서는 업데이트된 vector 값을 출력해
                                                    주기 때문에 출력값은 2,4,6dl 됨을 알 수 있어.
                cout << *it << ' ';
        cout << endl;
}
```

-> vector을 사용하면 동적으로 크기가 조절되는 배열을 만들어줄 수 있으며, iterator을 사용하면 모든 원소를 탐색함과 동시에 값을 업데이트 시켜줄 수 있음을 볼 수 있었어. 코드를 읽는 거까지는 이제 할 수 있겠어. 과제를 하면서 직접 코드를 작성해보면서 부족한 부분을 채워나가야겠어.

//좋아, 과제하면서 더 연습해보자.