객체지향프로그래밍 정리노트(Week7)

202004029 김정호, 202004040 노성민

Q1. 10-1 제네릭 myswap() 함수 만들기

```
#include <iostream>
using namespace std;
class Circle {
		int radius;
public:
		Circle(int radius=1) { this->radius = radius; }
		int getRadius() { return radius; }
};
template <class T>
void myswap(T & a, T & b) {
		T tmp;
		tmp = a;
		a = b;
		b = tmp;
}
int main() {
		int a=4, b=5;
		myswap(a, b);
		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;
}
```

김정호: 10-1 예제의 main 함수를 보면 myswap의 인자가 각각 타입이 다르고 함수명은 같은데 이런 함수 중복을 해결하기 위해서 사용하는 것이 템플릿이야. 먼저 template 키워드를 사용하고 제네릭 타입을 선언하는 키워드 class와 제네릭 타입 T를 선언하여 함수를 일반화했어. 따라서 코드가 실행될 때 각 코드에 맞게 T가 int&, double&, Circle& 이렇게 적용되면서 출력 값이
a=5, b=4
c=12.5, d=0.3
donut반지름=20, pizza반지름=5 이렇게 나오게 되는 거야.

노성민: 매개변수의 타입을 각각 다르게 해줘서 하는 것은 불가능할까? 예를 들면, int ,double 같이 인자로 넣는 거지.

김정호: ppt 내용을 살펴보면 두 개의 매개변수의 타입이 서로 다르면 안된다고 나오는 것 같아.

Q2. 10-2 큰 값을 리턴하는 bigger() 함수

두 값을 매개 변수로 받아 큰 값을 리턴하는 제네릭 함수 bigger()를 작성하라.

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

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

노성민: 우선 template <class T>를 선언하여 템플릿을 만들고 반환 타입이 int, char 등 다른 타입이 설정될 수 있도록 하기 위해서는 반환 타입을 T로 설정해야해. 그리고 bigger(T a,T b)로 함수를 만들어 if문을 이용해 큰 값을 리턴하게 하면 끝이야.

Q3. 10-3 배열의 합을 구하여 리턴하는 제네릭 add() 함수 만들기 연습

배열과 크기를 매개 변수로 받아 합을 구하여 리턴하는 제네릭 함수 add()를 작성하라.

```
#include <iostream>
using namespace std;
template <class T>
T add(T data [], int n) {
	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; // 배열 x와 원소 5개의 합을 계산
	cout << "sum of d[] = " << add(d, 6) << endl; // 배열 d와 원소 6개의 합을 계산
}
```

김정호: 위 문제에서도 10-2번 예제와 거의 일치하는 것 같아. 위처럼 Template을 선언하고 여기서도 보면 타입이 int, double로 하나로 정해져 있지 않고 다르기 때문에 T로 반환하고 add() 안에 매개변수는 main 함수에서 보면 배열을 인자로 받고 int 형 변수 하나를 받았기 때문에 add(T data[], int n)이 되는거야. 그리고 안에 코드는 기초적인 합계를 구하는 코드를 작성하면 끝이야. 여기서 주의할 점은 sum의 타입을 T로 하는 것에 유의하면 돼.

Q4. 10-6 제네릭 스택 클래스 만들기

```
#include <iostream>
using namespace std;
template <class T>
class MyStack {
	int tos;
	T data [100];
public:
	MyStack();
	void push(T element);
	T pop();
};
template <class T>
MyStack<T>::MyStack() {
	tos = -1;
}
template <class T>
void MyStack<T>::push(T element) {
	if(tos == 99) {
		cout << "stack full";
		return;
	}
	tos++;
	data[tos] = element;
}
template <class T>
T MyStack<T>::pop() {
	T retData;
	if(tos == -1) {
		cout << "stack empty";
		return 0; // 오류 표시
	}
	retData = data[tos--];
	return retData;
}
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;
}
```

김정호: 내가 생각했을 때는 제네릭 타입의 클래스를 만드는 거나 위에 예제들이나 개념은 큰 차이가 없다고 생각해. 추가적으로 볼 것이 있다면 제네릭 타입의 클래스는 함수나 생성자를 선언할 때 template<class T>를 코드에 적어줘야 하다는 점 같고 이런 제네릭 타입을 이용하여 함수 중복에 대한 문제를 해결할 수 있다고 생각해.

Q5. 예제 10-8 두 개의 제네릭 타입을 가진 클래스 만들기

```
#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);
};
template <class T1, class T2>
GClass<T1, T2>::GClass() {
	data1 = 0; data2 = 0;
}
template <class T1, class T2>
void GClass<T1, T2>::set(T1 a, T2 b) {
	data1 = a; data2 = b;
}
template <class T1, class T2>
void GClass<T1, T2>::get(T1 & a, T2 & b) {
	a = data1; b = data2;
}
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;
}
```

노성민: 두 개의 제네릭 타입을 가지는 클래스를 만드는 예제인데, 앞선 예제들과 개념이 크게 다르진 않아. 위에서 말했었던 타입이 다른 매개변수의 문제를 해결해줄 수 있는 것이 특징이고 앞선 코드들과 다른 점은 제네릭 선언이 된 것이 두 개라는 점이야.

Q6. 10-9 vector 컨테이너 활용하기

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
	vector<int> v;
	v.push_back(1);
	v.push_back(2);
	v.push_back(3);
	for(int i=0; i<v.size(); i++)
		cout << v[i] << " ";
	cout << endl;
	v[0] = 10;
	int n = v[2];
	v.at(2) = 5;
	for(int i=0; i<v.size(); i++)
		cout << v[i] << " ";
	cout << endl;
}
```

김정호: 여기서는 vector 컨테이너 클래스를 사용하기 위한 헤더 파일 #include <vector>가 추가되어 vector 컨테이너를 활용한 예제야.

노성민: vector 컨테이너를 이용해서 무엇을 할 수 있지?

김정호: vector 컨테이너의 특징은 가변 길이 배열을 구현한 제네릭 클래스로 벡터의 길이에 대해 고민할 필요가 없으며 다양한 멤버 함수를 통해 원소의 저장, 삭제, 검색 등이 가능해져. 우선 vector<int> v;를 선언하여 정수 타입의 벡터를 선언했어. 그리고 push_back() 함수를 이용하여 정수를 삽입했어.

노성민: 그리고 v.size()함수를 이용하여 v의 크기만큼 반복하여 v를 출력하고 있어. 그리고 v의 값을 조정해주는 코드도 보이고 v.at(2)를 이용해 index 위치의 원소에 대한 참조를 리턴하여 값을 수정하고 다시 출력해주는 예제라고 볼 수 있어.

Q7. 10-11 iterator를 사용하여 vector의 모든 원소에 2 곱하기

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
	vector<int> v;
	v.push_back(1);
	v.push_back(2);
	v.push_back(3);
	vector<int>::iterator it;
	for(it=v.begin(); it != v.end(); it++) {
		int n = *it;
		n = n*2;
		*it = n;
	}
	for(it=v.begin(); it != v.end(); it++)
		cout << *it << ' ';
	cout << endl;
}
```

노성민: v의 원소에 대한 포인터를 선언하여 iterator를 이용하여 v에 대한 모든 원소를 탐색하여 2를 곱하여 출력하는 예제야. 이렇게 STL을 이용해서 기존이랑 다른 방식으로 코딩을 할 수 있어.

김정호: 그러면 다른 컨테이너도 사용해보자.

#Q1. 예제 10-12 map으로 영한사전 만들기

map 컨테이너를 이용하여 (영어, 한글) 단어를 쌍으로 저장하고, 영어로 한글을 검색하는 사전을 작성하라.

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
	map<string, string> dic;
	dic.insert(make_pair("love", "사랑"));
	dic.insert(make_pair("apple", "사과"));
	dic["cherry"] = "체리";
	cout << "저장된 단어 개수 " << dic.size() << endl;
string eng;
while (true) {
	cout << "찾고 싶은 단어>> ";
	getline(cin, eng);
	if (eng == "exit")
		break;
	if(dic.find(eng) == dic.end())
```

```
            cout << "없음" << endl;
        else
            cout << dic[eng] << endl;
        }
        cout << "종료합니다..." << endl;
}
```

김정호: 이번에는 map이라는 컨테이너에 대해 알아보자. (‘키’, ‘값’)의 쌍을 원소로 저장하는 제네릭 컨테이너로서 ‘키’를 통해 ‘값’을 검색해서 필요한 정보를 찾을 수 있어. 하지만 동일한 ‘키’를 가진 원소가 중복되어 저장되면 오류가 발생하니 주의해야 해.

노성민: 이 예제에서는 dic라는 맵 컨테이너를 생성해서 insert()함수를 이용하여 3개의 정보를 저장해서 그 찾고 싶은 단어를 찾는 코드를 작성한 것같아. find 함수를 통해 ‘키’를 찾고 end 함수를 통해 맵의 끝을 가리키는 참조를 리턴하여 끝까지 검색하였을 때도 없는 경우를 나타냈어.