정리노트 #7

13조(전대원, 소재현)

템플릿 : 함수나 클래스를 일반화하는 C++ 도구

template 키워드로 함수나 클래스 선언

변수나 매개 변수의 타입만 다르고, 코드 부분이 동일한 함수를 일반화시킴

제네릭 타입 - 일반화를 위한 데이터 타입

템플릿 선언

Ex)
```
template <class T>
void myswap (T & a, T & b) {
T tmp;
tmp = a;
a = b;
b = tmp;
}
```

class T는 제네릭 타입을 선언한다

Template 템플릿을 선언한다

예제 10-1 정리

Myswap(a,b); // 에서 myswap(int& a, int& b)

함수 구체화 및 호출

myswap(c, d); // myswap(double& a, double& b)

함수 구체화 및 호출

myswap(donut, pizza); // myswap(Circle& a, Circle& b)

함수 구체화 및 호출

템플릿의 장단점과 제네릭 프로그래밍 정리

템플릿 장점

ㅇ 함수 코드의 재사용

ㅇ 높은 소프트웨어의 생산성과 유용성

템플릿 단점

ㅇ 포팅에 취약

ㅇ 컴파일러에 따라 지원하지 않을 수 있음

ㅇ 컴파일 오류 메시지 빈약, 디버깅에 많은 어려움

제네릭 프로그래밍

ㅇ 일반화 프로그래밍이라고도 부름

ㅇ 제네릭 함수나 제네릭 클래스를 활용하는 프로그래밍 기법

ㅇ C++에서 STL(Standard Template Library) 제공. 활용

예제 10-2

bigger() 함수의 등장

T bigger(T a, T b)

두 개의 매개 변수를 비교하여 큰 값을 리턴한다

STL 라이브러리

Standard Template Library

ㅇ 표준 템플릿 라이브러리

ㅇ C++ 표준 라이브러리 중 하나

ㅇ 많은 제네릭 클래스와 제네릭 함수 포함

ㅇ 개발자는 이들을 이용하여 쉽게 응용 프로그램 작성

STL의 구성

ㅇ 컨테이너 – 템플릿 클래스

ㅇ 데이터를 담아두는 자료 구조를 표현한 클래스

ㅇ 리스트, 큐, 스택, 맵, 셋, 벡터

ㅇ iterator – 컨테이너 원소에 대한 포인터

ㅇ 컨테이너의 원소들을 순회하면서 접근하기 위해 만들어진 컨테이너 원소에 대한 포인터

알고리즘 – 템플릿 함수

ㅇ 컨테이너 원소에 대한 복사, 검색, 삭제, 정렬 등의 기능을 구현한 템플릿

함수

ㅇ 컨테이너의 멤버 함수 아님

- Vector : 가변 크기의 배열을 일반화한 클래스
- 헤더파일 <vector>

vector 클래스를 사용하려면 #include <vector>

vector 컨테이너

특징 :

ㅇ 가변 길이 배열을 구현한 제네릭 클래스

ㅇ 개발자가 벡터의 길이에 대한 고민할 필요 없음

ㅇ 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원

ㅇ 벡터에 저장된 원소는 인덱스로 접근 가능

ㅇ 인덱스는 0부터 시작

iterator 사용

ㅇ 반복자라고도 부름

ㅇ 컨테이너의 원소를 가리키는 포인터

ㅇ iterator 변수 선언 : 구체적인 컨테이너를 지정하여 반복자 변수 생성

vector<int>::iterator it;

it = v.begin();

it는 원소가 int타입인 벡터의 원소에 대한 포인터

C++ auto

변수선언문에서 타입을 자동 선언하도록 지시한다

ㅇ 복잡한 변수 선언을 간소하게, 긴 타입 선언 시 오타 줄임

auto ref2 = ref; 로 자동 선언