(선택) 실습 084 - variadic template

제출 마감일: 2023-05-14 23:59
업로드 가능한 파일 수: 2
제출 방식: 개인

목적

이 실습은 variadic template 표현을 연습을 합니다.

설명

가변 템플릿을 이용하면

여러 개 (미리 정해지지 않음) 의 템플릿 타입 인자를 받아서,

개별 타입별로 특별한 처리를 할 수 있습니다.

유사한 문법을 떠올려 본다면

C 언어에서 지원하는 ... (three dotes) 이 있습니다.

이를 이용해서 함수의 가변 인자 (variable length argument) 를 정의하고,

가변 길이의 인자를 처리할 수 있었습니다.

대표적으로 printf () 함수가 가변 길이 인자를 이용해 구현되어 있습니다.

variadic template 도 문법적으로 유사합니다.

... 을 이용해서

가변 길이의 타입들을 정의할 수 있습니다.

예제 코드를 살펴 보겠습니다.

```
template<typename T, typename... Args>
void processValues(T arg0, Args... args){
   processValues(args...);
}
int main() {
   std::vector<int> vec = {100, 200, 300};
   processValues(1, "a", vec);
   return 0;
```

}

processValues() 함수를 variadic template 으로 정의하고 있습니다.

1) template<typename T, typename... Args> 의 typename... Args: 여기 부분에서 가변 길이의 타입을 받는다는 표시를 ... 를 이용해서 했습니다.

2) void processValues(T arg0, Args... args) : 이 함수의 두번째 인자로 (variadic template) Args 타입의 args 인자를 받습니다.

3) main() 함수에서 processValues (1, "a", vec) ; 를 호출합니다. 드디어 컴파일러가 타입 추론을 통해 템플릿 함수를 인스터스화 시킬 수 있게 되었습니다.

타입은 int, const char*, vector 3가지 종류입니다.

그럼 https://cppinsights.io/ 사이트를 이용하여

processValues 템플릿 함수가 어떻게 인스턴스화 되는지 살펴보겠습니다.

processValues<int, const char *, std::vector<int>>(int arg0, const char * **args1, std::vector<int** > args1)

processValues<const char *, std::vector<int>>(const char * arg0, std::vector<int > __args1)

processValues < std::vector<int>> (std::vector<int > __args1)

인스턴스화 과정에서

컴파일러는 processVaules() 재귀함수를 호출하는 부분에서 args... 연산자를 이용해

가변 템플릿 인자 타입을 추론하여

3가지 종류의 processValue (인자 3개), processValue(인자 2개), processValue(인자 1개) 를

인스턴스화 하면서 컴파일 에러를 발생 시킵니다.

재귀 함수를 정의할 때 우리는 항상 종료 조건에 신경을 써야 합니다.

재귀 호출의 마지막 단계, 즉 인자가 하나도 없는 함수 호출이 필요하기 때문에,

다음과 같은 processValues() 함수를 정의해 주면 정상적으로 컴파일 되는 것을 확인할 수 있습니다.

processValues() { }

이렇게 4번째 형태의 함수를 정의해 주는 방법도 있으나,

3번째 단계의 processValues < std::vector<int>> (std::vector<int > __args1) 템플릿 함수를

미리 특수화 (specialization) 해서 개발자가 다음과 같이 정의하면

4번째 함수를 정의하지 않고도 정상적으로 컴파일 가능합니다.

(아래 함수에서는 자기 자신을 호출하고 있지 않음)

```
void processValues(std::vector<int> vec) {
    std::cout << "vec: " ;
    for(auto& v : vec)
        std::cout << v << " ";
}
```

다음 코드를 실행하여 로그를 확인하면

variadic template의 인스턴스화 과정을 이해하는데 도움이 됩니다.

main.cpp ----------

```
#include <iostream>
#include <vector>

void processValues(int i) {
    std::cout << "int: " << i << std::endl;
}

void processValues(const char* s) {
    std::cout << "char*: " << s << std::endl;
}

void processValues(std::vector<int> vec) {
    std::cout << "vec: " ;
    for(auto& v : vec)
        std::cout << v << " ";
}

template<typename T, typename... Args>
void processValues(T arg0, Args... args){
    std::cout << arg0 << " "<< sizeof...(args) << '\n';
    processValues(arg0);
    processValues(args...);
}

int main() {
    std::vector<int> vec = {100, 200, 300};
    processValues(1, "a", vec);
    return 0;
}
```

문제

아래와 같이 합계를 계산하는 템플릿 함수 sum () 을 구현하시오.

단, variadic template 를 사용하시오.

```
int main() {
    std::cout<<sum(1, 2, 3, 4) <<std::endl;
}
```

일련

입력

없음

출력

10

제출파일

84.csv

main.cpp

참고

main.cpp ----------

```
#include <iostream>
#include <vector>
int main() {
    std::cout<<sum(1, 2, 3, 4) <<std::endl;
    return 0;
}
```