C++ Programming Basic 실습 문제 1번 답과 해설 입니다.

1. 참조를 사용해서 Swap을 만들어 보세요

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

void Swap(int& x, int& y)
{
    int temp = x;
    x = y;
    y = temp;
}
int main()
{
    int a = 10, b = 20;
    Swap(a, b);
    cout << a << ", " << b << endl;
}
```

이 문제에 대해서 아래의 질문이 있었습니다.

질문)

```
void Swap(int& x, int& y)
{
    int temp = x;   // temp 는 int 이고 x는 int& 이므로 서로 다른 타입인데..
                    // 어떻게 대입이 가능한가 ?
    x = y;
    y = temp;
}
```

해설)

```
double d = 3.4;
int n = d;
```

위 코드에서 d는 분명 double 타입이지만 int 변수에 대입될 수 있습니다. 그 이유는

"double형 변수(값)는 int 형 타입으로 암시적 형변환이 가능하다."

라는 C 표준 문법이 있기 때문에 가능합니다. 즉, 서로 다른 타입의 변수에 있는 값을 담기 위해서는 해당 타입 끼리의 암시적 형변환이 지원 되어야 합니다.

```
int& r1 = n;
int& r2 = r1; // r1과 r2는 같은 타입입니다. r2도 결국 n을 가리키게 됩니다.

int n2 = r1; // r1은 int& 이지만 int로의 암시적 형변환을 지원합니다.
             // 이경우 r1이 가리키는 메모리의 값이 n2에 들어가게 됩니다.
```

참고)

포인터와 참조는 모두 다른 메모리를 가리키는 역할을 하게 됩니다. 그런데, 다음과 같은 차이점이 있습니다.

```
int n = 10;

// p와 r은 결국 둘다 n을 가리키게 됩니다.
int* p = &n;
int& r = n;

// 하지만 p, r을 사용해서 n의 값을 꺼내는 방법이 다릅니다.
int a = *p; // 포인터는 값을 꺼내기 위해 * 연산자를 사용하므로 명확합니다.
int b = r;  // 참조 변수는 그냥 참조 변수이름만 사용하면 꺼낼 수 있습니다.
            // 암시적 형변환이 발생하고 있습니다.
```

그래서, 일부 C++고급 서적에는 참조 변수를 아래 처럼 표현하기도 합니다.

"참조는 자동으로 Dereference(* 연산) 되는 포인터 이다"

2번. Swap을 템플릿으로 만들어 보세요.

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

template<typename T> void Swap(T& x, T& y)
{
    T temp = x;
    x = y;
    y = temp;
}
int main()
{
    int a = 10, b = 20;
    Swap(a, b);
    cout << a << ", " << b << endl;
}
```

이 문제에 대해서는 다음과 같은 질문이 있었습니다.

질문)

Swap을 선언과 구현으로 분리 해서 별도의 파일로 제공하니, 에러가 발생합니다.

해설)

일반적으로 C언어에서는 함수를 만들 때 다른 파일에 제공하고 선언부를 헤더로 제공하는 것이 관례 입니다. 아래 처럼 템플릿이 아닌 Swap을 다른 파일에 구현하는 것은 아무 문제 없습니다.

```
// Swap.h
void Swap(int& a, int& b);

// Swap.cpp
void Swap(int& x, int& y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
// main.cpp
#include "Swap.h"

int main()
{
    int a = 10, b = 20;
    Swap(a, b);
}
```

이때 선언을 include 하는 것은 결국 Swap.h 파일을 main.cpp 의 위쪽에 복사하는 것과 동일합니다. 즉, main.cpp는 아래 처럼 변경 된 후 컴파일 됩니다.

```
// main.cpp
void Swap(int& a, int& b); // Swap.h 안의 내용이 이 부분에 놓이게 됩니다.

int main()
{
    int a = 10, b = 20;
    Swap(a, b);
}
```

그럼, 이제 템플릿의 경우를 생각해 봅시다. 만일 템플릿도 선언과 구현으로 분리했다면 main.cpp 는 아래 처럼 변경될 것입니다.

```
// template 은 진짜 함수가 아니고 틀입니다. 사용자가 Swap을 사용 할 때 어떤 타입으로
// 사용하는 지를 보고 틀(template)으로 부터 함수는 만들어 내야 합니다.
template<typename T> void Swap(T& a, T& b);

int main()
{
    int a = 10, b = 20;

    // 아래 코드를 컴파일 할 때 컴파일러는 Swap 템플릿으로 부터 T를 int로 변경한 함수를
    // 생성해야 합니다.
    // 그런데, 이 파일에는 Swap 템플릿의 완전한 구현이 없고, 선언만 있습니다.
    // 그래서, 컴파일러는 Swap 의 모양을 알 수 없기 때문에 함수를 생성할 수 없습니다.
    Swap(a, b);
}
```

핵심은

1. 템플릿은 진짜 함수가 아니고 함수를 만드는 틀이므로, 실제 Swap 함수는 Swap함수를 사용하는 코드는 컴파일 할 때, 컴파일러가 만들게 된다는 점.
2. 컴파일러는 프로젝트내의 모든 파일을 동시에 컴파일하지 않고 하나씩 따로 따로 컴파일 한다는 점입니다.
3. 그래서 Swap()을 사용하는 소스 코드 안에는 반드시 Swap 함수 템플릿의 완전한 구현부

가 있어야 합니다.

결국 해결책은, 어떤 함수를 템플릿으로 만들 때는 반드시 헤더 파일에 완전한 함수 템플릿의 구현을 제공해야 합니다.

```
// Swap.h
template<typename T> void Swap(T& x, T& y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

```
// main.cpp
#include "Swap.h"

int main()
{
    int a = 10, b = 20;
    Swap(a, b);
}
```

문제 3. Swap를 Util namespace 안에 넣으세요..

이 경우는 템플릿인 경우는 헤더에 완전한 구현을 모두 넣어야 하고, 템플릿이 아닌 경우는 선언과 구현을 분리해도 상관없습니다.

```
// Swap.h
namespace Util
{
    template<typename T>
    void Swap(T& x, T& y)
    {
        T temp = x;
        x = y;
        y = temp;
    }
}
```

```
// main.cpp
#include <iostream>
#include "Swap.h"
using namespace std;
using namespace Util;

int main()
{

    int a = 10, b = 20;
    Swap(a, b);
    cout << a << ", " << b << endl;
}
```

문제 4. C++ 표준의 swap을 사용해 보세요.

핵심) C++ 표준의 많은 함수는 <algorithm> 헤더에 있습니다. 또한, 대부분 템플릿으로 되어 있고, std namespace 안에 있습니다.

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

int main()
{
    int a = 10, b = 20;
    swap(a, b);
    cout << a << ", " << b << endl;
}
```