Template

일반화

- C++은 일반화 프로그래밍(Generic Programming)이 가능한 언어
  - Template을 이용해서 프로그래밍

Template

1. 개념

- 매개변수의 타입에 상관없이 함수 및 클래스를 작성하는 기법
  - 그 타입 자체가 매개변수에 의해서 다루어 진다.
  - 타입마다 별도의 함수나 클래스 불필요.
- 소스코드의 재 사용성을 극대화할 수 있는 객체 지향 프로그래밍 기법

2. Code

```
#include <iostream>
#include <string>

using namespce std;

template <typename T>
void Change(T& tFir, T& tSec)
{
  T tTemp;
  tTemp = tFir;
  tFir = tSec;
  tSec = tTemp;
}

int main(void)
```

```
{
  string a = "신대수";
  string b = "홍길동";
  Change(a, b);
  cout << a << ":" << b << endl;
  system("pause");

  return 0;
}
```

3. 동작

- 함수 템플릿이 각각의 자료형에 대해서 처음 호출 될때,
- C++ 컴파일러는 해당 타입의 인스턴스를 생성
- 생성된 하나의 인스턴스는 해당 자료형에 대해서 특수화가 이루어 짐
  - 인스턴스는 해당 함수 템플릿에 해당 자료형이 사용될 때마다 호출

Explicit Specialization

> *명시적 특수화*

1. 개념

- 특정한 타입에 대해서 명시적 특수화 기능 제공
  - 특정한 타입에 대해서 특수한 기능을 정의 할 수 있다.
- 컴파일러는 호출된 함수에 대응하는 특수화된 정의를 발견한 이후에는 해당 정의만을 사용한다.
  - 효율적인 메모리 운영

2. Code

```
#include <iostream>
#include <string>

using namespce std;
```

```
// Generic Template function
template <typename T>
void Change(T& tFir, T& tSec)
{
  T tTemp;
  tTemp = tFir;
  tFir = tSec;
  tSec = tTemp;
}

// Int Type Explicit specialization Template function
template <>
void Change<int>(int &nFir, int &nSec)
{
  int nTemp = nFir;
  nFir = nSec;
  nSec = nTemp;
}

int main(void)
{
  string a = "신대수";
  string b = "홍길동";
  Change(a, b); // 7번 함수 호출
  cout << a << ":" << b << endl;

  int nA = 1;
  int nB = 2;
  Change(nA, nB); // 17번 함수 호출
```

```
  cout << nA << ":" << nB << endl;

  system("pause");

  return 0;
}
```

Class Template

> *자료형에 따라서 다르게 동작하는 클래스 집합을 만들 수 있다.*

1. Code

```
#include <iosteam>
#include <string>

using namespce std;

template <typename T>
class Data
{
private:
  T tData;

public:
  Data(T tData) : tData(tData) {}
  void SetData(T tData) { this->tData = tData; }
  T GetData() { return tData; }
};

int main(void)
```

```
{
  Data<int> data1(1);
  Data<string> data2("신대수");

  cout << data1.GetData() << ":" << data2.GetData() << endl;

  system("pause");
  return 0;
}
```

2. Default Template parameter

```
#include <iosteam>
#include <string>

using namespce std;

template <typename T = int> //default Template parameter
class Data
{
private:
  T tData;

public:
  Data(T tData) : tData(tData) {}
  void SetData(T tData) { this->tData = tData; }
  T GetData() { return tData; }
};

int main(void)
{
```

```
  Data<> data1(1); // default template parameter
  Data<string> data2("신대수");

  cout << data1.GetData() << ":" << data2.GetData() << endl;

  system("pause");
  return 0;
}
```