

Week 9 : C++ Templates

Part1. Template

함수 템플릿

함수를 만드는 도구

다양한 자료형의 함수를 만들 수 있음

함수 템플릿 정의 : `template <typename T>` 또는 `template <class T>`

```
Template <typename type명>
return-type 함수명(...)
{
    ...
}
```

- Function template

```
int Add(int a, int b)
{
    return a+b;
}
```

```
template <typename T>
T Add(T a, T b)
{
    return a+b;
}
```

T라는 type name에 대해서, 다음에 정의하는 대상을 템플릿으로 선언한다

T는 자료형을 결정짓지 않는다. -> 함수의 템플릿화

- Template function

```
int Add(int num1, int num2)
{
    return num1+num2;
}

double Add(double num1, double num2)
{
```

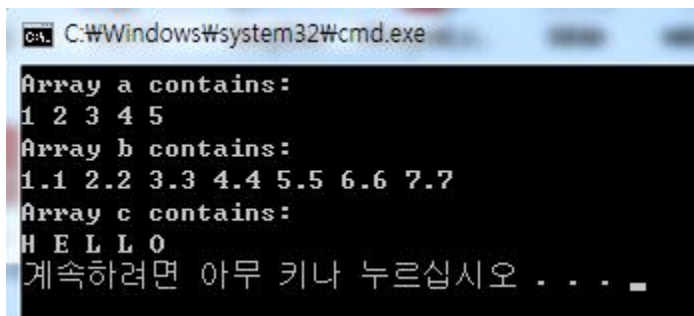
```
        return num1+num2;  
    }
```

```
#include <iostream>  
using std::cout;  
using std::endl;  
  
// function template printArray definition  
template< typename T >  
void printArray( const T *array, int count )  
{  
    for ( int i = 0; i < count; i++ )  
        cout << array[ i ] << " ";  
  
    cout << endl;  
} // end function template printArray  
  
int main()  
{  
    const int ACOUNT = 5; // size of array a  
    const int BCOUNT = 7; // size of array b  
    const int CCOUNT = 6; // size of array c  
    int a[ ACOUNT ] = { 1, 2, 3, 4, 5 };  
    double b[ BCOUNT ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };  
    char c[ CCOUNT ] = "HELLO"; // 6th position for null  
    // call integer function-template specialization  
    cout << "Array a contains:" << endl;  
    printArray( a, ACOUNT );  
  
    cout << "Array b contains:" << endl;  
    // call double function-template specialization
```

```
printArray( b, BCOUNT );

cout << "Array c contains:" << endl;
// call character function-template specialization
printArray( c, CCOUNT );

return 0;
} // end main
```



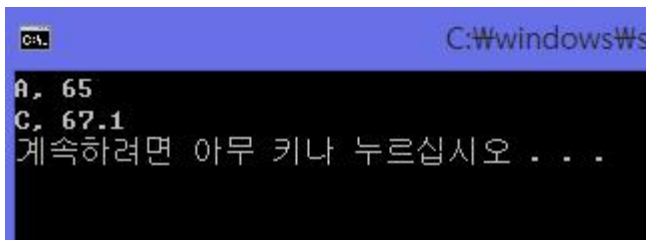
```
C:\Windows\system32\cmd.exe
Array a contains:
1 2 3 4 5
Array b contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
Array c contains:
H E L L O
계속하려면 아무 키나 누르십시오 . . .
```

- 둘 이상의 형태

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

template <class T1, class T2>
void ShowData(double num)
{
    cout<<(T1)num<<" , "<<(T2)num<<endl;
}
int main(void)
{
    ShowData<char, int>(65);
    ShowData<char, double>(67.1);
    return 0;
}
```

아스키코드에 해당하는 char 값과 숫자들이 함께 출력된다.



```
C:\windows\ws
A, 65
C, 67.1
계속하려면 아무 키나 누르십시오 . . .
```

- Class Template

1. Class Template 기반으로 객체 생성할 때, 결정하고자 하는 자료형을 명시적으로 선언해주어야 한다
2. Function Template 처럼 전달인자를 통해 자료형을 결정짓지 못한다
3. Constructor를 통해서 전달되는 인자의 자료형과 결정되어야 할 템플릿의 자료형이 다를수도 있기 때문에, 자료형을 명시적으로 선언해주어야 한다
하나의 파일 내에 선언과 정의가 함께 있어야 한다
 - Compiler: 템플릿 처리
 - 분리된 것(.cpp, .h)의 relationship은 Linker담당

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

template <typename T>

class Data
{
    T data;
public:
    Data(T d);
    void SetData(T d);
    T GetData();
};

template <typename T> // Define Class Template of T type
Data<T>::Data(T d){
```

```
        data = d;
    }

    template <typename T> // Define Class Template of T type
    void Data<T>::SetData(T d){
        data = d;
    }

    template <typename T> // Define Class Template of T type
    T Data<T>::GetData(){
        return data;
    }

    int main(void)
    {
        Data<int> d1(0);
        d1.SetData(10);
        Data<char> d2('a');

        cout<<d1.GetData()<<endl;
        cout<<d2.GetData()<<endl;

        return 0;
    }
}
```

T에 대한 템플릿 선언: 멤버 함수 정의할 때마다 선언해 주어야 한다

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

template <typename T>
T Add(T a, T b){
```

```
        return a+b;
    }

class Point{
    int x, y;
public:
    Point(int _x=0, int _y=0):x(_x),y(_y){}
    void ShowPosition();

    Point operator+(const Point& p);
};

void Point::ShowPosition(){
    cout<<x<<" "<<y<<endl;
}

Point Point::operator+(const Point& p){
    return Point(x+p.x, y+p.y);
}

int main(){
    Point p1(1,2);
    Point p2(1,2);

    Point p3=Add(p1, p2);
    p3.ShowPosition();

    return 0;
}
```

Point 객체를 Add 함수 템플릿의 인자로 전달

Point 객체가 + operation이 가능하도록 operator overloading 해줌