Week 8-1 : C++ Templates & Standard Library

Part1. Template

Function template

Tools for making function

Make a several data type function.

The definition of a function template : template <typename T> or template <class T>

```
Template <typename typename>
return-type functionname(…)
{
   …
}
```

- Function template

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

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

T = type name, making function using T

T can has several data type -> function ; templated function

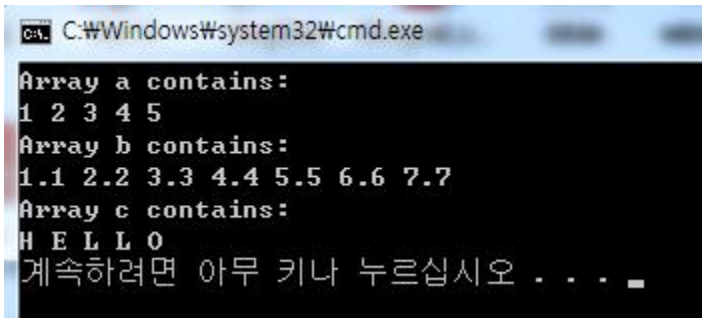
- Templated function

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


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

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

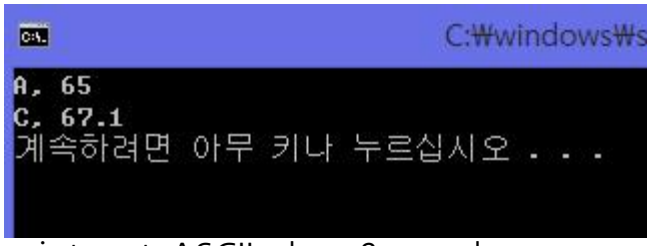
```
#include <iostream>
using std::cout;
using std::endl;

// 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
```



- has several data types

```
#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;
}
```



print out ASCII char & number

- **Class Template**

1. To generate an object based Class Template, declare a data type explicitly.
2. Cannot be decide a data type using parameter like as Function Template
3. constructor's data type and template function's data type can be different, so declare a data type explicitly.

declare and define in a same file.

- Complier: handling template
- Linker:.cpp, .h relationship

```
#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;
}
```

Declare a template about T : declare every time when you define member function.

```
#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 object pass to Add function template

Point object can be execute "+ operation" (operator overloading)

[Exercise]

1. Define a Max function which can print out a max value, a dictionary ordered string, or a max length. You cannot use a function template about string. Because it print out just address value. so you make an another function to compare a dictionary order, and length using char*. This called specialized funcion template. Make a program.

- 3 Max function

- **hint**

  a > b ? a : b

  a true has a value. a false has b value.

```
int main(void)
{
cout<< Max(20,50) <<endl;
cout<< Max('A','Z') <<endl;
cout<< Max(2.2, 9.9) <<endl;
cout<< Max("Programming","Computer") <<endl; //return, parameter type :  const char*

char str1[] = "computer"
char str2[] = "programming"
cout<< Max(str1,str2) <<endl; //return, parameter type : char*

return 0;
}
```

2. Make a exception handling about divide function. In the void Divide function, you can throw value. and In the main function. you try the Divide function and catch the exception about dividing 0.

- hint

```
void Divide (int n1, int n2)
{ if (.....) throw ....}
int main()
try
{
 //execute dividing
}catch(int ...)
{
 // exception handling
}
```