Template is the idea that you define a general variable type for your functions( return type or parameter type ). So you don’t have to define the exactly type (int, char, double … ) when you define the function. Instead of that you just define a general type and actually **you decide the type of variable when you call this function**.

**In other word, your function parameter and return type will be decide when you call the function (usually call functions in main)**

Let’s see the example.

template <class T>

T get\_the\_sum(T x, T y)

{

return x + y;

}

int main()

{

int a1 = 5, a2 = 7;

cout << get\_the\_sum(a1, a2) << endl;

double b1 = 3.14, b2 = 99.999;

cout << get\_the\_sum(b1, b2) << endl;

string c1 = "Hello ", c2 = "World! ";

cout << get\_the\_sum(c1, c2) << endl;

system("pause");

return 0;

}

I define the function “**get\_the\_sum**” as a template function. Just simply add the **template <class T>** before your function. So the system know T is a general type. It will be decided by the parameter you throw into the function parameter(when you call the function).

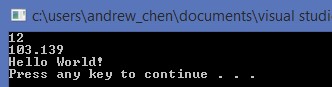
Firstly I put two integer in the function (a1, a2). So the system now knows that

T = “int” type. The function become “int get\_the\_sum(int x, int y)”

The second call I put two double variable (b1, b2) in the function parameter. So now T = “double”, The function become “double get\_the\_sum(double x, double y)”

The same thing that third call will make T = string, so you know we can combine two strings by “+”.The function become “string get\_the\_sum(string x, string y)”

Output:



The previous case show you only one general type ( **class T** ). By that case, both the two parameter and return type have to be the same.

template <class T>

T get\_the\_sum(T x, T y)

{

return x + y;

}

Which means if you put a int type variable for the first parameter, the system now know **T = int**. Since the return type and second parameter are also **T type**, so they must be int type. ( x and y and return type have to be the same because they are all the same type “T”)

How about you want to have different type parameters but they are all general type(decide the real type when you call this function) ?

Still simple ! Just add more general type before you define the function.

Example:

template < class A, class B >

void output\_variables(A x, B y)

{

cout << "The first variable is: " << x << endl;

cout << "The second variable is: " << y << endl;

cout << endl;

}

This function works for printing out what ever type of variables you put in the function parameters, since we define two general type template < class A, class B >

So the two parameters can be different types.

Calling statement:

int main()

{

int a1 = 5, a2 = 7;

double b1 = 3.14, b2 = 99.999;

string c1 = "Hello ", c2 = "World! ";

output\_variables(a1, b1);

output\_variables(b2, c1);

output\_variables(c1, c2);

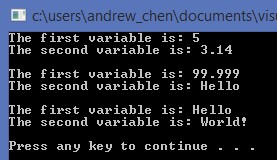
system("pause");

return 0;

}

Let’s just see the first calling **output\_variables(a1, b1).** Since I put an int(a1) for the first parameter, now system will know type A = int. And the same I put a double(b1) for the second parameter, the type B = double.

Output:



Now you know how to use template function, let’s briefly talk about template works in class. I’m going to just show you the example here, and you can ask question in SI session.

Here is how you define template for class. The same, you can define more than one general type if necessary.

#include <iostream>

#include <string>

using namespace std;

template<class T>

class test

{

private:

string name;

T price;

public:

test()

{

name = "Not setting yet";

price = 0;

}

test(string n, T pr)

{

name = n;

price = pr;

}

void set\_name(string s)

{

name = s;

}

void set\_price(T pr)

{

price = pr;

}

void display()

{

cout << "The product " << name << " is sale for " <<

price << " dollars" << endl;

}

};

int main()

{

test<int> s;

s.set\_name("frig");

s.set\_price(100);

s.display();

test<double> p("apple", 1.56);

p.display();

system("pause");

return 0;

}

Output:

