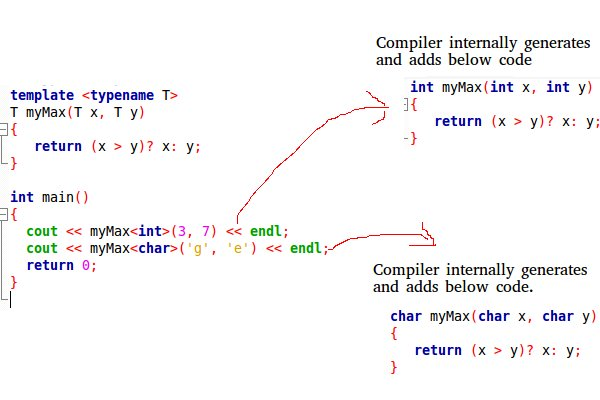
**Template**

**What and Why?**

Templates are the foundation of generic programming. The simple idea is to pass data type as a parameter so that we don’t need to write same code for different data types.

**How?**

Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.



Types of template:

1. Function template

2. Class template

Function template:

A generic function that can be used for different data types.

**Pgm**

#include<iostream>

template <typename Z>

Z mySum(Z a, Z b)

{

return a+b;

}

int main()

{

std::cout << mySum(3,7) << std::endl;

std::cout << mySum(1.0,2.5) << std::endl;

std::cout << mySum('a', '.') << std::endl;

return 0;

}

Class template:

Like function templates, class templates are useful when a class defines something that is independent of data type.

**Pgm**

#include<iostream>

using namespace std;

template<class T, class U>

class A {

T x;

U y;

public:

A() { cout<<"Constructor Called"<<endl; }

};

int main() {

A<char, char> a;

A<int, double> b;

return 0;

}