

# Templates in C++

Generic programming

$I + I \rightarrow \text{add}(\text{int}, \text{int})$

$F + F \rightarrow \text{add}(\text{float}, \text{float})$

$I + F \rightarrow \text{add}(\text{int}, \text{float})$

$F + I \rightarrow \text{add}(\text{float}, \text{int})$

} function overloading  
(Polymorphism)

✳ We want to create single function or single class that works with different data types. This can be done using templates.

## How templates works:-

Templates in C++ works in such a that it gets expended at compile time, just like macros & allow a function or class to work on different data types without being rewritten.

Macros:-

#define MAX 5

Template  class  
function

## Class Template

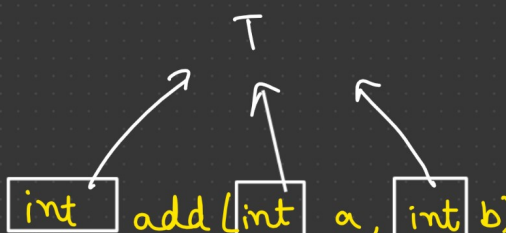
Syntax:-

typedef <class T> = template <class T>

```
class Addition  
{
```

```
public:
```

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



```
};
```

```
int main()  
{
```

```
    Addition <int> a;
```

```
    int x = a.add(2, 3);
```

```
}
```

## For more than One Data Type:-

```
template< class T1, class T2, class T3>
```

```
class Addition
```

```
{
```

```
    public:
```

```
        T3 add(T1 a, T2 b)
```

```
        {
```

```
            return a+b;
```

```
        }
```

```
};
```

```
int main()
```

```
{
```

```
    Addition <int, float, float> a;
```

```
    float y = a.add(2, 3.5);
```

```
    cout<<y;
```

```
}
```

For default Data type:-

```
template <class T1, class T2 = int, class T3 = float>
```

```
int main()
{
    Addition <int> a;
    float y = a.add(2, 3);
    cout << y;
}
```

## Function Template

If you want to create only few functions of your class to be generic, then you can use function template.

```
class Maximum
{
public:
    template <class T>
    T max(T a, T b)
    {
        return a > b ? a : b;
    }
};

int main()
{
    Maximum m;

    int x = m.max<int>(2, 3);

    cout << x;

    return 0;
}
```