

## TEMPLATES : template functions, template classes

---

### TEMPLATE FUNCTIONS

Anytime you find yourself writing several functions which perform the same function on different data types, you should consider writing a template function. Although you could 'overload' a function to perform the same operations on each data type needed, template functions allow you to write the code once. The compiler performs the work of creating the needed code for each data type used in the program. Here's an example:

```
// template.cp
#include <iostream.h>

template <class T> T abs( T val )

    if( val < 0 )
        val = -val;
    return val;
;

void main( void )

    int  i1 = -4;
    int  i2 = 237;
    float f1 = -3.14;
    float f2 = 5.83;

    cout << abs(i1) << endl;
    cout << abs(i2) << endl;
    cout << abs(f1) << endl;
    cout << abs(f2) << endl;

// end template.cp
```

The keyword 'class' in the declaration 'template <class T>' has nothing to do with class structures, and should be read something like "a template of class T".

### TEMPLATE CLASSES

You can also create template classes in much the same way as you create template functions. Template classes can be used to define container classes to make it easy for you to create linked lists of all sorts of data without having to write a whole bunch of code. The following is a simple example of using a template class that keeps a running total.

```
// sum.cp
#include <iostream.h>

template<class T> class Sum

    T totalValue;
    T lastValue;
public:
    Sum();
    ~Sum();
    void add(T &);
    T total(void);
    T last(void);
    void undo(void);
```

```

;

template<class T> Sum<T>::Sum()

    // constructor
    totalValue = (T)0;
    lastValue = (T)0;

template<class T> Sum<T>::~~Sum()

    // destructor

template<class T> void Sum<T>::add(T &val)

    lastValue = val;
    totalValue = totalValue + lastValue;

template<class T> T Sum<T>::total(void)

    return totalValue;

template<class T> T Sum<T>::last(void)

    return lastValue;

template<class T> void Sum<T>::undo(void)

    totalValue = totalValue - lastValue;
    lastValue = -lastValue;

void main( void )

    Sum<int> RunningTotal;

    int i1 = 4;
    int i2 = 8;
    int i3 = 1;
    int i4 = 7;

    RunningTotal.add(i1);
    RunningTotal.add(i2);
    RunningTotal.add(i3);
    RunningTotal.add(i4);

    cout << "Total = " << RunningTotal.total() << endl;
    cout << "Last Item = " << RunningTotal.last() << endl;

    cout << "Undo Last Operation..." << endl;
    RunningTotal.undo();
    cout << "Total = " << RunningTotal.total() << endl;

    cout << "Undo Last Operation (again)..." << endl;
    RunningTotal.undo();
    cout << "Total = " << RunningTotal.total() << endl;

// end sum.cp

```