Suppose that you are writing a C++ program that requires two stacks--one for integer data and one for string data. You could implement these classes as follows. The implementation below isn't very good (it limits the stack to 1000 elements and it does no error checking), but it will serve as an example.

|  |  |
| --- | --- |
| class IntStack {  public:  IntStack() { top = -1; }  void push(int i)  { st[++top] = i; }  int pop()  { return st[top--]; }  private:  int top;  int st[100];  }; | class StringStack {  public:  StringStack() { top = -1; }  void push(string i)  { st[++top] = i; }  string pop()  { return st[top--]; }  private:  int top;  string st[100];  }; |

The main program would declare and use these stacks as follows:

IntStack ii;

StringStack ss;

ii.push(25);

ss.push("Hello");

Notice that the only difference between these classes (other than the name of the class) is the type of data that's put onto the stack. C++ allows you to define a single, template class to represent a stack of any possible datatype (including a user-defined datatype). The declaration would look like this:

#ifndef STACK\_H

#define STACK\_H

template <class T>

class Stack {

public:

Stack() { top = -1; }

void push(T i)

{ st[++top] = i; }

T pop()

{ return st[top--]; }

private:

int top;

T st[100];

};

#endif

The T represents the type of stack desired. The main program would declare and use the stacks as follows:

Stack<int> ii;

Stack<string> ss;

ii.push(25);

ss.push("Hello");

There is never an implementation file (\*.cpp) for a template class. All of the member functions should be declared in the header file (in this case, table.h). The following shows how the functions for this class would be declared after the declaration of the class.

#ifndef STACK\_H

#define STACK\_H

template <class T>

class Stack {

public:

Stack();

void push(T i);

T pop();

private:

int top;

T st[100];

};

template <class T>

Stack<T>::Stack()

{

top = -1;

}

template <class T>

void Stack<T>::push(T i)

{

st[++top] = i;

}

template <class T>

T Stack<T>::pop()

{

return st[top--];

}

#endif