C++ Templates

C++ Templates

- We need a stack of operands and a stack of operators.
- Operands can be integers and floating point numbers, even variables.
- Operators are single characters.
- We would have to create classes FloatStack and CharStack.
- Yet the internal workings of both classes is the same.

C++ Templates

- We can use C++ Templates to create a “template” of a stack class.
- Instantiate float stack, char stack, or stack for any type of element we want.

Stack using templates

Stack.h:

```
template <class T>
class Stack {
public:
    Stack();
    int empty(void); // 1=true, 0=false
    void push(T & x);
    T pop(void);

    ~Stack();
private:
    int top;
    T* A;//char * A;// int * A;
};
```

Stack using templates

Stack.cpp

```
#include <iostream.h>
#include <stdlib.h>
#include "Stack.cpp"

#define MAXSTACKSIZE 50

template <class T>
Stack<T>::Stack()
{
    top = -1;
    A = new T[MAXSTACKSIZE];
}
```

Stack using templates

Stack.cpp

```
template <class T>
Stack<T>::~Stack()
{
    delete A;
}

template <class T>
int Stack<T>::empty(void)
{
    if( top < 0 ) return 1;
    return 0;
}
```

Stack using templates

Stack.cpp

```
template <class T>
void Stack<T>::push(T& x)
{
    if( top < MAXSTACKSIZE ) {
        A[++top] = x;

    }
    cout << "stack overflow in push.\n";
}
```

Case 2:
//push
cout<<“enter value?”;
char y;
cin>>y;
s.push(y);

Stack using templates

Stack.cpp

```
template <class T>
T Stack<T>::pop(void)
{
  T x;
  x = A[top--];
  return x;
}
```

Main.cpp

```
if(!isempty())
{
   cout<<pop()<<endl;
}
```

Stack using templates

main.cpp

```
#include "Stack.cpp"
Int main() {
  Stack<int> intstack;
  Stack<char> charstack;
  int x=10, y=20;
  char c='C', d='D';

  intstack.push(x);   intstack.push(y);
  cout << "intstack: " << intstack.pop() << ", "
       << intstack.pop() << "\n";

  charstack.push(c); charstack.push(d);
  cout << "charstack: " << charstack.pop() << ",
  "
       << charstack.pop() << "\n";
}
```