Template Classes

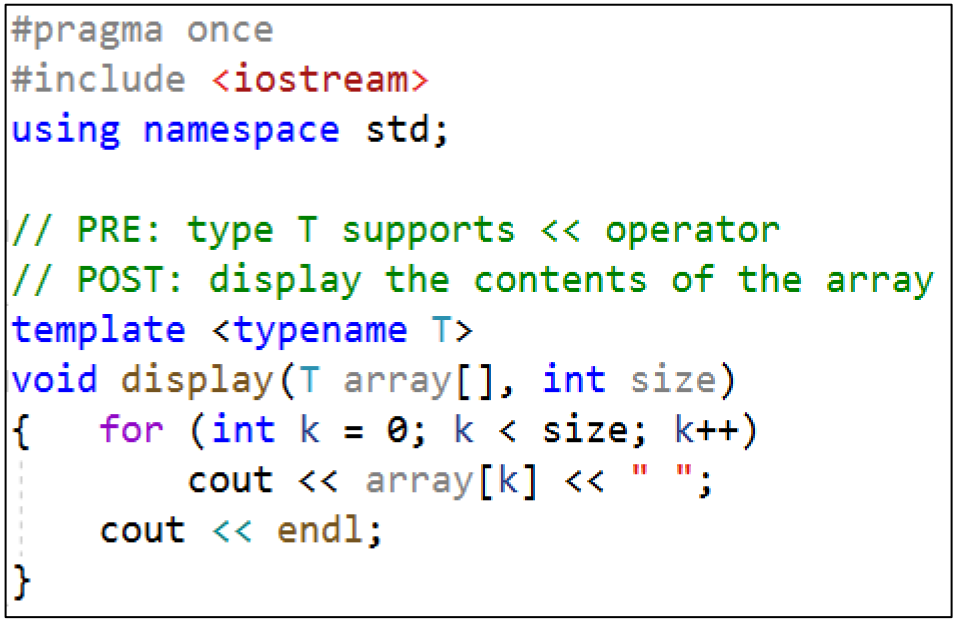
Template Functions

Template Classes

Template Display Function

A template display function (display.h) is written to display contents of an array of some type T

The PRE condition lists operators must be supported by type T



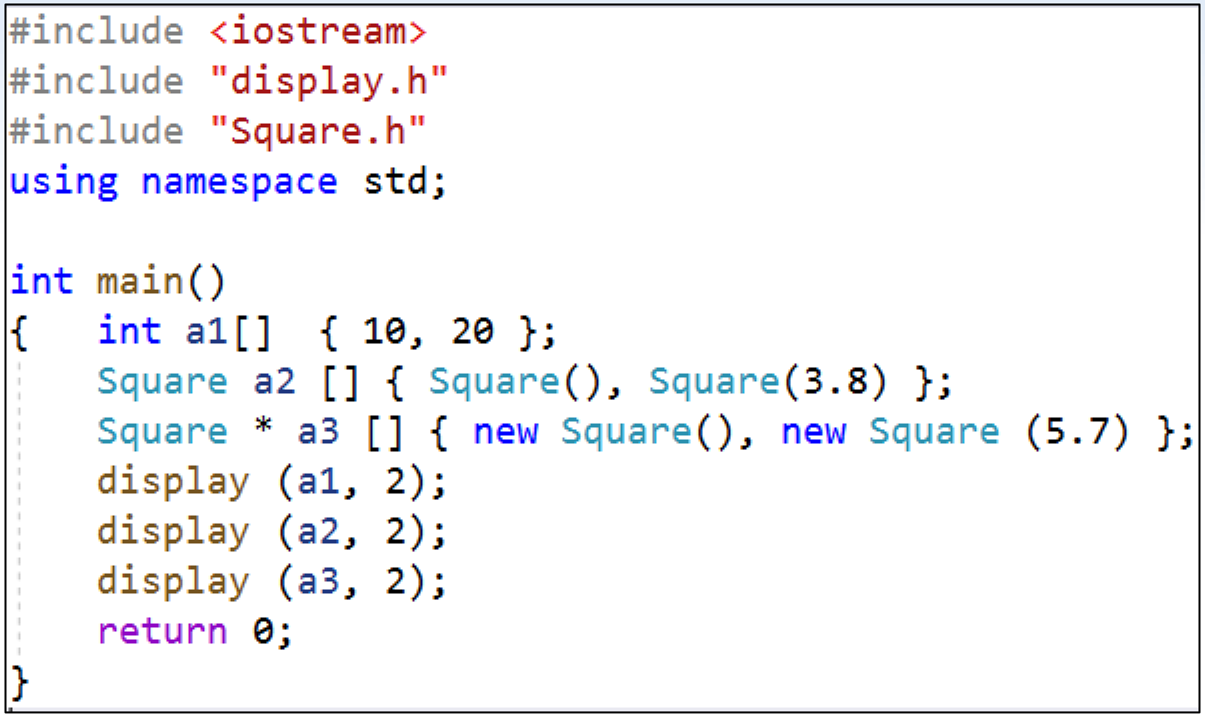
```
#pragma once
#include <iostream>
using namespace std;

// PRE: type T supports << operator
// POST: display the contents of the array
template <typename T>
void display(T array[], int size)
{   for (int k = 0; k < size; k++)
        cout << array[k] << " ";
    cout << endl;
}
```

Square Class Overloads <<

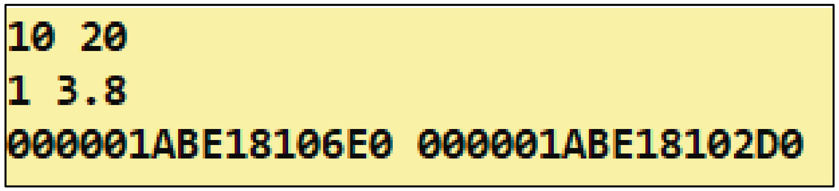
```cpp
#pragma once
#include <ostream>
using namespace std;

class Square
{   public:
        Square();
        Square (double d);
        void setSide (double s);
        double getSide () const;
    private:
        double side;
};

ostream& operator<< (ostream& out, const Square& square);
```

```cpp
#include "Square.h"

Square::Square()
{   side = 1.0;
}
Square::Square(double d)
{   side = d;
}
void Square::setSide(double s)
{   side = s;
}
double Square::getSide() const
{   return side;
}

// Note: use public accessor function getSide as
//       operator<< is NOT a class member
ostream& operator<< (ostream& out, const Square& square)
{   out << square.getSide();
    return out;
}
```

Using the Template Function

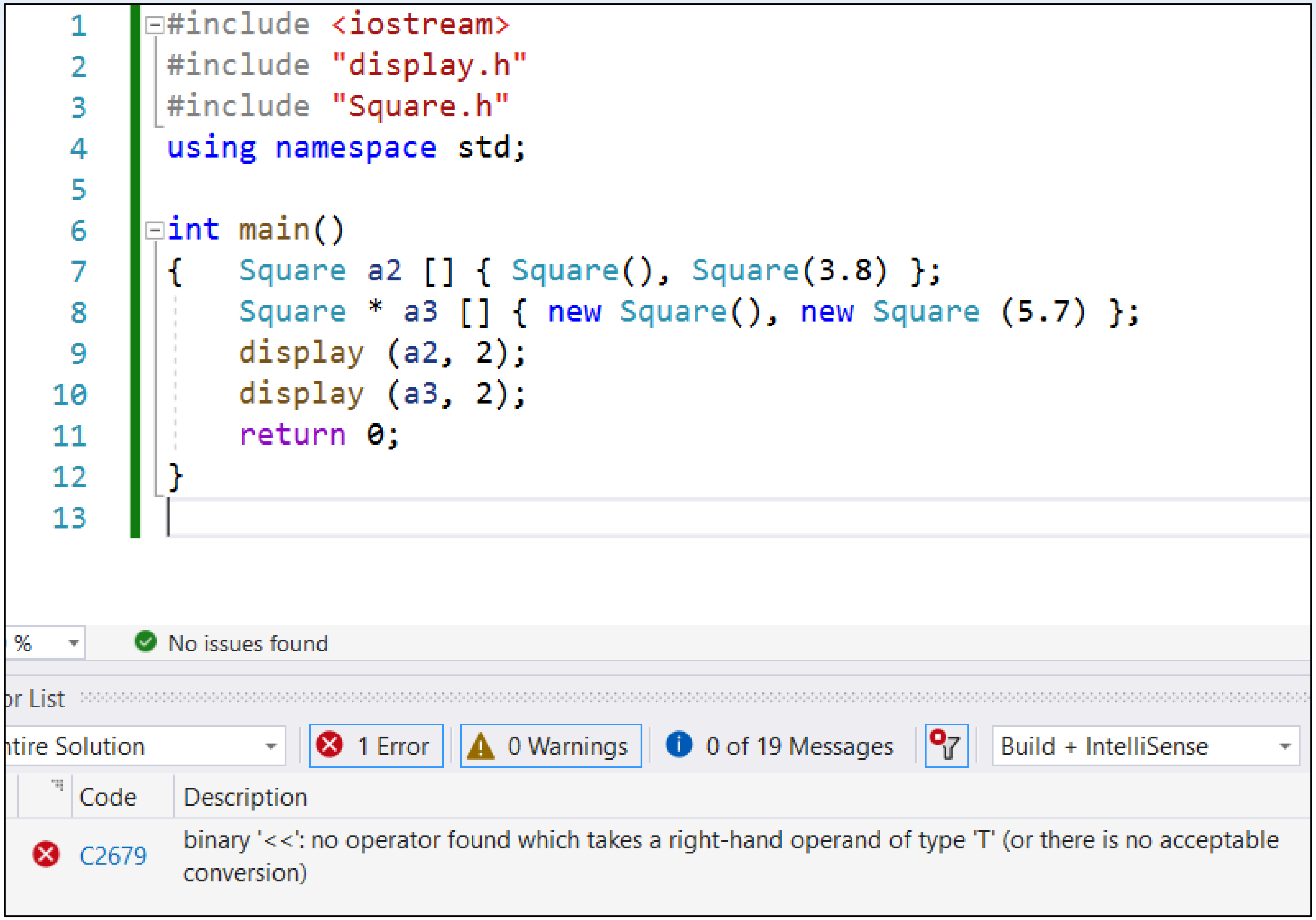


```
#include <iostream>
#include "display.h"
#include "Square.h"
using namespace std;

int main()
{   int a1[]  { 10, 20 };
    Square a2 [] { Square(), Square(3.8) };
    Square * a3 [] { new Square(), new Square (5.7) };
    display (a1, 2);
    display (a2, 2);
    display (a3, 2);
    return 0;
}
```



```
10 20
1 3.8
000001ABE18106E0 000001ABE18102D0
```

Remove Square << Yields Error



```
#include <iostream>
#include "display.h"
#include "Square.h"
using namespace std;

int main()
{   Square a2 [] { Square(), Square(3.8) };
    Square * a3 [] { new Square(), new Square (5.7) };
    display (a2, 2);
    display (a3, 2);
    return 0;
}

```

Code	Description
C2679	binary '<<': no operator found which takes a right-hand operand of type 'T' (or there is no acceptable conversion)

Template Stack Class

A template class supports an underlying container (ex. array) of unknown type T

The IntStack class we wrote is a good candidate to become a template class

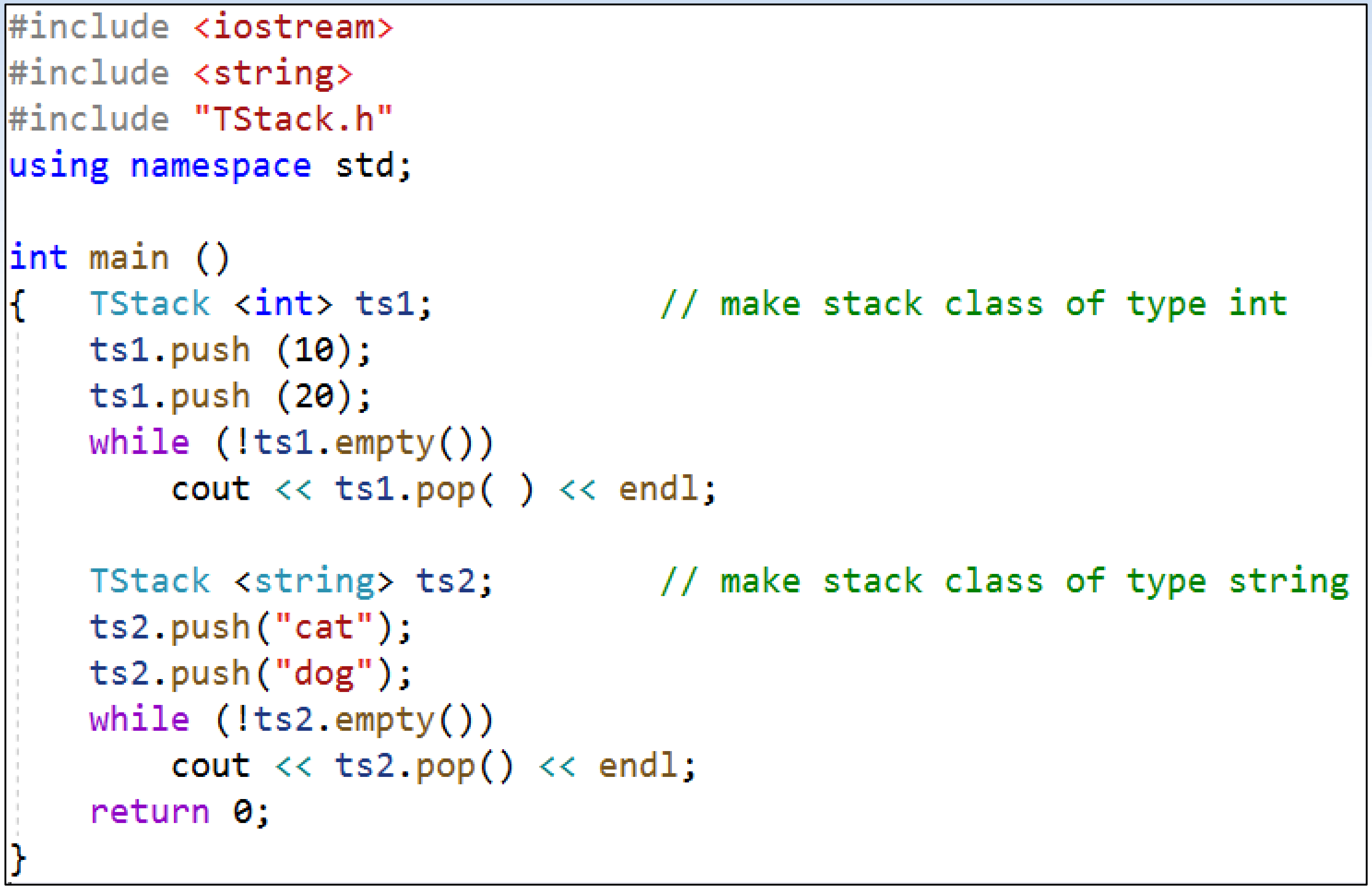
A template class is completely contained in a header file (ex. TStack.h) as it cannot be compiled

```
template <typename T>
class TStack {
public:
    TStack();
    ~TStack();
    void push(T item);
    T pop();
    T peek() const;
    bool empty() const;
    int count() const;
private:
    T* stack;
    int capacity;
    int size;
};
```

```
template <typename T>
TStack<T>::TStack()
{   capacity = 10;
    stack = new T[capacity];
    size = 0;
}

template <typename T>
TStack<T>::~TStack()
{   delete[] stack;
}

template <typename T>
void TStack<T>::push(T item)
{   if (size == capacity)
    {   capacity = 2 * capacity;
        T* temp = new T[capacity];
        for (int k = 0; k < size; k++)
            temp[k] = stack[k];
        delete[] stack;
        stack = temp;
    }
    stack[size] = item;
    size++;
}
```

```
template <typename T>
T TStack<T>::pop()
{   T topItem = stack[size-1];
    size--;
    return topItem;
}

template <typename T>
T TStack<T>::peek() const
{   return stack[size-1];
}

template <typename T>
bool TStack<T>::empty() const
{   return size == 0;
}

template <typename T>
int TStack<T>::count() const
{   return size;
}
```

Each function uses TStack<T>



```cpp
#include <iostream>
#include <string>
#include "TStack.h"
using namespace std;

int main ()
{   TStack <int> ts1;             // make stack class of type int
    ts1.push (10);
    ts1.push (20);
    while (!ts1.empty())
        cout << ts1.pop( ) << endl;

    TStack <string> ts2;          // make stack class of type string
    ts2.push("cat");
    ts2.push("dog");
    while (!ts2.empty())
        cout << ts2.pop() << endl;
    return 0;
}
```

```
20
10
dog
cat
```