

Motivation for Templates

Templates 1

You want both:

- a list of `Location` objects
- a list of `MazeMonster` objects

How can you accomplish this by writing one `LinkedList` class?

- state all the ways you can think of doing this

- state the pros/cons of each method

Computer Science Dept/Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

One Way to Look at Templates...

Templates 2

Until now, we have used variables:

- The type of a variable is fixed when you write code.
- The value of a variable isn't fixed when you write code.

With templates, type isn't fixed when you write code!

With templates, you use a type more or less as a variable!

Computer Science Dept/Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Example: Queue of some type `Foo`

Templates 3

C++ keywords

Name of template

```

template <class Foo> class QueueT {
private:
    Foo Buffer[100];
    int Head, Tail, Count;

public:
    QueueT();
    void Enqueue(const Foo& item);
    Foo Dequeue();
    ~QueueT();
};

```

Parameter,
can be any type

The header of a templated class declaration specifies one or more type names which are then used within the template declaration.

These type names are typically NOT standard or user-defined types, but merely placeholder names that serve as formal parameters.

Computer Science Dept/Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

C++ Templates

Templates 4

Definition of "template":
Parameterized class with formal parameters that denote unknown types.

Usage:
In situations where the same algorithms and/or data structures need to be applied to different data types.

Declaration syntax:

```

template <class Foo> class Queue {
    // template member declarations go here
};

```

Computer Science Dept/Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

What can a parameter be used for? Templates 5

To specify the type specifying of data which will be local to objects of the class:

```
private:
    Foo Buffer[100];
```

To specify the type of a parameter to a class member function:

```
void Enqueue(const Foo& item);
```

To specify the return type of a class member function:

```
Foo Dequeue();
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Instantiating a Template Templates 6

Given the template declaration:

```
template <class Foo> class QueueT {...};
```

Instantiate QueueT of objects in two ways:

- QueueT<Location> **LocationQueue**;
- typedef QueueT<int> IntegerQueue;

Each of these defines an **object** which is derived from the type QueueT.

Note how an actual type (Location or int) is substituted for the formal template parameter (Foo) in the object declaration.

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Usage of Templates Templates 7

Once created, the template object is used like any other object:

```
intQueue.Enqueue(100);    // add 100 to the queue
intQueue.Enqueue(200);    // add 200
```

The parameter type for the member function Insert() was specified as Foo in the template declaration and mapped to int in the declaration of the object intQueue. When calling Insert() we supply an int value.

```
int x = intQueue.Dequeue();    // remove 100
intQueue.Enqueue(300);    // queue now
                                  // has (200,300)
int Sz = intQueue.Size();    // size is 2
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Compiler view of templates... Templates 8

The compiler macro expands the template code:

- You write Queue<int> **intQueue**.
- Compiler emits new copy of a class named, say, "Queueint" and substitutes "int" for "Foo" throughout.
- Therefore, the compiler must have access to the parameterized implementation of the template member functions in order to carry out the substitution.
- Therefore, the template implementation CANNOT be pre-compiled.
- Most commonly, all template code goes in the header file with the template declaration.

The compiler "maps" the declaration:

```
private:
    Foo Buffer[100];
```

to the declaration:

```
private:
    int Buffer[100];
```

The compiler "mangles" the template name with the actual parameter (type name) to produce a unique name for the class.

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Implementing Template Class Methods

Templates 9

template and formal parameter(s)

Class name and formal parameter(s)

Scope resolution operator and function name

```
template <class Foo> QueueT<Foo>::Queue() {
    // ... member function body goes here
}
```

Return type goes here:

```
template <class Foo> void QueueT<Foo>::Enqueue(Foo item) {
    // ... member function body goes here
}
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

A Complete Template Queue Class

Templates 10

```
// QueueT.h
#ifndef QUEUE_T_H
#define QUEUE_T_H
#include <cassert>
const int Size = 100;
template <class Foo> class QueueT {
private:
    Foo Buffer[Size];
    int Head, Tail, Count;
public:
    QueueT();
    void Enqueue(const Foo& Item);
    Foo Dequeue();
    int getSize() const;
    bool isEmpty() const;
    bool isFull() const;
    ~QueueT();
};
// ... template implementation goes here
#endif
```

Using the template parameter: data type, parameter type, return type.

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

A Complete Template Class

Templates 11

```
// ... continuing header file QueueT.h

template <class Foo> QueueT<Foo>::QueueT() : Head(0),
                                           Tail(0), Count(0) {
}

template <class Foo> void QueueT<Foo>::Enqueue(Foo Item) {

    assert(Count < Size);    // die if Queue is full!

    Buffer[Tail] = Item;
    Tail = (Tail + 1) % Size; // circular array indexing
    Count++;
}
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

A Complete Template Class

Templates 12

```
// ... continuing header file QueueT.h

template <class Foo> Foo QueueT<Foo>::Dequeue() {

    assert(Count > 0);    // die if Queue is empty

    int oldHead = Head;    // remember where old Head was
    Head = (Head + 1) % Size; // reset Head
    Count--;
    return Buffer[oldHead]; // return old Head
}

template <class Foo> int QueueT<Foo>::getSize() const {

    return (Count);
}
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

A Complete Template Class

Templates 13

```
// . . . continuing header file QueueT.h

template <class Foo> bool QueueT<Foo>::isEmpty() const {

    return (Count == 0);

}

template <class Foo> bool QueueT<Foo>::isFull() const {

    return (Count >= Size);

}

template <class Foo> QueueT<Foo>::~QueueT() {

}

// . . . end template QueueT<Foo> implementation
```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

A Driver for the QueueT Template

Templates 14

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "QueueT.h"

void main() {

    const int numVals = 10;
    QueueT<int> intQ;

    for (int i = 0; i < numVals; i++) {
        intQ.Enqueue(i*i);
    }

    int Limit = intQ.getSize();
    for (i = 0; i < Limit; i++) {
        int nextVal = intQ.Dequeue();
        cout << setw(3) << i << setw(5) << nextVal << endl;
    }

}
```

0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Recap

Templates 15

Note that method bodies use the same algorithms for a queue of ints or a queue of doubles or a queue of Locations...

But the compiler still type checks!

It does a macro expansion, so if you declare

```
QueueT<int>    iQueue;
QueueT<char>  cQueue;
QueueT<Location> Vertices;
```

the compiler has instantiated three different classes after expansion to use with normal type checking rules.

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Implicit Assumptions in QueueT

Templates 16

Declaration of the array of Fools assumes Foo has a default constructor:

```
template <class Foo> class QueueT {
private:
    Foo Buffer[Size];
    ...
};
```

Assignment of Fools assumes Foo has appropriately overloaded the assignment operator:

```
template <class Foo> void QueueT<Foo>::Enqueue(Foo item) {
    ...
    Buffer[tail] = item;
    ...
};
```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Implicit Assumptions in QueueT

Templates 17

The way that Foos are returned by Dequeue() method assumes Foo has provided an appropriate copy constructor:

```
template <class Foo> Foo QueueT<Foo>::Dequeue() {
    ...
    return Buffer[oldHead];
}
```

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Variable and Constant Template Parameters

Templates 18

Template parameters may be:

- type names (we saw this previously)
- variables e.g., to specify a size for a data structure
- constants useful to define templates for special cases
(not terribly useful)

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Queue Template with a Variable Parameter

Templates 19

One weakness of the QueueT template is that the queue array is of a fixed size. We can easily make that user-selectable:

```
template <class Foo, int Size> class QueueT {
private:
    Foo buffer[Size];
    int Head,
        Tail;
    int Count;
public:
    QueueT();
    bool Enqueue(Foo Item);
    bool Dequeue(Foo& Item);
    int getSize() const;
    bool isEmpty() const;
    bool isFull() const;
    ~QueueT();
};
```

Second template parameter is just an int variable, which falls within the class scope just as a private data member would.

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Driver for Revised Queue Template

Templates 20

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "QueueT.h"
void main() {
    const int smallSize = 10;
    const int largeSize = 100;

    QueueT<int, smallSize> smallQ;
    QueueT<int, largeSize> largeQ;

    for (int i = 0; i < smallSize-1; i++)
        smallQ.Enqueue(i);

    for (i = 0; i < largeSize-1; i++) {
        largeQ.Enqueue(i);

        for (i = 0; i < largeSize-1; i++) {
            int nextVal;
            largeQ.Dequeue(nextVal);
            cout << setw(3) << i << setw(5) << nextVal << endl;
        }
    }
}
```

The value specified in the declaration must still be a constant though...

... that could be avoided by redesigning the template to take the array size as a parameter to a constructor...

Computer Science Dept Va Tech March 2001 OO Software Design and Construction ©2001 McQuain WD & Keller BJ

Template Type-checking

Templates 21

Suppose we have the declarations:

```
QueueT<int, 100>    smallIntegerQueue;
QueueT<int, 1000>  largeIntegerQueue;
QueueT<int, 1000>  largeIntegerQueue2;
QueueT<float, 100> smallRealQueue;
QueueT<float, 1000> largeRealQueue;
```

Which (if any) of the following are legal assignments:

```
smallIntegerQueue = largeIntegerQueue;

smallIntegetQueue = smallRealQueue;

largeIntegerQueue = largeIntegerQueue2;
```

Computer Science Dept/Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template

Templates 22

```
// LinkedList.h
#ifndef LINKLISTT_H
#define LINKLISTT_H

#include <cassert>
#include "LinkNodeT.h"

template <class Item> class LinkedList {
private:
    LinkNodeT<Item>* Head; // points to head node in list
    LinkNodeT<Item>* Tail; // points to tail node in list
    LinkNodeT<Item>* Curr; // points to "current" node in list

public:
    LinkedList();
    LinkedList(const LinkedList<Item>& Source);
    ~LinkedList();
    ...
};
```

These are pointers to template objects so any template parameters must be specified explicitly.

Function parameter is a template object, so...

Computer Science Dept/Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template

Templates 23

```
...
bool isEmpty() const;
bool atEnd() const;
bool PrefixNode(Item newData);
bool AppendNode(Item newData);
bool InsertAfterCurr(Item newData);
bool Advance();
void gotoHead();
void gotoTail();
bool MakeEmpty();
bool DeleteCurrentNode();
bool DeleteValue(Item Target);
Item getCurrentData() const;
void PrintList(ostream& Out);
LinkedList<Item>& operator=(const LinkedList<Item>&
                           Source);
};
#include "LinkedList.cpp"
#endif
```

Operator return type is a template object, so...

Computer Science Dept/Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Node Template

Templates 24

```
// LinkNodeT.h
#ifndef LINKNODET_H
#define LINKNODET_H

template <class Item> class LinkNodeT {
private:
    Item Data;
    LinkNodeT<Item>* Next;

public:
    LinkNodeT();
    LinkNodeT(Item newData);
    void setData(Item newData);
    void setNext(LinkNodeT<Item>* newNext);
    Item getData() const;
    LinkNodeT<Item>* getNext() const;
};
#include "LinkNodeT.cpp"
#endif
```

Function return type is a pointer to a template object, so...

Computer Science Dept/Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

A Node Template Constructor

Templates 25

```

////////////////////////////////////
// Constructor for LinkNode objects with assigned
// Data field.
//
// Parameters:
//   newData  Data element to be stored in node
// Pre:      none
// Post:     new LinkNode has been created with
//           given Data field and NULL
//           pointer
//
template <class Item>
LinkNodeT<Item>::LinkNodeT(Item newData) {
    Data = newData;
    Next = NULL;
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Node Template Mutators

Templates 26

```

////////////////////////////////////
// Sets new value for Data element of object.
//
// Parameters:
//   newData  Data element to be stored in node
// Pre:      none
// Post:     Data field of object has been
//           modified to hold newData
//
template <class Item>
void LinkNodeT<Item>::setData(Item newData) {

    Data = newData;
}

////////////////////////////////////
// Suppressed to save space.
//
template <class Item>
void LinkNodeT<Item>::setNext(LinkNodeT<Item>* newNext) {

    Next = newNext;
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Node Template Reporters

Templates 27

```

////////////////////////////////////
// Returns value of Data element of object.
//
// Parameters: none
// Pre:      object has been initialized
// Post:     Data field of object has been
//           returned
//
template <class Item>
Item LinkNodeT<Item>::getData() const {

    return Data;
}

////////////////////////////////////
// Suppressed to save space.
//
template <class Item>
LinkNodeT<Item>* LinkNodeT<Item>::getNext() const {

    return Next;
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template Destructor

Templates 28

```

////////////////////////////////////
// Destructor for LinkListT objects.
//
// Parameters: none
// Pre:      LinkListT object has been constructed
// Post:     LinkListT object has been destructed;
//           all dynamically-allocated nodes
//           have been deallocated.
//
template <class Item> LinkListT<Item>::~LinkListT() {

    LinkNodeT<Item>* toKill = Head;

    while ( toKill != NULL) {
        Head = Head->getNext();
        delete toKill;
        toKill = Head;
    }
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template Prefix Function

Templates 29

```

////////////////////////////////////
// Inserts a new LinkNodeT at the front of the list.
//
template <class Item> bool LinkListT<Item>::PrefixNode(Item
newData) {

    LinkNodeT<Item>* newNode =
                                new LinkNodeT<Item>(newData);

    if (newNode == NULL) return false;

    if ( isEmpty() ) {
        newNode->setNext(NULL);
        Head = Tail = Curr = newNode;
        return true;
    }
    newNode->setNext(Head);
    Head = newNode;

    return true;
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template Assignment Overload

Templates 30

```

////////////////////////////////////
// Deep copy assignment operator for LinkListT objects.
//
template <class Item>
LinkListT<Item>& LinkListT<Item>::
operator=(const LinkListT<Item>& Source) {

    if (this != &Source) {

        MakeEmpty();           // delete target's list, if any

        LinkNodeT<Item>* myCurr = Source.Head; // copy list

        while (myCurr != NULL) {
            Item xferData = myCurr->getData();
            AppendNode(xferData);
            myCurr = myCurr->getNext();
        }

        return *this;
    }
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Linked List Template Copy Constructor

Templates 31

```

////////////////////////////////////
// Deep copy constructor for LinkListT objects.
//
template <class Item>
LinkListT<Item>::LinkListT(const LinkListT<Item>& Source) {

    Head = Tail = Curr = NULL;

    LinkNodeT<Item>* myCurr = Source.Head; // copy list

    while (myCurr != NULL) {
        Item xferData = myCurr->getData();
        AppendNode(xferData);
        myCurr = myCurr->getNext();
    }
}

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Template Functions

Templates 32

The template mechanism may also be used with non-member functions:

```

template <class Foo> Swap(Foo& First, Foo& Second) {
    Foo tmpFoo = First;
    First      = Second;
    Second     = tmpFoo;
}

```

Given the template function above, we may swap the value of two variables of ANY type, provided that a correct assignment operation and copy constructor are available.

However, the two actual parameters MUST be of EXACTLY the same type:

```

double X = 3.14159;
int     a = 5;
Swap(a, X);           // error at compile time

```

Computer Science Dept Va Tech March 2001

OO Software Design and Construction

©2001 McQuain WD & Keller BJ

Template Sort Function

Templates 33

```
template <class Foo> InsertionSort(Foo* const A, int Size) {  
  
    int Begin, Look;  
    Foo Item;  
  
    for (Begin = 1; Begin < Size; Begin++) {  
        Look = Begin - 1;  
        Item = A[Begin];  
        while ( Look >= 0 && A[Look] > Item) {  
            A[Look + 1] = A[Look];  
            Look--;  
        }  
        A[Look + 1] = Item;  
    }  
}
```

This will use the insertion sort algorithm to sort an array holding ANY type of data, provided that there are > and deep = operators for that type (if a deep assignment is logically necessary).