Lecture 19: templates and the standard template library
Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides
Outline

• Announcements
• Special Topics
• Review
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
Announcements: Projects

• 3G: due Monday
• 3H: due “Friday”
• 4B: due “Friday”
• 4C: due “Friday”
• “Friday” → not late until June 11th
• And: all work must be turned by June 11th
• And: Project 3 grading may be simplified
Announcements: Logistics

• Office Hours: right here after class
• Lab: no Lab this week!
• Final: Monday June 8\textsuperscript{th} @ 2:45pm in this room
• Grading...
Outline

• Announcements
• **Special Topics**
• Review
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
Special Topics

- Virtual functions in constructors
- const Image with Update
Outline

• Announcements
• Special Topics
• **Review**
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
“this”: pointer to current object

• From within any struct’s method, you can refer to the current object using “this”
How methods work under the covers (4/4)

The compiler secretly slips “this” onto the stack whenever you make a method call.

It also automatically changes “myInt” to this->myInt in methods.

```cpp
class MyIntClass {
  public:
    int myInt;

  public:
    void IncrementMethod() {
      myInt++;
    }

  protected:
    MyIntClass(int init) : myInt(init) {}  // Constructor

  private:
    MyIntClass() : myInt(0) {}  // Default constructor

  friend void IncrementFunction(MyIntClass &mic);

  MyIntClass MIC(12);
  // Increment function
  FriendIncrementFunction(MIC);
  MIC.IncrementMethod();
  cout << "My int is " << MIC.GetMyInt() << endl;
};

MyIntClass main() {
  FriendIncrementFunction(&MIC);
  MIC.IncrementMethod();
  cout << "My int is " << MIC.GetMyInt() << endl;
  return 0;
}
```

<table>
<thead>
<tr>
<th>Addr.</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000</td>
<td>MIC/myInt</td>
<td>12</td>
</tr>
<tr>
<td>0x8004</td>
<td>mic</td>
<td>0x8000</td>
</tr>
<tr>
<td>0x8000</td>
<td>MIC/myInt</td>
<td>12</td>
</tr>
<tr>
<td>0x8004</td>
<td>this</td>
<td>0x8000</td>
</tr>
</tbody>
</table>
Picking the right virtual function

class A
{
  public:
  virtual const char *GetType() { return "A"; }
};

class B : public A
{
  public:
  virtual const char *GetType() { return "B"; }
};

void ClassPrinter(A *ptrToA)
{
  cout << "ptr points to a " << ptrToA->GetType() << endl;
}

int main()
{
  A a;
  B b;

  ClassPrinter(&a);
  ClassPrinter(&b);
}

So how does the compiler know?

How does it get “B” for “b” and “A” for “a”??
Virtual Function Table

- Let C be a class and X be an instance of C.
- Let C have 3 virtual functions & 4 non-virtual functions.
- C has a hidden data member called the “virtual function table”.
- This table has 3 rows:
  - Each row has the correct definition of the virtual function to call for a “C”.
- When you call a virtual function, this table is consulted to locate the correct definition.
Virtual Function Table

• Let C be a class and X be an instance of C.
• Let C have 3 virtual functions & 4 non-virtual functions
• Let D be a class that inherits from C and Y be an instance of D.
  – Let D add a new virtual function
• D’s virtual function table has 4 rows
  – Each row has the correct definition of the virtual function to call for a “D”.
More notes on virtual function tables

• There is one instance of a virtual function table for each class
  – Each instance of a class shares the same virtual function table

• Easy to overwrite (i.e., with a memory error)
  – And then all your virtual function calls will be corrected
  – Don’t do this! ;)

Calling a virtual function

• Let X be an instance of class C.
• Let the virtual function be the 4\textsuperscript{th} function
• Let the arguments to the virtual function be an integer Y and a float Z.
• Then call:

\[(X.vptr[3])(&X, Y, Z);\]

The pointer to the virtual function pointer (often called a vptr) is a data member of X

The 4\textsuperscript{th} virtual function has index 3 (0-indexing)

Secretly pass “this” as first argument to method
Inheritance and Virtual Function Tables

```cpp
class A {
    public:
        virtual void Foo1();
};

class B : public A {
    public:
        virtual void Foo1();
        virtual void Foo2();
};

class C : public B {
    public:
        virtual void Foo1();
        virtual void Foo2();
        virtual void Foo3();
};
```

This whole scheme gets much harder with virtual inheritance, and you have to carry around multiple virtual function tables.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foo1</td>
<td>Location of Foo1</td>
<td></td>
</tr>
<tr>
<td>Foo2</td>
<td>Location of Foo2</td>
<td></td>
</tr>
<tr>
<td>Foo3</td>
<td>Location of Foo3</td>
<td></td>
</tr>
</tbody>
</table>

Same as B’s
This is how you can treat a C as a B
Virtual Function Table: Summary

- Virtual functions require machinery to ensure the correct form of a virtual function is called.
- This is implemented through a virtual function table.
- Every instance of a class that has virtual functions has a pointer to its class’s virtual function table.
- The virtual function is called via following pointers:
  - Performance issue
Upcasting and Downcasting

• Upcast: treat an object as the base type
  – We do this all the time!
  – Treat a Rectangle as a Shape

• Downcast: treat a base type as its derived type
  – We don’t do this one often
  – Treat a Shape as a Rectangle
    • You better know that Shape really is a Rectangle!!
class A
{
};

class B : public A
{
    public:
        B() { myInt = 5; };
        void       Printer(void) { cout << myInt << endl; };

    private:
        int       myInt;
};

void Downcaster(A *a)
{
    B *b = (B *) a;
    b->Printer();
}

int main()
{
    A a;
    B b;
    Downcaster(&b); // no problem
    Downcaster(&a); // no good
}

fawcett:330 childs$ g++ downcaster.C
fawcett:330 childs$ ./a.out
5
-1074118656

what do we get?
Upcasting and Downcasting

- C++ has a built in facility to assist with downcasting: `dynamic_cast`
- I personally haven’t used it a lot, but it is used in practice
- Ties in to std::exception
Default Arguments

```c
void Foo(int X, int Y = 2) {
    cout << "X = " << X << ", Y = " << Y << endl;
}

int main() {
    Foo(5);
    Foo(5, 4);
}
```

default arguments: compiler pushes values on the stack for you if you choose not to enter them
Booleans

- New simple data type: bool (Boolean)
- New keywords: true and false

```c
int main()
{
    bool b = true;
    cout << "Size of boolean is " << sizeof(bool) << endl;
}
fawcett:330 childs$ g++ Boolean.C
fawcett:330 childs$ ./a.out
```
gettimeofday example

fawcett:330 childds$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took \%f\n", seconds);
}
Outline

• Announcements
• Special Topics
• Review
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
Inline function

• inlined functions:
  – hint to a compiler that can improve performance
  – basic idea: don’t actually make this be a separate function that is called
    • Instead, just pull the code out of it and place it inside the current function
  – new keyword: inline

```c
inline int doubler(int X) {
    return 2*X;
}

int main() {
    int Y = 4;
    int Z = doubler(Y);
    //
}
```
Inlines can be automatically done within class definitions

• Even though you don’t declare this as inline, the compiler treats it as an inline

```cpp
class MyDoublerClass
{
    int doubler(int X) { return 2*X; }
};
```
You can do inlines within header files, but not regular functions.

Left: function is inlined in every .C that includes it
... no problem
Right: function is defined in every .C that includes it
... duplicate symbols
Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides
Motivation

```c
int Doubler(int X) { return 2*X; };
float Doubler(float X) { return 2*X; };

int main()
{
    int X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << " , 2*Y = " << Doubler(Y) << endl;
}
```

```
fawcett:330 childs$ g++ loggerDefines.c
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

```
fawcett:330 childs$ nm a.out
0000000100000d7a s stub helpers
00000001000010b0 D __NXArgc
00000001000010b8 D __NXArgv
000000010000ac7 t __GLOBAL__I__Z7Doubleri
000000010000a84 t __Z41__static_initialization_and_destruction_0ii
000000010000b26 T __Z7Doublerf
000000010000b18 T __Z7Doubleri
```
Motivation

```c
// DOUBLER_MACRO(int)
DOUBLER_MACRO(float)

int main()
{
    int X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << endl;
}
```

```c
// DOUBLER_MACRO(int)
DOUBLER_MACRO(float)

int main()
{
    int X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << endl;
}
```
First Template

```cpp
#include <iostream>

using std::cout;
using std::endl;

template <class T> T Doubler(T X) { return 2*X; };

int main()
{
    int X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << "", 2*Y = " << Doubler(Y) << endl;
}
```
Will now do an example to compare templates and virtual functions

• Will take some buildup...
class Money
{
    public:
        Money(int d, int c) { dollars = d; cents = c; }
        bool operator<(const Money &m);

    private:
        int dollars;
        int cents;
};

bool Money::operator<(const Money &m)
{
    if (dollars < m.dollars)
        return true;
    if (dollars == m.dollars)
        return (cents < m.cents);

    return false;
}

int main()
{
    Money m(6, 85);
    Money m2(6, 25);
    bool lt = m < m2;
    cerr << "LT = " << lt << endl;
    lt = m2 < m;
    cerr << "LT = " << lt << endl;
}

C02LN00GFD58:330 hank$ g++ money.C
C02LN00GFD58:330 hank$ ./a.out
LT = 0
LT = 1
class LicensePlate
{
    public:
        LicensePlate(char c1, char c2, char c3,
                      int i1, int i2, int i3)
        {
            letters[0] = c1;
            letters[1] = c2;
            letters[2] = c3;
            numbers[0] = i1;
            numbers[1] = i2;
            numbers[2] = i3;
        }

        bool operator<(const LicensePlate &); 

    private:
        char    letters[3];
        int     numbers[3];
};

bool LicensePlate::operator<(const LicensePlate &rhs) 
{ 
    for (int i = 0; i < 3; i++) 
    { 
        if (letters[i] < rhs.letters[i]) 
            return true;
        if (letters[i] > rhs.letters[i]) 
            return false;
    }
    for (int i = 0; i < 3; i++) 
    { 
        if (numbers[i] < rhs.numbers[i]) 
            return true;
        if (numbers[i] > rhs.numbers[i]) 
            return false;
    }
    // equal
    return false;
}

int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    bool lt = lp1 < lp2; 
    cerr << "LT = " << lt << endl;
    lt = lp2 < lp1;
    cerr << "LT = " << lt << endl;
}
### Sorting With Templates

```cpp
template <class T>
void Sort(T **X, int nX)
{
    for (int i = 0; i < nX; i++)
        for (int j = i+1; j < nX; j++)
            if (*X[j] < *X[i])
                {
                    T *tmp = X[j];
                    X[j] = X[i];
                    X[i] = tmp;
                }
}
```

```cpp
int main()
{
    Money m1(6, 85);
    Money m2(6, 25);
    Money m3(4, 25);
    Money m4(5, 25);
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);
    Money *money_list[4] = { &m1, &m2, &m3, &m4 };
    LicensePlate *lp_list[4] = { &lp1, &lp2, &lp3, &lp4 };
    Sort(money_list, 4);
    Sort(lp_list, 4);
    for (int i = 0; i < 4; i++)
        cout << i << " : ": $" << money_list[i]->dollars << "." << money_list[i]->cents << endl;
    for (int i = 0; i < 4; i++)
        {
            cout << i << " : ": ";
            PrintLicensePlate(lp_list[i]);
            cout << endl;
        }
}
```
Doing the same with inheritance

```cpp
class Sortable
{
    public:
        virtual bool operator<(const Sortable *) = 0;
};

class LicensePlate : public Sortable
{
    public:
        LicensePlate(char c1, char c2, char c3,
                      int i1, int i2, int i3)
        {
            letters[0] = c1;
            letters[1] = c2;
            letters[2] = c3;
            numbers[0] = i1;
            numbers[1] = i2;
            numbers[2] = i3;
        }
        bool operator<(const Sortable *);
    public:
        char    letters[3];
        int     numbers[3];
};

void Sort(Sortable **X, int nX)
{
    for (int i = 0 ; i < nX ; i++)
        for (int j = i+1 ; j < nX ; j++)
            if (*X[j] < X[i])
                { 
                    Sortable *tmp = X[j];
                    X[j] = X[i];
                    X[i] = tmp;
                }
}

int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);

    Sortable *lp_list[4] = { &lp1, &lp2, &lp3, &lp4 };
    Sort(lp_list, 4);
    for (int i = 0 ; i < 4 ; i++)
        { 
            cout << i << " : ";
            PrintLicensePlate((LicensePlate *)lp_list[i]);
            cout << endl;
        }
}
```
Templates vs Virtual Functions

• Virtual functions:
  – Had to affect inheritance hierarchy
  – Overhead in function call (virtual function table)

• Templates:
  – Did not need to affect inheritance hierarchy, although function names had to coincide
  – No additional overhead (resolved at compile time)
Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides
Standard Template Library

• Standard Template Library: STL
• Many, many templated types
• Can ease your programming
• Can also hide significant performance issues
  — And you use C/C++ for performance
• My recommendation: use with caution for code that needs to be performant
Vector

#include <vector>

using std::vector;

int main()
{
    vector<int> intArray(2);
    intArray[0] = 0;
    intArray[1] = 1;
    intArray.push_back(1);
    intArray.push_back(2);
    intArray.push_back(3);
    intArray.push_back(5);
    cout << "Size is " << intArray.size() << endl;
    cout << "Last val of Fib is " << intArray[5] << endl;
}

C02LN00GFD58:330 hank$ g++ vector.C
C02LN00GFD58:330 hank$ ./a.out
Size is 6
Last val of Fib is 5
std::vector

• Always has the amount of memory you need
• Many STL algorithms work on it
• Memory allocation:
  – If you don’t have enough, double it
    • (can be a big overestimation)
• Overhead in access
  – Maybe not a big deal if data-intensive?
#include <map>
#include <string>

using std::map;
using std::string;

int main()
{
    map<string, int> ageLookup;
    ageLookup["Hank"] = 37;
    ageLookup["Charlotte"] = 11;
    ageLookup["William"] = 9;

    cout << "Hank's age is " << ageLookup["Hank"] << endl;
    cout << "Carissa's age is " << ageLookup["Carissa"] << endl;
}

C02LN00GFD58:330 hank$ g++ map.C
C02LN00GFD58:330 hank$ ./a.out
Hank's age is 37
Carissa's age is 0
Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides
(not a template thing): String

- C++ string class is very useful
- Great implementation of a class that encapsulates a string

```cpp
#include <string>

using std::string;

int main()
{
    string str = "Hello";
    str += " world";
    cout << str << endl;
}
```
# String methods

## Iterators:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>begin</code></td>
<td>Return iterator to beginning (public member function)</td>
</tr>
<tr>
<td><code>end</code></td>
<td>Return iterator to end (public member function)</td>
</tr>
<tr>
<td><code>rbegin</code></td>
<td>Return reverse iterator to reverse beginning (public member function)</td>
</tr>
<tr>
<td><code>rend</code></td>
<td>Return reverse iterator to reverse end (public member function)</td>
</tr>
<tr>
<td><code>cbegin</code></td>
<td>Return const_iterator to beginning (public member function)</td>
</tr>
<tr>
<td><code>cend</code></td>
<td>Return const_iterator to end (public member function)</td>
</tr>
<tr>
<td><code>crbegin</code></td>
<td>Return const_reverse_iterator to reverse beginning (public member function)</td>
</tr>
<tr>
<td><code>crend</code></td>
<td>Return const_reverse_iterator to reverse end (public member function)</td>
</tr>
</tbody>
</table>

## Capacity:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size</code></td>
<td>Return length of string (public member function)</td>
</tr>
<tr>
<td><code>length</code></td>
<td>Return length of string (public member function)</td>
</tr>
<tr>
<td><code>max_size</code></td>
<td>Return maximum size of string (public member function)</td>
</tr>
<tr>
<td><code>resize</code></td>
<td>Resize string (public member function)</td>
</tr>
<tr>
<td><code>capacity</code></td>
<td>Return size of allocated storage (public member function)</td>
</tr>
<tr>
<td><code>reserve</code></td>
<td>Request a change in capacity (public member function)</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>Clear string (public member function)</td>
</tr>
<tr>
<td><code>empty</code></td>
<td>Test if string is empty (public member function)</td>
</tr>
<tr>
<td><code>shrink_to_fit</code></td>
<td>Shrink to fit (public member function)</td>
</tr>
</tbody>
</table>
# String methods

**Element access:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator[]</code></td>
<td>Get character of string (public member function)</td>
</tr>
<tr>
<td><code>at</code></td>
<td>Get character in string (public member function)</td>
</tr>
<tr>
<td><code>back</code></td>
<td>Access last character (public member function)</td>
</tr>
<tr>
<td><code>front</code></td>
<td>Access first character (public member function)</td>
</tr>
</tbody>
</table>

**Modifiers:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator+=</code></td>
<td>Append to string (public member function)</td>
</tr>
<tr>
<td><code>append</code></td>
<td>Append to string (public member function)</td>
</tr>
<tr>
<td><code>push_back</code></td>
<td>Append character to string (public member function)</td>
</tr>
<tr>
<td><code>assign</code></td>
<td>Assign content to string (public member function)</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>Insert into string (public member function)</td>
</tr>
<tr>
<td><code>erase</code></td>
<td>Erase characters from string (public member function)</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>Replace portion of string (public member function)</td>
</tr>
<tr>
<td><code>swap</code></td>
<td>Swap string values (public member function)</td>
</tr>
<tr>
<td><code>pop_back</code></td>
<td>Delete last character (public member function)</td>
</tr>
</tbody>
</table>

**String operations:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c_str</code></td>
<td>Get C string equivalent (public member function)</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Get string data (public member function)</td>
</tr>
<tr>
<td><code>get_allocator</code></td>
<td>Get allocator (public member function)</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Copy sequence of characters from string (public member function)</td>
</tr>
<tr>
<td><code>find</code></td>
<td>Find content in string (public member function)</td>
</tr>
<tr>
<td><code>rfind</code></td>
<td>Find last occurrence of content in string (public member function)</td>
</tr>
<tr>
<td><code>find_first_of</code></td>
<td>Find character in string (public member function)</td>
</tr>
<tr>
<td><code>find_last_of</code></td>
<td>Find character in string from the end (public member function)</td>
</tr>
<tr>
<td><code>find_first_not_of</code></td>
<td>Find absence of character in string (public member function)</td>
</tr>
<tr>
<td><code>find_last_not_of</code></td>
<td>Find non-matching character in string from the end (public member function)</td>
</tr>
<tr>
<td><code>substr</code></td>
<td>Generate substring (public member function)</td>
</tr>
<tr>
<td><code>compare</code></td>
<td>Compare strings (public member function)</td>
</tr>
</tbody>
</table>
Outline

• Announcements
• Special Topics
• Review
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
Backgrounding

- “&”: tell shell to run a job in the background
  – Background means that the shell acts as normal, but the command you invoke is running at the same time.

- “sleep 60” vs “sleep 60 &”

When would backgrounding be useful?
Suspending Jobs

• You can suspend a job that is running
  Press “Ctrl-Z”
• The OS will then stop job from running and not schedule it to run.
• You can then:
  – make the job run in the background.
    • Type “bg”
  – make the job run in the foreground.
    • Type “fg”
      – like you never suspended it at all!!
printf

• Print to stdout
  – printf(“hello world\n”);
  – printf(“Integers are like this %d\n”, 6);
  – printf(“Two floats: %f, %f”, 3.5, 7.0);

Have you ever wondered how printf takes a variable number of arguments?
Variable-Length Argument Lists

```c
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

int SumIntList(int X, ...)
{
    va_list ap;  /* points to each unnamed arg in turn */
    int sum = 0;
    int ival;
    int i;
    va_start(ap, X);  /* make ap point to 1st unnamed arg */
    for (i = 0; i < X; i++)
    {
        ival = va_arg(ap, int);
        sum += ival;
    }
    va_end(ap);
    return sum;
}

int main()
{
    printf("List sum = %d\n", SumIntList(3, 13, 17, 22));
    printf("List sum = %d\n", SumIntList(5, 1, 2, 3, 4, 5));
}
```

Adapted from Kernigan & Ritchie C book