CSE 333
Section 6

Inheritance & VTable, HW3 Overview, maybe (Templates and STL)
Logistics

Friday (tomorrow)
   Exercise 14 @ 10:30 am

Monday
   Exercise 13 @ 10:30 am

Wednesday:
   NO EXERCISE DUE!!!

Thursday:
   HW3 @ 11:59 pm
Section Plan

- Inheritance & VTables
- HW 3 Overview
- STL & Templates (if we have time)
Inheritance

- **Derived** class inherits from the **base** class
  - In 333, we always use *public* inheritance
  - Inherits all *non-private* member **variables**
  - Inherits all *non-private* member **functions**
    except for ctor, cctor, dtor, op=

- Access specifiers revisited:
  - **Private** members cannot be accessed by derived classes
  - **Protected** members are available to base & derived
Static vs. Dynamic Dispatch

How to resolve invoking a method via a polymorphic pointer:

1. Static dispatch
   ○ Default behavior in C++
   ○ More to come in Friday’s lecture!

2. Dynamic dispatch
   ○ Which implementation is determined at runtime via lookup
   ○ Compiler generates code that accesses function pointers added to the class
**Dispatch Decision Tree**

PromisedT *ptr = new ActualT();
ptr->Fcn(); // which version is called?

1. **Is Fcn() defined in PromisedT?**
   - **Yes**: Is PromisedT::Fcn() static or dynamic dispatch?
     - **Static**: Static dispatch of PromisedT::Fcn()
     - **Dynamic**: Dynamic dispatch of most-derived version of Fcn() visible to ActualT
   - **No**: Compiler Error

   **How can we tell?**
Dispatch Keywords

- **virtual** – request dynamic dispatch
  - Is “sticky”: overridden virtual method in derived class is still virtual with or without the keyword

- **override** – ensures that the function is virtual and is overriding a virtual function from a base class (@override in Java)
  - Generates a compiler error if conditions are not met
  - Catches overloading vs. overriding bugs at compile time
Practice: static, dynamic, or error?

class Base {
    void Foo();   //
    void Bar();   //
    virtual void Baz();  //
};

class Derived : public Base {
    virtual void Foo();   //
    void Bar() override;  //
    void Baz();           //
};
Practice: static, dynamic, or error?

class Base {
    void Foo();       // static dispatch
    void Bar();       // static dispatch
    virtual void Baz(); // dynamic dispatch
};

class Derived : public Base {
    virtual void Foo();   //
    void Bar() override;  //
    void Baz();           //
};
Practice: static, dynamic, or error?

class Base {
    void Foo();    // static dispatch
    void Bar();    // static dispatch
    virtual void Baz();   // dynamic dispatch
};

class Derived : public Base {
    virtual void Foo();    // now dynamic (for more derived)
    void Bar() override; //
    void Baz();   //
};
Practice: static, dynamic, or error?

class Base {
    void Foo(); // static dispatch
    void Bar(); // static dispatch
    virtual void Baz(); // dynamic dispatch
};

class Derived : public Base {
    virtual void Foo(); // now dynamic (for more derived)
    //void Bar() override; // compiler error
    void Bar(); // static dispatch
    void Baz(); //
};
class Base {
    void Foo(); // static dispatch
    void Bar(); // static dispatch
    virtual void Baz(); // dynamic dispatch
};

class Derived : public Base {
    virtual void Foo(); // now dynamic (for more derived)
    // void Bar() override; // compiler error
    void Bar(); // static dispatch
    void Baz(); // still dynamic (sticky!)
};
Vtable (Virtual Function Table) & Vptr (Vtable pointer)

- **vtable**: An array of function pointers defined for each class that has at least one virtual method to enable dynamic dispatch
  - One per class
- **vptr**: Each class object instance has a pointer to that vtable
  - One per object instance
Vtable Diagrams

class IntQueue {
public:
    virtual void Push(int x);
    virtual int Pop();
private:
    vector<int> data_
};

IntQueue q;

Object Instance

vptr

data_

vtable

IntQueue::Push

IntQueue::Pop

Implementation
Vtable Diagrams

Base *b1_ptr = new Base;
Base *b2_ptr = new Derived;
Derived *d_ptr = new Derived;

class Base {
    void Foo();
    void Bar();
    virtual void Baz();
};

class Derived : public Base {
    virtual void Foo();
    void Baz();
};
PromisedT *ptr = new ActualT();
ptr->Fcn(); // which version is called?

- Is Fcn() defined in PromisedT?
  - Yes
    - Is PromisedT::Fcn() marked virtual in PromisedT or in classes it derives from?
      - Yes
        - Dynamic dispatch of most-derived version of Fcn() visible to ActualT
      - No
        - Static dispatch of PromisedT::Fcn()
  - No
    - Compiler Error
Exercise 1!
Exercise 1
Exercise 1 Solution (pointers)
# Exercise 1 Solution (output)

```cpp
#include <iostream>
using namespace std;

class A {
public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
public:
  void f1() { f2(); cout << "C::f1" << endl; }
};

A* aa = new A();
aa->f1();

A::f2
A::f1
```

```
A* aa = new A();

A::f2
A::f1
```
#include <iostream>
using namespace std;

class A {
    public:
        virtual void f1() { f2(); cout << "A::f1" << endl; }
        void f2() { cout << "A::f2" << endl; }
};

class B: public A {
    public:
        virtual void f3() { f1(); cout << "B::f3" << endl; }
        virtual void f2() { cout << "B::f2" << endl; }
    
};

class C: public B {
    public:
        void f1() { f2(); cout << "C::f1" << endl; }
    
};

B* bb = new B();
bb->f1();

A::f2
A::f1
#include <iostream>
using namespace std;

class A {
public:
    virtual void f1() { f2(); cout << "A::f1" << endl; }
    void f2() { cout << "A::f2" << endl; }
};

class B: public A {
public:
    virtual void f3() { f1(); cout << "B::f3" << endl; }
    virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
public:
    void f1() { f2(); cout << "C::f1" << endl; }
};

B* bb = new B();
A* ab = bb;
bb->f2();
cout << "----" << endl;
ab->f2();

B::f2
----
A::f2
#include <iostream>
using namespace std;

class A {
public:
    virtual void f1() { f2(); cout << "A::f1" << endl; }
    void f2() { cout << "A::f2" << endl; }
};

class B: public A {
public:
    virtual void f3() { f1(); cout << "B::f3" << endl; }
    virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
public:
    void f1() { f2(); cout << "C::f1" << endl; }
};

B* bb = new B();
bb->f3();
Exercise 1 Solution (output)

```cpp
#include <iostream>
using namespace std;

class A {
    public:
    virtual void f1() { f2(); cout << "A::f1" << endl; }
    void f2() { cout << "A::f2" << endl; }
};

class B: public A {
    public:
    virtual void f3() { f1(); cout << "B::f3" << endl; }
    virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
    public:
    void f1() { f2(); cout << "C::f1" << endl; }
};

A* ac = new C();
ac->f1();
```

A::f1
B::f2
C::f1
Any Questions on Inheritance & VTable?
HW 3 Overview!
Crawling a file tree in HW2 takes a long time.

To save time, write the completed DocTable and MemIndex to a File!
Index File Components

- **Header (metadata)**
- **DocTable**
- **MemIndex**
Index File Header

- magic_number: 0xCAFEF00D
- checksum: mathematical signature
- doctable_size: in bytes
- index_size: in bytes
1. Find a hex editor/viewer of your choice
   - `xxd <indexfile>`
   - `hexdump -vC <indexfile>`

Pipe the output into a file or less to view.

The header:

**Magic word**  **Checksum**  **Doctable size**  **Index size**

```
000000: cafe_f00d_1c42_4620_0000_205b_0000_075d ....BF ..[...]
000010: 0000_0400_0000_0000_0000_2014_0000_0001 .................
000020: 0000_2014_0000_0001_0000_2031_0000_0001 ..........1...
000030: 0000_204e_0000_0000_0000_206b_0000_0000 ..N........k...
000040: 0000_206b_0000_0000_0000_206b_0000_0000 ..k........k...
000050: 0000_206b_0000_0000_0000_206b_0000_0000 ..k........k..
```

man `xxd`
man `hexdump`
Byte Ordering and Endianness

- Network (Disk) Byte Order (Big Endian)
  - The most significant byte is stored in the highest address
- Host byte order
  - Might be big or little endian, depending on the hardware
- To convert between orderings, we can use
  - `uint32_t htonl (uint32_t hostlong);`  // host to network
  - `uint32_t ntohl (uint32_t hostlong);`  // network to host
- Pro-tip:
The structs in HW3 have toDiskFormat() and toHostFormat() functions that will convert endianness for you.
Hex View

- emacs “M-x hexl-mode”

- vim “:%!xxd”
DocTable & MemIndex

• At their core, both DocTable & MemIndex are HashTables.
• Lets first look at how we write a HashTable.
HashTable

• HashTable can have varying amount of buckets, so start with num_buckets.

• Buckets can be of varying lengths. To know the offset, we store some bucket records.
Buckets

• A bucket is a list that contains elements in the table. Offset to a bucket is found in a bucket record.
• Elements can be of various sizes, so we need to store element positions to know where each element is.
DocTable & MemIndex

• At their core, both DocTable & MemIndex are HashTables.

• The difference between DocTable and MemIndex is entirely what type of element is stored in them.
### DocTable (Hex)

The header

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num buckets</td>
<td>4 bytes</td>
<td>Number of buckets</td>
</tr>
<tr>
<td>Chain len</td>
<td>4 bytes</td>
<td>Total length of chains</td>
</tr>
<tr>
<td>Bucket offset</td>
<td>4 bytes</td>
<td>Offset of buckets</td>
</tr>
</tbody>
</table>

#### Example Data

```
0000000: cafe f00d 1c42 4620 0000 205b 0000 075d ...BF .. [...] 0000010: 0000 0400 0000 0000 0000 2014 0000 0001 ............. 1...
0000020: 0000 204e 0000 0000 0000 206b 0000 0000 N... k...
0000030: 0000 206b 0000 0000 0000 206b 0000 0000 k... k...
0000040: 0000 206b 0000 0000 0000 206b 0000 0000 k... k...
0000050: 0000 206b 0000 0000 0000 206b 0000 0000 k... k...
0000060: 0000 206b 0000 0000 0000 206b 0000 0000 k... k...
0000070: 0000 206b 0000 0000 0000 206b 0000 0000 k... k...
```

---

Note: The table and diagram illustrate the structure of the DocTable, showing how the header fields relate to the actual data storage. The header contains information about the number of buckets, chain length, and bucket offsets, which are crucial for processing and retrieving document information.
The buckets:

\[(\text{Element offset})^n \ (\text{DocID} \quad \text{Filename len} \quad \text{Filename})^n\]
**docID table**

- **word length**: 2 bytes
- **docID table length**: 4 bytes
- **word**: word length bytes
- **docID table**: docID table length bytes

- **bucket_rec**
  - **num_buckets**: 4 bytes
  - bucket_rec 0: 8 bytes
  - bucket_rec 1: 8 bytes
  - bucket_rec 2: 8 bytes
- **element 0 position**: 4 bytes
  - element
  - chain_len-1
  - position
- **element**: position
  - num_positions-1
  - position 0
  - position 1
  - position 2
- **docID**: 8 bytes
  - num_positions
  - position 0
  - position 1
  - position 2
The Full Picture
HW Tips

• When Writing, you should (almost) always:
  1. .toDiskFormat()
  2. fseek()
  3. fwrite()

• When Reading, you should (almost) always:
  1. fseek()
  2. fread()
  3. .toHostFormat()

• The most common bugs in the hw involve forgetting to change byte ordering, or forgetting to fseek().
Exercise 3!
# Exercise 3 Solution

```cpp
#include <iostream>
using namespace std;

class List {
public:
    // construct empty list
    List() : head_(nullptr) { }

    // add new node with value n to the front of the list
    virtual void add(int n) {
        Link *p = new Link(n, head_);
        head_ = p;
    }

    ...

};
```
#include <iostream>
using namespace std;

template <typename T>
class List {
    public:
        // construct empty list
    List() : head_(nullptr) {} 

        // add new node with value n to the front of the list
    virtual void add(int n) {
            Link *p = new Link(n, head_);
            head_ = p;
    }

...
Exercise 3 Solution

```cpp
#include <iostream>
using namespace std;

template <typename T>
class List {
    public:
        // construct empty list
        List() : head_(nullptr) { }

        // add new node with value n to the front of the list
        virtual void add(int T n) {
            Link *p = new Link(n, head_);
            head_ = p;
        }

    ...
```
private:
    struct Link {  // nodes for the linked list
        int val;
        Link * next;
        Link(int n, Link* nxt): val(n), next(nxt) { }
    };
    // List instance variable
    Link * head_;  // head of list or nullptr if list is empty
};  // end of List class

int main() {
    List nums;
    nums.add(1);
    nums.add(2);
    return EXIT_SUCCESS;
}
Exercise 3 Solution

...  
private:
  struct Link {   // nodes for the linked list
    int T val;
    Link * next;
    Link(int T n, Link* nxt): val(n), next(nxt) { }
  };
  // List instance variable
  Link * head_;  // head of list or nullptr if list is empty
};  // end of List class

int main() {
  List nums;
  nums.add(1);
  nums.add(2);
  return EXIT_SUCCESS;
}
Exercise 3 Solution

... 
private:
struct Link {   // nodes for the linked list
    int T val;
    Link * next;
    Link(int T n, Link* nxt): val(n), next(nxt) { } 
};
// List instance variable
Link * head_;   // head of list or nullptr if list is empty
};  // end of List class

int main() {
    List<int> nums;
    nums.add(1);
    nums.add(2);
    return EXIT_SUCCESS;
}
C++ standard lib is built around templates

- Containers
  - Store data
  - Define iterators to go over that data

- Iterators
  - Different flavors (random access, bidirectional, etc)
  - Common interface to containers

- Algorithms
  - Use the common interface of iterators to do things
    - Different algorithms require different ‘complexities’ of iterators
Common STL Data Structures

- **map<Key, Value, Order=std::Less<Key>>**
  - Store key -> value pairs where we can use a key to get the value (TreeMap)

- **set<Item, Order=std::Less<Item>>**
  - An unindexed collection of items
  - When you care most about “do I have this”

- **vector<Item>**
  - Resizable array (ArrayList, in Java)

- **unordered_map<Key, Value, Hash=std::Hash<Key>>**
  - HashMap → use hash of key to order. Usually faster than map

- Assorted others (queue, linkedlist, etc.)
Much like in Java, some structures require ordering elements
○ E.g. set is implemented as a binary tree

Want to let users store custom types.
○ Java uses Comparable, C++ uses operator< (in std::less)

However, maybe you want to use a different ordering
○ Ordering is templated function so you can substitute
○ E.g. set<int, std::greater<int>> or set<int, myIntCompare>

Now what's that ‘std::less’? // Out of scope

```cpp
std::less<T>(const T& lhs, const T& rhs) {
  return lhs < rhs;
}
```
Exercise 4!
Exercise 4!

Exercises:

2) Standard Template Library
Complete the function ChangeWords below. This function has as inputs a vector of strings, and a map of <string, string> key-value pairs. The function should return a new vector<string> value (not a pointer) that is a copy of the original vector except that every string in the original vector that is found as a key in the map should be replaced by the corresponding value from that key-value pair.

Example: if vector words is {"the", "secret", "number", "is", "xlii"} and map subs is {{"secret", "magic"}, {"xlii", "42"}}, then ChangeWords(words, subs) should return a new vector {"the", "magic", "number", "is", "42"}.

Hint: Remember that if m is a map, then referencing m[k] will insert a new key-value pair into the map if k is not already a key in the map. You need to be sure your code doesn’t alter the map by adding any new key-value pairs. (Technical nit: subs is not a const parameter because you might want to use its operator[] in your solution, and [] is not a const function. It’s fine to use [] as long as you don’t actually change the contents of the map subs.)
Write your code below. Assume that all necessary headers have already been written for you.

```cpp
using namespace std;

vector<string> ChangeWords(const vector<string> &words,
                           map<string,string> &subs) {

}
```
using namespace std;
vector<string> ChangeWords(const vector<string> &words,
                          map<string,string> &subs) {
    vector<string> result;
    for (auto &word : words) {
        if (subs.find(word) != subs.end()) {
            result.push_back(subs[word]);
        } else {
            result.push_back(word);
        }
    }
    return result;
}
Exercise 5!
Here is a little program that has a small class `Thing` and main function (assume that necessary `#include` s and `using namespace std;` are included).

```cpp
class Thing {
public:
    Thing(int n): n_(n) { }
    int getThing() const { return n_; }
    void setThing(int n) { n_ = n; }
private:
    int n_{};
};

int main() {
    Thing t(17);
    vector<Thing> v;
    v.push_back(t);
}
```

This code compiled and worked as expected, but then we added the following two lines of code (plus the appropriate `#include <set>`):

```cpp
set<Thing> s;
s.insert(t);
```

The second line (s.insert(t)) failed to compile and produced dozens of spectacular compiler error messages, all of which looked more-or-less like this (edited to save space):

```
In file included from string:48:0, from bits/locale_classes.h:40, from bits/ios_base.h:41, from ios:42, from ostream:38, from /iostream:39, from thing.cc:3: bits/stl_function.h: In instantiation of 'bool std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp = Thing]':
  thing.cc:37:13: required from here
bits/stl_function.h:387:20: error: no match for 'operator<' (operand types are 'const Thing' and 'const Thing') { return __x < __y; }
```

What on earth is wrong? Somehow class `Thing` doesn't work with `set<Thing>` even though `insert` is the correct function to use here. (a) What is the most likely reason, and (b) what would be needed to fix the problem? (Be brief but precise – you don’t need to write code in your answer, but you can if that helps make your explanation clear.)
Here is a little program that has a small class `Thing` and main function (assume that necessary `#include` s and `using namespace std;` are included).

```cpp
class Thing {
public:
    Thing(int n): n_(n) { }
    int getThing() const { return n_; }
    void setThing(int n) { n_ = n; }
private:
    int n_;}

int main() {
    Thing t(17);
    vector<Thing> v;
    v.push_back(t);
}
```

The second line (`s.insert(t)`) failed to compile and produced dozens of spectacular compiler error messages, all of which looked more-or-less like this (edited to save space):

```
In file included from string:48:0, from bits/locale_classes.h:40, from bits/ios_base.h:41, from ios:42, from ostream:38, from /iostream:39, from thing.cc:3: bits/stl_function.h: In instantiation of 'bool std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp = Thing]': <<many similar lines omitted>> thing.cc:37:13: required from here bits/stl_function.h: 387:20: error: no match for 'operator<' (operand types are 'const Thing' and 'const Thing') { return __x < __y; }
```

What on earth is wrong? Somehow class `Thing` doesn’t work with `set<Thing>` even though `insert` is the correct function to use here. (a) What is the most likely reason, and (b) what would be needed to fix the problem? (Be brief but precise – you don’t need to write code in your answer, but you can if that helps make your explanation clear.)

STL has to compare them using `operator<`. Add an appropriate `operator<` as either a member function in `Thing`, or as a free-standing function that compares two `Thing&` parameters.
Hex View Exercise

• Split up into break out rooms.
• Take a look at https://courses.cs.washington.edu/courses/cse333/20sp/sections/sec06.idx
  • Log into attu, use wget to download the file, then look into it.
• Try to figure out:
  How many documents are in this index?
  Which words are in each document?
Hex View Exercise

• Split up into break out rooms.
• Take a look at [https://courses.cs.washington.edu/courses/cse333/20sp/sections/sec06.idx](https://courses.cs.washington.edu/courses/cse333/20sp/sections/sec06.idx)
  • Log into attu, use wget to download the file, then look into it.
• Try to figure out:
  How many documents are in this index?
  Which words are in each document?

• Answer: This index file was built off of test_tree/tiny