CSE333 SECTION 7
Midterm Debrief
Hex View

1. Find a hex editor.
2. Learn ‘goto offset’ command.
3. See HW3 pictures.

The header:

Magic word    Checksum    Doctable size    Index size
Hex View

The doctable
Hex View

The index
Hex View

The docID table
class IntPair {

public:
    IntPair(const int first, const int second)
    : first_(first), second_(second) { }
    int first() const { return first_; }
    int second() const { return second_; }

private:
    int first_;  // int member variable
    int second_; // int member variable
};

class DoublePair {

public:
    DoublePair(const double first, const double second)
    : first_(first), second_(second) { }
    double first() const { return first_; }
    double second() const { return second_; }

private:
    double first_;  // double member variable
    double second_; // double member variable
};
class FooPair {
public:
    FooPair(const Foo& first, const Foo& second)
        : first_(first), second_(second) { }
    Foo first() const { return first_; }
    Foo second() const { return second_; }
private:
    Foo first_
    Foo second_;
};

• This is really repetitive!
Templates

template <typename T>
class Pair {

public:
    Pair(const T& first, const T& second) :
        first_(first), second_(second) { }
    T first() const { return first_; }
    T second() const { return second_; }

private:
    T first_;  
    T second_;  
};
Templates

- Functions can be templated too
- Each “type” of template class/function generates distinct code
  - `Pair<int>` and `Pair<Foo>` are two distinct classes with code located in two distinct regions of the binary
- Templates are generated at compile time
  - Compiler needs to know how each template will be used
  - Full definitions of templated code must be included in translation unit
Standard Template Library

- C++ comes with a rich set of templated collections
  - cplusplus.com
  - cppreference.com
- All collections pass by value (copy), *not* by reference
- Automatic resizing of a collection can trigger multiple copy operations
  - One way to make this more efficient: move semantics
    - Outside the scope of this class, but ask Sunjay about it any time
  - Another way to avoid this: pass in pointers to data
    - Memory management gets messy
      - Use smart pointers!
Smart Pointers

- Encapsulate memory management through ctors/dtors
- Wraps a “normal” pointer
- Automatically calls delete when lifetime is over
- Three types:
  - unique_ptr ensures only one pointer to underlying data
    - Does this by disallowing copy construction/assignment
    - You can still use it in STL containers though (move semantics!)
  - shared_ptr keeps a reference count
    - Only deletes wrapped pointer when reference count hits zero
  - weak_ptr does not contribute to the reference count
    - Think circular linked lists, you’d want a weak_ptr at the end of the list to ensure the reference count to the front can go down to 0.
    - Very rarely used otherwise
Smart Pointer Examples

- unique_ptr.cc
- shared_ptr_leaky.cc
- shared_ptr_good.cc
Inheritance Constructors/Destructors

- The derived class:
  - Does not inherit any constructors.
  - MUST call their base class constructor.
    - Omission == calling the default constructor.

- Constructors resolve from base to derived.

- Destructors should be virtual!
Inheritance Examples

- Example:
- destructex.cc
  - This code compiles with no warnings so it must be right?
Vtables

- Dynamic dispatch
- All virtual functions are stored in a “virtual function table”
  - Each class has its own vtable
- Each instance contains an extra “field”
  - Pointer to class vtable
  - Only exists if class has virtual methods
- Derived classes have functions in same order as base class
  - Overriding functions replace base functions at same indices
Vtable Example

class Base {
    virtual void other_fn();
    virtual void overridden();
};

class Derived {
    void overridden() override;
};

class Base vtable
<table>
<thead>
<tr>
<th>Base::other_fn()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base::overridden()</td>
</tr>
</tbody>
</table>

class Derived vtable
<table>
<thead>
<tr>
<th>Base::other_fn()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived::overridden()</td>
</tr>
</tbody>
</table>
Vtable Example

- Example:
- vtable.cc
  - Poke around this code with objdump or gdb!