CSE 333
Section 7
HW3 Overview, Casting
Logistics

Friday, Feb 26:
HW3 @ 11 pm
Section Plan

- Casting
- HW 3 Overview
Casting in C++
Casting in C

- Types are enforced unless converted
- **Casting** is a conversion between data types
- Can cast with anything in C!

**Implicit Casting**

```c
double a = 10.5;
int b = a;
```

**Explicit Casting**

```c
double a = 10.5;
int b = (int) a;
```
Casting in C++

Four different casts that are more explicit and help prevent unintended errors:

1. `static_cast<to_type>(expression)`
2. `dynamic_cast<to_type>(expression)`
3. `const_cast<to_type>(expression)`
4. `reinterpret_cast<to_type>(expression)`

When programming in C++, you should use these casts!
Static Cast

\texttt{static\_cast<to\_type>}(expression)

Used to:

1) Convert pointers of related types
   
   \texttt{Base* b = static\_cast<Base*>(new Derived);}
   
   - compiler error if types aren't related

2) Non-pointer conversion
   
   \texttt{int qt = static\_cast<int>(3.14);}
Static Cast

\texttt{static\_cast<to\_type>>(expression)}

[[!]] Be careful when casting down:

\begin{verbatim}
Derived* d = \texttt{static\_cast<Derived\*>(new Base)};
d->y = 5;
\end{verbatim}

- compiler will let you do this
- dangerous if you want to do things defined in Derived, but not in Base!
Dynamic Cast

dynamic_cast<to_type>(expression)

Used to:
1) Convert pointers of related types
   Base* b = dynamic_cast<Base*>(new Derived);
   - compiler error if types aren't related
   - at runtime, returns nullptr if it is actually an
     unsafe downwards cast:
   Derived* d = dynamic_cast<Derived*>(new Base);
Const Cast

\texttt{const\_cast<to\_type>}(\text{expression})

Used to:
1) Add or remove const-ness
   \begin{verbatim}
   const int x = 5;
   const int *ro_ptr = &x
   int *ptr = const_cast<int*>(ro_ptr);
   \end{verbatim}
Reinterpret Cast

`reinterpret_cast<to_type>(expression)`

Used to:

1) Cast between incompatible types

```c
int* ptr = 0xDEADBEEF;
int64_t x = reinterpret_cast<int64_t>(ptr);
```

- types must be of same size
- does not do float-integer conversions
Exercise 1
class Base {
    public:
    int x;
};

class Derived : public Base {
    public:
    int y;
};

int64_t x = 0x7fffffffffffffff;
char* str = reinterpret_cast<char*>(x);

void foo(Base *b) {
    Derived *d = dynamic_cast<Derived*>(b);
    // additional code omitted
}

Derived *d = new Derived;
Base *b = static_cast<Base*>(d);

double x = 64.382;
int64_t y = static_cast<int64_t>(x);
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

- **DocTable**
- **MemIndex**

Header (metadata)

---

Index file

- `magic_number`: 4 bytes
- `checksum`: 4 bytes
- `doctable_size`: 4 bytes
- `index_size`: 4 bytes

- `doctable`:
  - `doctable_size`: bytes

- `index`:
  - `index_size`: bytes
Index File Header

- magic_number: 0xCAFEF00D
- checksum: mathematical signature
- doctable_size: in bytes
- index_size: in bytes
Index File Header - HEX

1. Find a hex editor/viewer of your choice
   - xxd <indexfile>
   - hexdump –vC <indexfile>

The header:

<table>
<thead>
<tr>
<th>Magic word</th>
<th>Checksum</th>
<th>Doctable size</th>
<th>Index size</th>
</tr>
</thead>
<tbody>
<tr>
<td>cafe f00d</td>
<td>1c42 4620</td>
<td>0000 205b 0000 075d</td>
<td>....BF ... [ ...]</td>
</tr>
<tr>
<td>0000 0400</td>
<td>0000 0000</td>
<td>0000 2014 0000 0001</td>
<td>........ 1 ...</td>
</tr>
<tr>
<td>0000 204e</td>
<td>0000 0000</td>
<td>0000 206b 0000 0000</td>
<td>... N ... k ...</td>
</tr>
<tr>
<td>0000 206b</td>
<td>0000 0000</td>
<td>0000 206b 0000 0000</td>
<td>... k ... k ...</td>
</tr>
</tbody>
</table>

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.
• Let's 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

Num buckets  ( Chain len  Bucket offset )*
The buckets:

\(( (\text{Element offset})^n (\text{DocID} \quad \text{Filename len} \quad \text{Filename } )^n )^n \)*
docID table
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().
Hex View Exercise

• Split up into break out rooms.
• Take a look at https://courses.cs.washington.edu/courses/cse333/21wi/sections/sec07.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/cse333/20au/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