Agenda

- Constructors and Destructors

- PA2 questions / discussion
  - Bits and binary operations
  - Binary IO / Files
  - ios::binary
  - Gprof
  - subversion/cvs (brief)
Constructors

- **What are constructors used for?**
  - Custom initialization

- **When a variable comes into scope**
  - First space is allocated for it
  - The variable is initialized
    - Constructor called!
  - Can a constructor have parameters?
  - Example

- **What about new statements?**
Destructors

- What are destructors used for?
  - Custom deallocation
  - Can destructors have parameters?

- When a variable goes out of scope
  - Destructor called!
  - Space is deallocated
  - Local variables are destroyed in the reverse order of creation

- Useful for deallocated all virtual memory used in the class
Example

class Dog {
public:
    char* name;
    int age;
    Dog(char* name="Rex", int age=0) {
        this->name=new char[strlen(name)+1];
        strcpy(this->name,name);
        this->age=age;
        cout << name << " is born" << endl;
    }
    ~Dog() {
        cout << name << " is gone" << endl;
        delete[] name;
    }
}

+1 for the terminating \0

releases dynamically allocated memory back to the heap.
Example (contd)

```c++
main() {
    Dog dogA("Fido", 3), dogB;
    cout << "Here boy!" << endl;
    if (true) { Dog dogC = "Spot"; }  // destructor call
    cout << "Good girl!" << endl; }
```

Output:  Fido is born
         Rex is born
         Here boy!
         Spot is born
         Spot is gone
         Good girl!
         Rex is gone
         Fido is gone
Bits and Binary

• Why do we care?
  - A deep understanding of computers requires an understanding of what is happening at the bit level
  - Common interview questions

• Application examples
  - Device drivers, parts of operating systems
  - Heavily optimized code
  - Embedded devices
  - Low-level file manipulation (e.g. compression)
  - Storing arbitrary data formats efficiently
Data Representation

- **Standard variable types hold sets of bits**
  - `char` (1 byte): `10010001`
  - `int` (2-4 bytes): `1001000111001001`
  - `long` (4+ bytes): `100100011100100110010001`

- **Unsigned flag is often useful to prevent any of the bits being interpreted as a negation flag**

- **We often interpret chars as ASCII codes, but this is only a particular interpretation of the bits**

- **Assigning 00010000 to a char**
  - `char x = 16;` (decimal)
  - `char x = 020;` (octal)
  - `char x = 0x10;` (hex)
Bit Manipulations

- &  AND  \[1010 \& 1001 = ???\]
- |  OR  \[1010 \mid 1001 = ???\]
- ^  XOR  \[1010 ^ 1001 = ???\]
- ~  complement  \[\sim 1010 = ???\]
- << left shift  \[1010 << 2 = ???\]
- >> right shift  \[1010 >> 1 = ???\]
Bit Manipulations

- **Masking**
  - How to extract the middle 4 bits from 01011100?
  - How to clear the middle 4 bits from 01011100?

- **Packing**
  - How do we combine 0x04 and 0x03 into a single byte?
Bit Fields

- Bit fields are sometimes useful for storing bits in a compressed structure in memory.

- Not directly useful for IO.

- Bit shifts/operations can accomplish the same thing.

```c
struct DISK_REGISTER {
    unsigned ready:1;
    unsigned error_occurred:1;
    unsigned disk_spinning:1;
    ...
};
```
Files are streams of bits

010010010101010101010101010100110
000100101011111101001001010100101
010100111001001010100101010101010
100000000011110110010011110010100
Symbols are just chunks of bits

0100100101010101010101010101000110
000100101011111101001001010100101
010100110110010001010101010101010
10000000001111010100100011110010100
IO for binary files

- Open files using binary mode
  - Prevents interpretations of newlines, spaces, etc. from causing trouble

```cpp
ifstream myFile ("datain.bin", ios::in | ios::binary);
```
Reading and Writing Bits

- Can only read/write on byte boundaries

- Don’t use >> and << (insertion/extraction)
  - These are for *formatting*

- Use read/write functions for blocks of chars
  - No null character appended, no interpretation, just the raw bits
  - May not want to read everything at one time for very large files; use blocks (e.g. 1KB at a time)
Reading and Writing Bits

Read from a file:

```cpp
char buffer[100];
ifstream myFile ("datain.bin", ios::in | ios::binary);
myFile.read (buffer, 100);
// do stuff with the data
```

Write to a file:

```cpp
char buffer[100];
// fill the buffer with the data you want
ofstream myFile ("dataout.bin", ios::out | ios::binary);
myFile.write (buffer, 100);
```
**ios::binary**

- Used while opening files:
  ```cpp
  myInStream.open("fileName.txt", ios::in | ios::binary);
  ```

- Does it convert the data in file to binary?
  - Trick Question!
  - The files are already in binary! Duh!

- What `ios::binary` does is prevent interpretation of end-of-line characters, etc
Viewing binary files

- A text editor interprets files as ASCII, which won’t work well for generic binary files

  - `xxd` (hex + ASCII)
  - `xxd -b` (binary + ASCII)
  - `hexdump -C` (hex + ASCII)
xxd output

0006530: 0000 0016 0000 0a26 000b 4601 000d c605 ..........&..F.....
0006540: 0001 d002 0037 2400 0f84 d002 5e84 d002 .....7$.....^...
0006550: 0e84 0000 5d84 0000 1184 98fe 6084 98fe ....].......`...
0006560: 0f00 0037 2400 0f84 3804 5e84 3804 0e84 ...7$...8.^8...
0006570: 0000 5d84 0000 1184 0000 6084 0000 0016 ..].......`.....
0006580: 0000 0a26 010b 4601 000d c605 0001 a005 ...&..F........
0006590: 0037 2400 0f84 a005 5e84 a005 0e84 0000 .7$.....^.......
Profiling

- A profiler allows you to identify which pieces of code are taking the most time
  - Allows you to focus optimization effort where it is necessary

- To make your program 5% faster:
  - Optimizing a function that takes 1% of the total time?
  - Optimizing a function that takes 10% of the total time?
  - Optimizing a function that takes 50% of the total time?
gprof

- Compile with the -pg flag

- Run the program as usual
  - Will run somewhat slower

- You will have a “gmon.out” file in your directory
  - This is overwritten each time the program is run

- Run gprof to examine the data:
  
gprof options [executable-file [profile-data-files...]] [> outfile]
gprof: flat profile output

Flat profile:

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self seconds</th>
<th>cumulative time</th>
<th>calls</th>
<th>us/call</th>
<th>self total</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37.50</td>
<td>0.15</td>
<td>0.15</td>
<td>48000</td>
<td>3.12</td>
<td>3.12</td>
<td>Life::neighbor_count(int, int)</td>
</tr>
<tr>
<td>17.50</td>
<td>0.22</td>
<td>0.07</td>
<td></td>
<td></td>
<td></td>
<td>_IO_do_write</td>
</tr>
<tr>
<td>10.00</td>
<td>0.26</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td>__overflow</td>
</tr>
<tr>
<td>7.50</td>
<td>0.29</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td>_IO_file_overflow</td>
</tr>
<tr>
<td>7.50</td>
<td>0.32</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td>_IO_putc</td>
</tr>
<tr>
<td>5.00</td>
<td>0.34</td>
<td>0.02</td>
<td>12</td>
<td>1666.67</td>
<td>14166.67</td>
<td>Life::update(void)</td>
</tr>
<tr>
<td>5.00</td>
<td>0.36</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td>stdiobuf::overflow(int)</td>
</tr>
<tr>
<td>5.00</td>
<td>0.38</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td>stdiobuf::sys_write(char const *, int)</td>
</tr>
<tr>
<td>2.50</td>
<td>0.39</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>ostream::operator&lt;&lt;(char)</td>
</tr>
<tr>
<td>2.50</td>
<td>0.40</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>internal_mcount</td>
</tr>
<tr>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>12</td>
<td>0.00</td>
<td>0.00</td>
<td>Life::print(void)</td>
</tr>
<tr>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>12</td>
<td>0.00</td>
<td>0.00</td>
<td>to_continue(void)</td>
</tr>
<tr>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>Life::initialize(void)</td>
</tr>
<tr>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>instructions(void)</td>
</tr>
<tr>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>170000.00</td>
<td>main</td>
</tr>
</tbody>
</table>
### gprof: call graph

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>0.15</td>
<td>12/12</td>
<td></td>
<td></td>
<td>main [2]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.17</td>
<td>1</td>
<td></td>
<td></td>
<td>_start [3]</td>
</tr>
<tr>
<td>0.02</td>
<td>0.15</td>
<td>12/12</td>
<td></td>
<td></td>
<td>main [2]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.17</td>
<td>1</td>
<td></td>
<td></td>
<td>_start [3]</td>
</tr>
<tr>
<td>0.15</td>
<td>0.00</td>
<td>48000/48000</td>
<td></td>
<td></td>
<td>Life::update(void) [1]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>12/12</td>
<td></td>
<td></td>
<td>main [2]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>12/12</td>
<td></td>
<td></td>
<td>Life::print(void) [13]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>12/12</td>
<td></td>
<td></td>
<td>to_continue(void) [14]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td></td>
<td>instructions(void) [16]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td></td>
<td>Life::initialize(void) [15]</td>
</tr>
<tr>
<td>0.15</td>
<td>0.00</td>
<td>48000/48000</td>
<td></td>
<td></td>
<td>Life::update(void) [1]</td>
</tr>
<tr>
<td>0.15</td>
<td>0.00</td>
<td>48000</td>
<td></td>
<td></td>
<td>Life::neighbor_count(int, int) [4]</td>
</tr>
</tbody>
</table>
Version Control

- **Subversion (svn), cvs, proprietary tools**
  - [http://subversion.tigris.org/](http://subversion.tigris.org/)
  - [http://svnbook.red-bean.com/](http://svnbook.red-bean.com/)

- **Useful for individual projects; *crucial* for multiple developers**

- **Functionality**
  - central repository for code
  - version history, rollbacks
  - revision merging
  - conflict detection, resolution assistance
  - change lists, logging
  - branch management
  - tagging
Typical Use

- Create a repository
- Import some files into it
- Check out a local “working directory”
- Make changes
- Update to latest version
- Fix any conflicts, test changes in latest version
- Check in your changes
Questions?