Lecture 13:
C++ streams
Outline

• Announcements
• Review
• Project 3C
• C++ streams
• Project 3D
Outline

• Announcements
• Review
• Project 3C
• C++ streams
• Project 3D
Announcements

• Projects:
  – 3D assigned today, due Tuesday
• “Optional”
  – You get 0%/3% if you skip (so not optional)
  – But you don’t need it for 3E, 3F, etc
“Reproducers”

• Very hard to debug your problems with partial information
  – Just a compiler error
  – Just one of several source files
  – … leads to extra iterations and extra work for me

• Reproducer: a self-contained environment that reproduces your problem.

• Ideally:
  – all your source code
  – a Makefile
  – the compile error message or output from the code that is problematic

Please do your best to send me good reproducers. This is good practice for later in your career.
Functions in header files
Outline

• Announcements
• Review
• Project 3C
• C++ streams
• Project 3D
• Memory Errors
• Project 4A
C++ memory management

• C++ provides new constructs for requesting heap memory from the memory manager
  – stack memory management is not changed
    • (automatic before, automatic now)

• Allocate memory: “new”
• Allocate memory: “delete”
new / delete syntax

```c
fawcett:330 child$ cat new.C
int main()
{
    int *oneInt = new int;
    *oneInt = 3;
    int *intArray = new int[3];

    delete oneInt;
    delete [] intArray;
}
```

No header necessary

Allocating array and single value is the same.

new knows the type and allocates the right amount.

new int $\rightarrow$ 4 bytes
new int[3] $\rightarrow$ 12 bytes

Deleting array takes [], deleting single value doesn’t.
new calls constructors for your classes

• Declare variable in the stack: constructor called
• Declare variable with “malloc”: constructor not called
  – C knows nothing about C++!
• Declare variable with “new”: constructor called
Destructors

• A destructor is called automatically when an object goes out of scope (via stack or delete)

• A destructor’s job is to clean up before the object disappears
  – Deleting memory
  – Other cleanup (e.g., linked lists)

• Same naming convention as a constructor, but with a prepended ~ (tilde)
Destructors example

```c++
struct Pixel
{
    unsigned char R, G, B;
};

class Image
{
public:
    Image(int w, int h);
    ~Image();
private:
    int width, height;
    Pixel *buffer;
};

Image::Image(int w, int h)
{
    width = w; height = h;
    buffer = new Pixel[width*height];
}

Image::~Image()
{
    delete [] buffer;
}
```

Class name with ~ prepended

Defined like any other method, does cleanup

If Pixel had a constructor or destructor, it would be getting called (a bunch) by the new’s and delete’s.
Inheritance and Constructors/ Destructors: Example

• Constructors from base class called **first**, then next derived type second, and so on.

• Destructor from base class called **last**, then next derived type second to last, and so on.

• Derived type always assumes base class exists and is set up
  – ... base class never needs to know anything about derived types
#include <stdio.h>

class C
{
    public:
        C() { printf("Constructing C\n"); }
        ~C() { printf("Destructing C\n"); }
};

class D : public C
{
    public:
        D() { printf("Constructing D\n"); }
        ~D() { printf("Destructing D\n"); }
};

int main()
{
    printf("Making a D\n");
    { 
        D b;
    }

    printf("Making another D\n");
    { 
        D b;
    }
}
Possible to get the wrong destructor

• With a constructor, you always know what type you are constructing.
• With a destructor, you don’t always know what type you are destructing.
• This can sometimes lead to the wrong destructor getting called.

Solution: always use virtual destructor!
Objects in objects

```c++
#include <stdio.h>

class A {
  public:
    A() { printf("Constructing A\n"); }
    ~A() { printf("Destructing A\n"); }
};

class B {
  public:
    B() { printf("Constructing B\n"); }
    ~B() { printf("Destructing B\n"); }
};

class C {
  public:
    C() { printf("Constructing C\n"); }
    ~C() { printf("Destructing C\n"); }
  private:
    A a;
    B b;
};

int main() {
  C c;
}
```

fawcett:330 childds$ ./a.out
Constructing A
Constructing B
Constructing C
Destructing C
Destructing B
Destructing A
Initializers

- New syntax to have variables initialized before even entering the constructor

```c
#include <stdio.h>

class A
{
    public:
        A() : x(5)
        {
            printf("x is %d\n", x);
        }
    private:
        int x;
};

int main()
{
    A a;
}
```
Initializers

• Initializers are a mechanism to have a constructor pass arguments to another constructor

• Needed because
  – Base class constructors are called before derived constructors & need to pass arguments in derived constructor to base class
  – Constructors for objects contained in a class are called before the container class & need to pass arguments in container class’s destructor
Initializers

- Needed because
  - Base class constructors are called before derived constructors & need to pass arguments in derived constructor to base class

```c++
#include <stdio.h>

class A
{
    public:
        A(int x) { v = x; }
    private:
        int v;
};

class B
{
    public:
        B(int x) { v = x; }
    private:
        int v;
};

class C
{
    public:
        C(int x, int y): b(x), a(y) { }
    private:
        B b;
        A a;
};

int main()
{
    C c(3,5);
}
```
Initializers

• Needed because
  – Constructors for objects contained in a class are called before the container class & need to pass arguments in container class’s destructor
The “is a” test

- Inheritance should be used when the “is a” test is true
- Base class: Shape
- Derived types: Triangle, Rectangle, Circle
  - A triangle “is a” shape
  - A rectangle “is a” shape
  - A circle “is a” shape

You can define an interface for Shapes, and the derived types can fill out that interface.

I will do a live coding example of this next week, and will discuss how C++ implements virtual functions.
Multiple inheritance

• A class can inherit from more than one base type
• This happens when it “is a” for each of the base types
  – Inherits data members and methods of both base types
Multiple inheritance

class Professor
{
    void Teach();
    void Grade();
    void Research();
};

class Father
{
    void Hug();
    void Discipline();
};

class Hank : public Father, public Professor
{
};
Diamond-Shaped Inheritance

- Base A, has derived types B and C, and D inherits from both B and C.
  - Which A is D dealing with??

- Diamond-shaped inheritance is controversial & really only for experts
  - (For what it is worth, we make heavy use of diamond-shaped inheritance in my project)
Pure Virtual Functions

- Pure Virtual Function: define a function to be part of the interface for a class, but do not provide a definition.
- Syntax: add “=0” after the function definition.
- This makes the class be “abstract”
  - It cannot be instantiated
- When derived types define the function, then they are “concrete”
  - They can be instantiated
Pure Virtual Functions Example

```cpp
class Shape
{
  public:
    virtual double GetArea(void) = 0;
};

class Rectangle : public Shape
{
  public:
    virtual double GetArea() { return 4; }
};

int main()
{
  Shape s;
  Rectangle r;
}
```

fawcett:330 childs$ g++ pure_virtual.C
pure_virtual.C: In function ‘int main()’:
pure_virtual.C:15: error: cannot declare variable ‘s’ to be of abstract type ‘Shape’
pure_virtual.C:2: note: because the following virtual functions are pure within ‘Shape’:
pure_virtual.C:4: note:     virtual double Shape::GetArea()
Outline

• Announcements
• Review
• Project 3C
• C++ streams
• Project 3D
Assignment: make your code base be data flow networks with OOP
Project 3C

CIS 330: Project #3B
Assigned: May 7th, 2014
Due: May 14th, 2014
(which means submitted by 6am on May 15th, 2014)
Worth 5% of your grade

Please read this entire prompt!

Assignment: Change your 3B project to be object-oriented.

== New code available on the website ==

=== main3C.C ===

Start with my main3C.C. It shows what the interfaces should be for the modules. Do not modify my main3C.C, aside from adding "#includes" and print statements (if you want).

=== Makefile ===

I added another Makefile. Note that I put all of my filters in a file called "filters.C" / "filters.h". I didn’t think it was worth splitting them into separate files.
Outline

- Announcements
- Review
- Project 3C
- C++ streams
- Project 3D
C++ lets you define operators

• You declare a method that uses an operator in conjunction with a class
  – +, -, /, !, ++, etc.
• You can then use operator in your code, since the compiler now understands how to use the operator with your class
• This is called “operator overloading”
  – ... we are overloading the use of the operator for more than just the simple types.
Example of operator overloading

```cpp
class MyInt {
  public:
    MyInt(int x) { myInt = x; }
    MyInt& operator++();

  protected:
    int myInt;
};

MyInt & MyInt::operator++() {
    myInt++;
    return *this;
}

int main() {
    MyInt mi(6);
    ++mi;
    ++mi;
    printf("Value is %d\n", mi.GetValue());
}
```

- Define operator ++ for MyInt
- Declare operator ++ will be overloaded for MyInt
- We will learn more about operator overloading later in the quarter.
- Call operator ++ on MyInt.
New operators: << and >>

• “<<”: Insertion operator
• “>>”: Extraction operator
  – Operator overloading: you can define what it means to insert or extract your object.

• Often used in conjunction with “streams”
  – Recall our earlier experience with C streams
    • stderr, stdout, stdin
  – Streams are communication channels
cout: the C++ way of accessing stdout

New header file (and no ".

New way of accessing stdout stream.

Insertion operation (<<)
cout is in the “standard” namespace

```
#include <iostream>

using std::cout;

int main()
{
    cout << "The answer is: ";
    cout << 8;
    cout << "\n";
}
```

“using” command puts the “cout” portion of the standard namespace (“std”) in the global namespace.

Don’t need “std::cout” any more...
endl: the C++ endline mechanism

• prints a newline
• flushes the stream
  – C version: fflush(stdout)
  – This is because printf doesn’t always print when you ask it to.
    • It buffers the requests when you make them.
    • This is a problem for debugging!!
endl in action

```cpp
#include <iostream>

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

int main()
{
    cout << "The answer is: ";
    cout << 8;
    cout << endl;
}
```

<< and >> have a return value

• ostream & ostream::operator<<(int);
  – (The signature for a function that prints an integer)

• The return value is itself
  – i.e., the cout object returns “cout”

• This allows you to combine many extractions (or insertions) in a single line.
  – This is called “cascading”.
Cascading in action

```cpp
fawcett:330 childds$ cat printCPP.C
#include <iostream>

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

int main()
{
    cout << "The answer is: " << 8 << endl;
}
fawcett:330 childds$ g++ printCPP.C
fawcett:330 childds$  
```
Putting it all together

fawcett:330 childs$ cat print.c
#include <stdio.h>

int main()
{
    printf("The answer is: ");
    printf("%d", 8);
    printf("\n");
}
fawcett:330 childs$ gcc print.c
fawcett:330 childs$ ./a.out
The answer is: 8

fawcett:330 childs$ cat printCPP.C
#include <iostream>

int main()
{
    std::cout << "The answer is: ";
    std::cout << 8;
    std::cout << "\n";
}
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$ ./a.out
The answer is: 8

fawcett:330 childs$ cat printCPP.C
#include <iostream>

int main()
{
    printf("The answer is: %d\n", 8);
}
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$ ./a.out
The answer is: 8

fawcett:330 childs$ cat printCPP.C
#include <iostream>

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

int main()
{
    cout << "The answer is: " << 8 << endl;
}
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$ ./a.out
The answer is: 8
Three pre-defined streams

- `cout <= => fprintf(stdout, ...`
- `cerr <= => fprintf(stderr, ...`
- `cin <= => fscanf(stdin, ...`
cin in action

```cpp
#include <iostream>

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

int main()
{
    int X, Y, Z;
    cin >> X >> Y >> Z;
    cout << Z << ", " << Y << ", " << X << endl;
}
```

```bash
fawcett:330 childs$ ./a.out
3 5
4
4, 5, 3
```
cerr

- Works like cout, but prints to stderr
- Always flushes everything immediately!

```
#include <iostream>

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

int main()
{
    int *X = NULL;
    stream << "The value is ";
    stream << *X << endl;
}
```

"See the error"
fstream

- ifstream: input stream that does file I/O
- ofstream: output stream that does file I/O

• Not lecturing on this, since it follows from:
  - C file I/O
  - C++ streams

http://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm
Outline

• Announcements
• Review
• Project 3C
• C++ streams
• Project 3D
Now show Project 2G in C++
Project 3D

• Assigned: today, 5/14
• Due: Tuesday, 5/20
• Important: if you skip this project, you will still be able to do future projects (3E, 3F, etc)
• Assignment:
  – Write PNMreaderCPP and PNMwriterCPP ... new version of the file reader and writer that use fstream.
Unix and Windows difference

• Unix:
  – “\n”: goes to next line, and sets cursor to far left

• Windows:
  – “\n”: goes to next line (cursor does not go to left)
  – “\m”: sets cursor to far left

• Text files written in Windows often don’t run well on Unix, and vice-versa
  – There are more differences than just newlines

vi: “set ff=unix” solves this
memcpy

MEMCPY(3)          BSD Library Functions Manual          MEMCPY(3)

NAME
  memcpy -- copy memory area

LIBRARY
  Standard C Library (libc, -lc)

SYNOPSIS
  #include <string.h>

  void *
  memcpy(void *restrict dst, const void *restrict src, size_t n);

DESCRIPTION
  The memcpy() function copies n bytes from memory area src to memory area
dst. If dst and src overlap, behavior is undefined. Applications in
which dst and src might overlap should use memmove(3) instead.

RETURN VALUES
  The memcpy() function returns the original value of dst.
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!!