CIS 330:

Univ and OCP

Lecture 15:
methods invocations &
virtual function tables
Announcements

- Proj 3C and 3D due 5/19
- Hank’s Tuesday OH are shifted: 230pm-330pm
- Final on Friday June 9th @ 10am
Review
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()
Project 3C
Project 3C

CIS 330: Project #3C
Assigned: May 7th, 2016
Due May 17th, 2016
(which means submitted by 6am on May 18th, 2016)
Worth 7% of your grade

*Please read this entire prompt!*

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

3D will be due on May 17 as well.
BUT: you can skip 3D.
You get 0/3 points.
But you don’t need 3D to do 3E-3I.
Assignment: make your code base be data flow networks with OOP
More C++
C++ lets you define operators

• You declare a method that uses an operator in conjunction with a class
  – +, -, /, !, ++, etc.

• You can then use your 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.

You can also do this with functions.
Example of operator overloading

class MyInt
{
    public:
        MyInt(int x) { myInt = x; }
    private:
        int myInt;

    MyInt& operator++();

    int GetMyInt() const
    {
        return myInt;
    }

    void Print() const
    {
        printf("Value is %d
", GetMyInt());
    }
};

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

int main()
{
    MyInt mi(6);
    ++mi;
    ++mi;
    printf("Value is %d
", mi.GetMyInt());
}

fawcett:330 childs$ ./a.out
Value is 8
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
Putting it all together

```c
#include <stdio.h>

int main()
{
    printf("The answer is: ");
    printf("%d", 8);
    printf("\n");
}
```

```cpp
#include <iostream>

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

```c
#include <stdio.h>

int main()
{
    printf("The answer is: %d\n", 8);
}
```

```cpp
#include <iostream>

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

```bash
fawcett:330 childs$ gcc print.c
fawcett:330 childs$ ./a.out
The answer is: 8
```

```bash
fawcett:330 childs$ g++ print.C
fawcett:330 childs$ ./a.out
```

```bash
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$ ./a.out
```

```bash
fawcett:330 childs$ g++ printCPP.C
fawcett:330 childs$
```
Three pre-defined streams

- `cout <= => fprintf(stdout, ...`
- `cerr <= => fprintf(stderr, ...`
- `cin <= => fscanf(stdin, ...`
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
Project 3D

• Assigned: today, 5/12
• Due: Friday May 19
• 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.
Default Methods

• C++ makes 4 methods for you by default:
  – Default constructor
  – Copy constructor
  – Assignment operator
  – Destructor

```cpp
class A
{
    A() { ; };
    A(A &a) { ; };
    ~A() { ; };
    A & operator=(A &a) { ; };

};
```
What if there are data members?

class A
{
    public:
        A() { ; };
        A(A &a) { x = a.x; };
        A & operator=(A &a) { x = a.x; return a; };
    private:
        int x;
};
class Image
{
    public:
        Image() { buffer = NULL; };
    ~Image() { if (buffer != NULL) delete [] buffer; };
    Resize() { ... };
    private:
        Pixel *buffer;
};

int main()
{
    Image i;
    i.ResizeSize(1000, 1000);
    Image i2 = i;
}

THIS WILL CRASH
Solution

```cpp
class Image
{
public:
    Image() { buffer = NULL; };
    ~Image() { if (buffer != NULL) delete [] buffer; };
    ResetSize() { ... };
private:
    Pixel *buffer;
    Image(Image &i) { ; };
    Image &operator=(Image &i) { ; };
};
```

• This will prevent you from accidentally calling copy constructor or assignment operator
• (You should add this to your Image class)
And you may be using assignment operators right now without knowing it…

```c
#include <stdio.h>

struct Pixel
{
    unsigned char r, g, b;
    Pixel &operator=(Pixel &p)
    {
        printf("Calling pixel assignment operator\n");
        r = p.r;
        g = p.g;
        b = p.b;
    }
};

int main()
{
    Pixel p1;
    p1.r = 255;
    p1.g = 0;
    p1.b = 0;

    Pixel p2;
    p2 = p1;
}
```

• … so “=” is doing more work than you might expect
New stuff
Inline function

• inlined functions:
  – hint to a compiler that can improve performance
  – basic idea: don’t actually make this be a separate function that is called
    • Instead, just pull the code out of it and place it inside the current function
  – new keyword: inline

```c
inline int doubler(int X) {
    return 2*X;
}

int main() {
    int Y = 4;
    int Z = doubler(Y);
}
```

The compiler sometimes refuses your inline request (when it thinks inlining won’t improve performance), but it does it silently.
Inlines can be automatically done within class definitions

- Even though you don’t declare this as inline, the compiler treats it as an inline

```cpp
class MyDoublerClass
{
    int doubler(int X) { return 2*X; }
};
```
You should only do inlines within header files

Left: function is inlined in every .C that includes it ... no problem

Right: function is defined in every .C that includes it ... duplicate symbols
New Content
How C++ Does Methods
“this”: pointer to current object

• From within any struct’s method, you can refer to the current object using “this”
How methods work under the covers (1/4)

class MyIntClass
{
    public:
        MyIntClass(int x) { myInt = x; }

    friend void FriendIncrementFunction(MyIntClass *);
    int GetMyInt() { return myInt; }

    protected:
        int myInt;
};

void FriendIncrementFunction(MyIntClass *mic)
{
    mic->myInt++;
}

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    FriendIncrementFunction(&MIC);
    cout << "My int is " << MIC.GetMyInt() << endl;
}
How methods work under the covers (2/4)

class MyIntClass
{
    public:
        MyIntClass(int x) { myInt = x; };
        friend void FriendIncrementFunction(MyIntClass *); // Method definition
        int GetMyInt() { return myInt; };
    
    protected:
        int myInt;
};

void FriendIncrementFunction(MyIntClass *mic)
{
    mic->myInt++;
}

int main()
{
    MyIntClass MIC(12); // Initialize MIC with value 12
    FriendIncrementFunction(&MIC); // Call FriendIncrementFunction
    FriendIncrementFunction(&MIC); // Call FriendIncrementFunction again
    cout << "My int is " << MIC.GetMyInt() << endl;
}
How methods work under the covers (3/4)

class MyIntClass
{
  public:
    MyIntClass(int x) { myInt = x; }

    friend void FriendIncrementFunction(MyIntClass *);
    void IncrementMethod(void);
    int GetMyInt() { return myInt; }

  protected:
    int myInt;
};

void FriendIncrementFunction(MyIntClass *mic)
{
  mic->myInt++;
}

void MyIntClass::IncrementMethod(void)
{
  this->myInt++;
}

int main()
{
  MyIntClass MIC(12);
  FriendIncrementFunction(&MIC);
  MIC.IncrementMethod();
  cout << "My int is " << MIC.GetMyInt() << endl;
}
How methods work under the covers (4/4)

The compiler secretly slips “this” onto the stack whenever you make a method call.

It also automatically changes “myInt” to this->myInt in methods.

```cpp
class MyIntClass {

private:
  int myInt;

public:
  void increment() {
    this->myInt++;  // Increment self
  }

  int getMyInt() const {
    return this->myInt;  // Use self
  }

};

int main() {
  MyIntClass MIC(12);
  FriendIncrementFunction(&MIC);
  MIC.increment();  // Increment self
  cout << "My int is " << MIC.getMyInt() << endl;
}
```

### Variable Table

<table>
<thead>
<tr>
<th>Addr.</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000</td>
<td>MIC/myInt</td>
<td>12</td>
</tr>
<tr>
<td>0x8004</td>
<td>mic</td>
<td>0x8000</td>
</tr>
<tr>
<td>0x8000</td>
<td>MIC/myInt</td>
<td>13</td>
</tr>
<tr>
<td>0x8004</td>
<td>this</td>
<td>0x8000</td>
</tr>
</tbody>
</table>
Virtual Function Tables
Virtual functions

• Virtual function: function defined in the base type, but can be re-defined in derived type.
• When you call a virtual function, you get the version defined by the derived type
#include <stdio.h>

struct SimpleID {
    int id;
    virtual int GetIdentifier() { return id; }
};

struct ComplexID : SimpleID {
    int extraId;
    virtual int GetIdentifier() { return extraId*128+id; }
};

int main() {
    ComplexID cid;
    cid.id = 3;
    cid.extraId = 3;
    printf("ID = %d\n", cid.GetIdentifier());
}
Picking the right virtual function

```cpp
class A {
    public:
        virtual const char *GetTypeDef() { return "A"; }
};

class B : public A {
    public:
        virtual const char *GetTypeDef() { return "B"; }
};

int main() {
    A a;
    B b;
    cout << "a is " << a.GetTypeDef() << endl;
    cout << "b is " << b.GetTypeDef() << endl;
}
```

It seems like the compiler should be able to figure this out ... it knows that a is of type A and it knows that b is of type B
Picking the right virtual function

class A
{
  public:
    virtual const char *GetType() { return "A"; }
};

class B : public A
{
  public:
    virtual const char *GetType() { return "B"; }
};

void ClassPrinter(A *ptrToA)
{
  cout << "ptr points to a " << ptrToA->GetType() << endl;
}

int main()
{
  A a;
  B b;

  ClassPrinter(&a);
  ClassPrinter(&b);
}

So how to does the compiler know?
How does it get “B” for “b” and “A” for “a”??
Virtual Function Table

• Let C be a class and X be an instance of C.
• Let C have 3 virtual functions & 4 non-virtual functions
• C has a hidden data member called the “virtual function table”
• This table has 3 rows
  – Each row has the correct definition of the virtual function to call for a “C”.
• When you call a virtual function, this table is consulted to locate the correct definition.
Showing the existence of the virtual function pointer with sizeof()

```c++
class A {
public:
  virtual
};
class B : public A {
public:
  virtual
};
class C {
public:
  const char *Get Type() { return "C"; };
};
int main()
{
  A a;
  B b;

cout << "Size of A is " << sizeof(A) << endl;
cout << "Size of a pointer is " << sizeof(int *) << endl;
cout << "Size of C is " << sizeof(C) << endl;
}
```

empty objects have size of 1? why?!?

Answer: so every object has a unique address.

```
fawcett:330 child$d$ ./a.out
Size of A is 8
Size of a pointer is 8
Size of C is 1
```
Virtual Function Table

• Let C be a class and X be an instance of C.
• Let C have 3 virtual functions & 4 non-virtual functions
• Let D be a class that inherits from C and Y be an instance of D.
  – Let D add a new virtual function
• D’s virtual function table has 4 rows
  – Each row has the correct definition of the virtual function to call for a “D”.
More notes on virtual function tables

• There is one instance of a virtual function table for each class
  – Each instance of a class shares the same virtual function table

• Easy to overwrite (i.e., with a memory error)
  – And then all your virtual function calls will be corrupted
  – Don’t do this! ;)

Virtual function table: example

CIS 330: Project #2C
Assigned: April 17th, 2014
Due April 24th, 2014
(which means submitted by 6am on April 25th, 2014)
Worth 6% of your grade

Please read this entire prompt!

Assignment: You will implement subtypes with C.

1) Make a union called ShapeUnion with the three types (Circle, Rectangle, Triangle).
2) Make a struct called FunctionTable that has pointers to functions.
3) Make an enum called ShapeType that identifies the three types.
4) Make a struct called Shape that has a ShapeUnion, a ShapeType, and a FunctionTable.
5) Modify your 9 functions to deal with Shapes.
6) Integrate with the new driver function. Test that it produces the correct output.
Virtual function table: example

class Shape
{
    virtual double GetArea() = 0;
    virtual void GetBoundingBox(double *) = 0;
};

class Rectangle : public Shape
{
    public:
        Rectangle(double, double, double, double, double);
    virtual double GetArea();
    virtual void GetBoundingBox(double *);
    protected:
        double minX, maxX, minY, maxY;
};

class Triangle : public Shape
{
    public:
        Triangle(double, double, double, double, double);
    virtual double GetArea();
    virtual void GetBoundingBox(double *);
    protected:
        double pt1X, pt2X, minY, maxY;
};
Questions

• What does the virtual function table look like for a Shape?

```c
typedef struct
{
    double (*GetArea)(Shape *);
    void (*GetBoundingBox)(Shape *, double *);
} VirtualFunctionTable;
```

• What does Shape’s virtual function table look like?
  
  – Trick question: Shape can’t be instantiated, precisely because you can’t make a virtual function table
  
  • abstract type due to pure virtual functions
Questions

• What is the virtual function table for Rectangle?

```c
  c->ft.GetArea = GetRectangleArea;
  c->ft.GetBoundingBox = GetRectangleBoundingBox;
```

• (this is a code fragment from my 2C solution)
Calling a virtual function

• Let X be an instance of class C.
• Assume you want to call the 4\textsuperscript{th} virtual function
• Let the arguments to the virtual function be an integer Y and a float Z.
• Then call:

```c
(X.vptr[3])(&X, Y, Z);
```

The pointer to the virtual function pointer (often called a vptr) is a data member of X

The 4\textsuperscript{th} virtual function has index 3 (0-indexing)

Secretly pass “this” as first argument to method
Inheritance and Virtual Function Tables

This whole scheme gets much harder with multiple inheritance, and you have to carry around multiple virtual function tables.

```cpp
class A
{
    public:
    virtual void Foo2();
};

class C : public B
{
    public:
    virtual void Foo1();
    virtual void Foo2();
    virtual void Foo3();
};
```

Same as B’s
This is how you can treat a C as a B

<table>
<thead>
<tr>
<th>Location of Foo1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location of Foo1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location of Foo2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location of Foo3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Virtual Function Table: Summary

- Virtual functions require machinery to ensure the correct form of a virtual function is called
- This is implemented through a virtual function table
- Every instance of a class that has virtual functions has a pointer to its class’s virtual function table
- The virtual function is called via following pointers
  - Performance issue
Now show Project 2D in C++

• Comment:
  – C/C++ great because of performance
  – Performance partially comes because of a philosophy of not adding “magic” to make programmer’s life easier
  – C has very little pixie dust sprinkled in
    • Exception: ‘\0’ to terminate strings
  – C++ has more
    • Hopefully this will demystify one of those things (virtual functions)
```
#include <iostream>
using std::cerr;
using std::endl;

class Shape
{
  public:
    int s;
    virtual double GetArea() = 0;
    virtual void GetBoundingBox(double *) = 0;
};

class Triangle : public Shape
{
  public:
    virtual double GetArea() { cerr << "In GetArea for Triangle" << endl; return 1; }
    virtual void GetBoundingBox(double *) { cerr << "In GetBBox for Triangle" << endl; }
};

class Rectangle : public Shape
{
  public:
    virtual double GetArea() { cerr << "In GetArea for Rectangle" << endl; return 2; }
    virtual void GetBoundingBox(double *) { cerr << "In GetBBox for Rectangle" << endl; }
};

struct VirtualFunctionTable
{
  double (*GetArea)(Shape *);
  void (*GetBoundingBox)(Shape *, double *);
};

int main()
{
  Rectangle r;
  cerr << "Size of rectangle is " << sizeof(r) << endl;
  VirtualFunctionTable *vft = *((VirtualFunctionTable**) &r);
  cerr << "Vptr = " << vft << endl;
  double d = vft->GetArea(&r);
  cerr << "Value = " << d << endl;
  double bbox[4];
  vft->GetBoundingBox(&r, bbox);
}
```
Project 3E

• You will need to think about how to accomplish the data flow execution pattern and think about how to extend your implementation to make it work.

• This prompt is vaguer than some previous ones
  — ... not all of the details are there on how to do it
Project 3E

• Worth 5% of your grade
• Will be assigned Wednesday
Pitfalls
This is using call-by-value, not call-by-reference.
struct Image
{
    int width;
    int height;
    unsigned char *buffer;
};

Image *ReadFromFile(char *filename)
{
    Image *rv = NULL;
    
    /* OPEN FILE, descriptor = f */
    /* ... */
    /* set up width w, and height h */
    /* ... */

    rv = malloc(sizeof(Image));
    rv->width = w;
    rv->height = h;
    fread(rv->buffer, sizeof(unsigned char), w*h, f);
}
Pitfall #3

- int *s = new int[6*sizeof(int)];
Pitfall #4

int main()
{
    // new black image
    int height = 1000, width = 1000;
    unsigned char *buffer = new unsigned char[3*width*height];
    for (int i = 0 ; i < sizeof(buffer) ; i++)
    {
        buffer[i] = 0;
    }
}

• Assume:
  int *X = new int[100];

• What is sizeof(X)?

• What is sizeof(*X)?
Pitfall #5

```c
/* struct definition */
struct Image {
    /* data members */
};

/* prototypes */
void WriteImage(Image *, const char *);

/* main */
int main()
{
    Image *img = NULL;
    /* set up Image */
    const char *filename = "out.pnm";
    WriteImage(img, filename);
}

/* WriteImage function */
void WriteImage(char *filename, Image *img)
{
    /* code to write img to filename */
}
```

```
fawcett:330 childs$ g++ write_image.c
Undefined symbols:
    "WriteImage(Image*, char const*)", referenced from:
    _main in ccSjC6w2.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
```
(not-a-)Pitfall #6

unsigned char* Image::getPixel(int i, int j) {
    int pixStart = 3*i*this->width+3+j;
    unsigned char *pixel = new unsigned char[3];
    pixel[0] = this->data[pixStart];
    pixel[1] = this->data[pixStart + 1];
    pixel[2] = this->data[pixStart + 2];
    return pixel;
}

unsigned char* Image::getPixel(int i, int j) {
    int pixStart = 3*i*this->width+3+j;
    return this->data+pixStart;
}

Top requires memory allocation / deletion, and does extra copy.
Pitfall #7

• For objects on the stack, the destructors are called when a function goes out of scope
  – You may have a perfectly good function, but it seg-faults on return

• Especially tricky for main
  – program ran to completion, and crashed at the very end
# Pitfall #8

```cpp
#include <stdlib.h>

class Image {
  public:
    Image() { width = 0; height = 0; buffer = NULL; };
    virtual ~Image() { delete [] buffer; };

    void ResetSize(int width, int height);
    unsigned char *GetBuffer(void) { return buffer; };

  private:
    int width, height;
    unsigned char *buffer;
};

void Image::ResetSize(int w, int h) {
  width = w;
  height = h;
  if (buffer != NULL)
    delete [] buffer;
  buffer = new unsigned char[3*width*height];
}

int main()
{
  Image img;
  unsigned char *buffer = img.GetBuffer();
  img.ResetSize(1000, 1000);
  for (int i = 0; i < 1000; i++)
    for (int j = 0; j < 1000; j++)
      for (int k = 0; k < 1000; k++)
        buffer[3*(i*1000+j)+k] = 0;
}
```
Bonus Topics
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!!
Web pages

- ssh -l <user name> ix.cs.uoregon.edu
- cd public_html
- put something in index.html
- → it will show up as

http://ix.cs.uoregon.edu/~<username>
Web pages

• You can also exchange files this way
  – scp file.pdf <username>@ix.cs.uoregon.edu:~/public_html
  – point people to http://ix.cs.uoregon.edu/~<username>/file.pdf

Note that ~/public_html/dir1 shows up as
http://ix.cs.uoregon.edu/~<username>/dir1
(“~/dir1” is not accessible via web)