Lecture 15:
Even more pointer stuff
Virtual function table
Any 3C, 3D questions?
PNMreader::PNMreader(char *f)
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
```csharp
blender.SetInput(tbconcat2.GetOutput());
blender.SetInput2(reader.GetOutput());
writer.SetInput(blender.GetOutput());
reader.Execute();
shrinker1.Execute();
lrconcat1.Execute();
tbconcat1.Execute();
shrinker2.Execute();
lrconcat2.Execute();
tbconcat2.Execute();
blender.Execute();
writer.Write(argv[2]);
}
```
Project 3E

- Worth 3% of your grade
- Assigned today, due May 23
Example of data flow (image processing)
Make it easy on yourself to run...

```
128-223-223-73-wireless:330 hank$ cat r
./proj3C 3C_input.pnm 3C_output.pnm
```
Other ways to make life easier

• tab from shell: auto-completes
• Ctrl-R: searches backwards in your shell history
More on Pointers
(Poor) Analogy

• Safe deposit box
(Poor) Analogy

• You go to the bank
• You ask for a safe deposit box
  – The key to the box is a pointer
• You get access to a space in the vault
  – The box in the vault is the memory on the heap
Analogy Continued

```cpp
int main()
{
    float *buffer = new float[1000];
    // ... 
    buffer = new float[100];
}
```

- You go to teller and request a safe deposit box
- Teller gives you a key, to box #105
- Then you go back the next day and request another safe deposit box, to box #107
- And you throw out the key to box #105 and only keep the key to box #107
- This is a memory leak
  - No one will ever be able to access to box #105
Now let’s think about stack/heap

```c
int main()
{
    float *buffer = new float[1000];
    // ...
    buffer = new float[100];
}
```
Analogy Continued

```
int main()
{
    float *buffer = new float[1000];
    // ...
    float *buffer2 = buffer;
}
```

- You go to teller and request a safe deposit box
- Teller gives you a key, to box #105 (buffer)
- You make a copy of the key for your friend (buffer2)
- Now you and your friend have access to box #105
- If your friend changes the contents, then it affects you
- Terminology: this is called a “shallow copy”
Now let’s think about stack/heap

```c
int main()
{
    float *buffer = new float[1000];
    // ...
    float *buffer2 = buffer;
}
```
Analogy Continued  (But Starting to Break Down)

```c
int main()
{
    float *buffer = new float[1000];
    // ...
    float *buffer2 = new float[1000];
    for (int i = 0 ; i < 1000 ; i++)
        buffer2[i] = buffer[i];
}
```

- You go to teller and request a safe deposit box
- Teller gives you a key, to box #105 (buffer)
- You fill the box
- You later request a second safe deposit box
- Teller gives you a key, to box #107 (buffer2)
- You examine box #105. Whatever is in 105, you put in 107
  - Example: $10K in 105. So put an additional $10K in 107. ($20K total)
- This is called a “deep copy”
Now let’s think about stack/heap

```c
int main()
{
    float *buffer = new float[1000];
    // ...
    float *buffer2 = new float[1000];
    for (int i = 0; i < 1000; i++)
        buffer2[i] = buffer[i];
}
```
Arrays on the stack are different

```c
int main()
{
    float A[3] = { 0.5, 1.5, 2.5 };
}
```

- You cannot re-assign A to another value.
  - A is bound to its stack location
  - But you can assign a pointer to point at A’s location.
  - And compiler can do this automatically (int *A_ptr = A;)

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fff0</td>
<td>0.5</td>
</tr>
<tr>
<td>0x7ffec</td>
<td>1.5</td>
</tr>
<tr>
<td>0x7ffe8</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Code | Data | Stack | Free

Heap
Default Methods

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

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

---->

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

int main()
{
    Image i;
    i.ResetSize(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…

• ... so “=” is doing more work than you might expect
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 *);  
    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;
}

<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>0x8000</td>
<td>MIC/myInt</td>
<td>12</td>
</tr>
<tr>
<td>0x8004</td>
<td>mic</td>
<td>0x8000</td>
</tr>
</tbody>
</table>
How methods work under the covers (3/4)

```cpp
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++;  // Added this line
}

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

int main()
{
    MyIntClass MIC(12);
    FriendIncrementFunction(&MIC);
    MIC.IncrementMethod();
    cout << "My int is " << MIC.GetMyInt() << endl;
}
```

```bash
child$ g++ this.C
child$ ./a.out
My int is 14
```
The compiler secretly slips “this” onto the stack whenever you make a method call.

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

```
class MyIntClass
{
  int myInt;
};

MyIntClass MIC(12);
FriendIncrementFunction(&MIC); // MIC.myInt++
MIC.IncrementMethod();
MIC.myInt++;
MIC.GetMyInt();
```

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

int main()
{
  MyIntClass MIC(12);
  FriendIncrementFunction(&MIC);
  MIC.IncrementMethod();
  cout << "My int is " << MIC.GetMyInt() << endl;
}
```
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
```c
#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());
}
```

```
#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());
}
128-223-223-72-wireless:330 hank$ g++ virtual.C
128-223-223-72-wireless:330 hank$ ./a.out
ID = 387
```
Picking the right virtual function

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

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

int main()
{
    A a;
    B b;

    cout << "a is " << a.GetType() << endl;
    cout << "b is " << b.GetType() << 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

```cpp
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 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()

empty objects have size of 1? why?!?

Answer: so every object has a unique address.

```cpp
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;
}
```

fawcett:330 childs$ ./a.out
Size of A is 8
Size of a pointer is 8
Size of C is 1

what will this print?
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:

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

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

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

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

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

Same as B’s
This is how you can treat a C as a B
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)
fawcett:vptr childs$ cat vptr.C
#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);
}
Pitfalls
Pitfall #1

```c
void AllocateBuffer(int w, int h, unsigned char **buffer)
{
    *buffer = new unsigned char[3*w*h];
}

int main()
{
    int w = 1000, h = 1000;
    unsigned char *buffer = NULL;
    AllocateBuffer(w, h, &buffer);
}
```

This is using call-by-value, not call-by-reference.
Pitfall #2

```c
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

```c
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)?
/* 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 */
}
(not-a-)Pitfall #6

```cpp
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;
}
```

```cpp
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

```c++
#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)