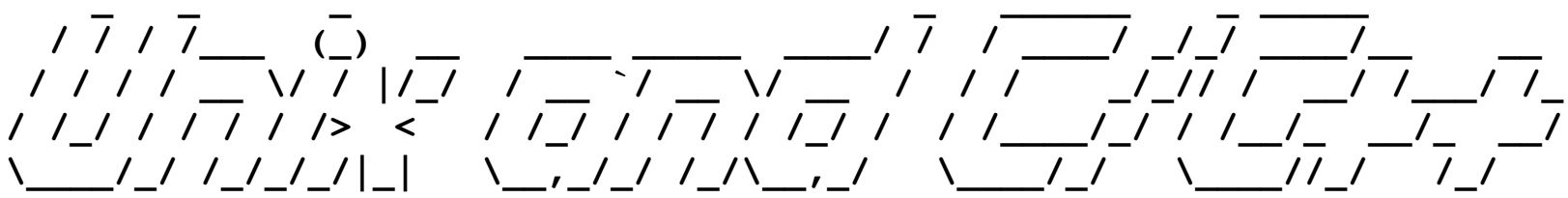
UNIVERSITY OF OREGON

CIS 330:



Lecture 15: exceptions

May 18th, 2016

Hank Childs, University of Oregon

Announcements

- Weekend OH?
- Extra Credit

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

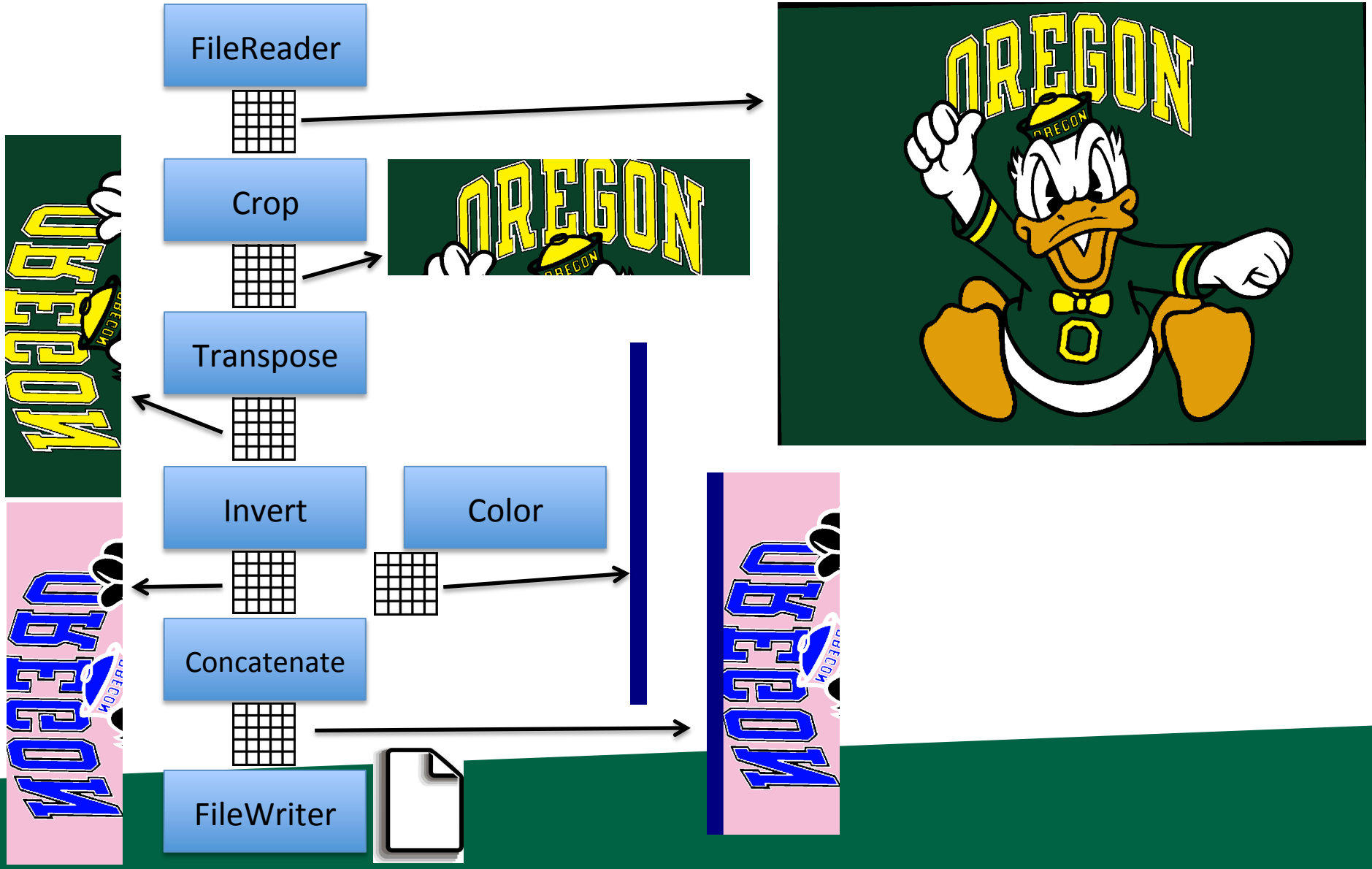
```
    blender.SetInput(tbconcat2.GetOutput());
    blender.SetInput2(reader.GetOutput());

    writer.SetInput(blender.GetOutput());

    reader.Execute();
    shrinker1.Execute();
    lrconcat1.Execute();
    tbconcat1.Execute();
    shrinker2.Execute();
    lrconcat2.Execute();
    tbconcat2.Execute();
    blender.Execute();

    writer.Write(argv[2]);
}
```

```
    blender.SetInput(tbconcat2.GetOutput());
    blender.SetInput2(reader.GetOutput());

    writer.SetInput(blender.GetOutput());

    blender.GetOutput()->Update();
    writer.Write(argv[2]);
}
```

Project 3E

- Worth 5% of your grade
- Assigned May 13, due ~~May 20th~~ 21st

Example of data flow (image processing)



Review: Access Control

Two contexts for access control

```
class A : public B {
    public:
        A() { x=0; y=0; };
        int foo() { x++; return foo2(); };
    private:
        int x, y;
        int foo2() { return x+y; };
};
```

defines how a class inherits from another class

defines access controls for data members and methods

Inheritance ("class A : public B")

- public → "is a"
  - (I never used anything but public)
- private → "implemented using"
  - (I have never used this, but see how it could be useful)
- protected → the internet can not think of any useful examples for this

Access Control

```
class Hank
{
   public/private/protected:
     BankAccount hanksId;
};
```

Access control type	Who can read it
private	Only Hank class
public	Anyone
protected	Those who inherit from Hank

Class Vs Struct

- Class:
  - Default inheritance is private
    - That's why you add public (class A : public B)
  - Default access control is private
- Struct:
  - Default inheritance is public
    - That's why you don't have to add public (struct A : B)
  - Default access control is public

Review: Lecture 14

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.

```
class  MyIntClass
{
...
Fri
{
    mic->myInt++;
}

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

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

Addr.	Variable	Value
0x8000	MIC/ myInt	12

Addr.	Variable	Value
0x8000	MIC/ myInt	12
0x8004	mic	0x8000

Addr.	Variable	Value
0x8000	MIC/ myInt	12
0x8004	this	0x8000

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

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);
}

fawcett:330 childs$ g++ virtual2.C
fawcett:330 childs$ ./a.out
                ??????
```

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()

```
class A
{
  public:
    virtual
};

class B : pub
{
  public:
    virtual
};

class C
{
  public:
    const char *GetType() { 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 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".

Questions

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

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

- What goes in Shape's virtual function table?
  - 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->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.
- Let the virtual function be the 4th 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 4th virtual function has index 3 (0-indexing)

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

Secretly pass "this" as first argument to method

Inheritance and Virtual Function Tables

```
class A
{
  public:
    virtual void Foo1();
    virtual void Foo2();
};

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

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

A	
	Location of Foo1
Foo2	Location of Foo2

Foo1	Location of Foo1
Foo2	Location of Foo2

C	
Foo1	Location of Foo1
Foo2	Location of Foo2
Foo3	Location of Foo3

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

vptr.C

```
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

```
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

```
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

```
/* 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

```
#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;
}
```

New Stuff: Misc. from OH

Make it easy on yourself to run…

```
128-223-223-73-wireless:330 hank$ cat r
./proj3C 3C_input.pnm 3C_output.pnm
128-223-223-73-wireless:330 hank$ chmod 755 r
128-223-223-73-wireless:330 hank$ ./r
```

Other ways to make life easier

- tab from shell: auto-completes
- Ctrl-R: searches backwards in your shell history

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)

- make clean ; make
- making Images in your functions and then copying them…

New Stuff: Exceptions

Exceptions

- C++ mechanism for handling error conditions
- Three new keywords for exceptions
  - try: code that you "try" to execute and hope there is no exception
  - throw: how you invoke an exception
  - catch: catch an exception … handle the exception and resume normal execution

Exceptions

```
fawcett:330 childs$ cat exceptions.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 105" << endl;
        throw 105;
        cout << "Done throwing 105" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
}
fawcett:330 childs$ g++ exceptions.C
```

Exceptions: catching multiple types

```
fawcett:330 childs$ cat exceptions2.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 105" << endl;
        throw 105;
        cout << "Done throwing 105" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
    catch (float &theFloat)
    {
        cout << "Caught a float: " << theFloat << endl;
    }
}
fawcett:330 childs$ g++ exceptions2.C
fawcett:330 childs$ ./a.out
About to throw 105
Caught an int: 105
```

Exceptions: catching multiple types

```
fawcett:330 childs$ cat exceptions3.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 10.5" << endl;
        throw 10.5;
        cout << "Done throwing 10.5" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
    catch (float &theFloat)
    {
        cout << "Caught a float: " << theFloat << endl;
    }
}
fawcett:330 childs$ g++ exceptions3.C
fawcett:330 childs$ ./a.out
About to throw 10.5
terminate called after throwing an instance of 'double'
Abort trap
```

Exceptions: catching multiple types

```
fawcett:330 childs$ cat exceptions4.C
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    try
    {
        cout << "About to throw 10.5" << endl;
        throw 10.5;
        cout << "Done throwing 10.5" << endl;
    }
    catch (int &theInt)
    {
        cout << "Caught an int: " << theInt << endl;
    }
    catch (float &theFloat)
    {
        cout << "Caught a float: " << theFloat << endl;
    }
    catch (double &theDouble)
    {
        cout << "Caught a double: " << theDouble << endl;
    }
}
```

```
fawcett:330 childs$ g++ exceptions4.C
fawcett:330 childs$ ./a.out
About to throw 10.5
Caught a double: 10.5
fawcett:330 childs$
```

Exceptions: throwing/catching complex types

```
class MyExceptionType  { };

class MemoryException : public MyExceptionType {};
class FailedAllocationException : public MemoryException {};
class NULLPointerException : public MemoryException {};

class FloatingPointException : public MyExceptionType {};
class DivideByZeroException : public FloatingPointException {};
class OverflowException : public FloatingPointException {};
```

```
void Foo();

int main()
{
    try
    {
        Foo();
    }
    catch (MemoryException &e)
    {
         cout << "I give up" << endl;
    }
    catch (OverflowException &e)
    {
         cout << "I think it is OK" << endl;
    }
    catch (DivideByZeroException &e)
    {
         cout << "The answer is bogus" << endl;
    }
}
```

Exceptions: cleaning up before you return

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (MyExceptionType &e)
    {
         delete [] arr;
         return NULL;
    }

    return arr;
}
```

Exceptions: re-throwing

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (MyExceptionType &e)
    {
         delete [] arr;
         throw e;
    }

    return arr;
}
```

Exceptions: catch and re-throw anything

```
void Foo(int *arr);

int *
Foo2(void)
{
    int  *arr = new int[1000];
    try
    {
        Foo(arr);
    }
    catch (...)
    {
        delete [] arr;
        throw;
    }

    return arr;
}
```

Exceptions: declaring the exception types you can throw

```
int *
MyIntArrayMemoryAllocator(int num) throw(FloatingPointException)
{
    int  *arr = new int[num];
    if (arr == NULL)
        throw DivideByZeroException();

    return arr;
}
```

Exceptions: declaring the exception types you can throw … not all it is cracked up to be

```
int *
MyIntArrayMemoryAllocator(int num) throw(FloatingPointException)
{
    int  *arr = new int[num];
    if (arr == NULL)
        throw MemoryException();

    return arr;
}
```

This will compile … compiler can only enforce this as a run-time thing.

As a result, this is mostly unused (I had to read up on it)

But: “standard” exceptions have a “throw” in their declaration.

std::exception

- c++ provides a base type called "std::exception"
- It provides a method called "what"

```
// using standard exceptions
#include <iostream>
#include <exception>
using namespace std;

class myexception: public exception
{
  virtual const char* what() const throw()
  {
    return "My exception happened";
  }
} myex;

int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << '\n';
  }
  return 0;
}
```

Source: cplusplus.com

Exceptions generator by C++ standard library

exception	description
bad_alloc	thrown by new on allocation failure
bad_cast	thrown by dynamic_cast when it fails in a dynamic cast
bad_exception	thrown by certain dynamic exception specifiers
bad_typeid	thrown by typeid
bad_function_call	thrown by empty function objects
bad_weak_ptr	thrown by shared_ptr when passed a bad weak_ptr

Source: cplusplus.com

3F

Project 3F in a nutshell

- Logging:
  - infrastructure for logging
  - making your data flow code use that infrastructure
- Exceptions:
  - infrastructure for exceptions
  - making your data flow code use that infrastructure

The webpage has a head start at the infrastructure pieces for you.

Warning about 3F

- My driver program only tests a few exception conditions
- Your stress tests later will test a lot more.
  - Be thorough, even if I'm not testing it

3F timeline

- Assigned today, due Weds

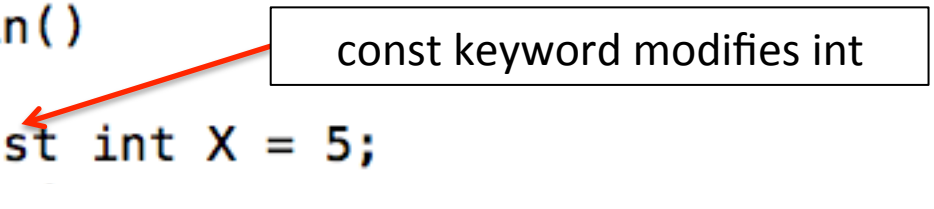
const

const

- const:
  - is a keyword in C and C++
  - qualifies variables
  - is a mechanism for preventing write access to variables

const example

```
fawcett:330 childs$ cat const1.C
int main()
{
    const int X = 5;

}
```



The compiler enforces const … just like public/ private access controls

Efficiency

```
int  NumIterations() { return 10; }

int main()
{
    int         count = 0;
    int         i;
    const int X = 10;
    int         Y = 10;
    for (i = 0 ; i < X ; i++)
        count++;
    for (i = 0 ; i < Y ; i++)
        count++;
    for (i = 0 ; i < NumIterations() ; i++)
        count++;
}
```

Are any of the three for loops faster than the others? Why or why not?

Answer: NumIterations is slowest … overhead for function calls.

Answer: X is probably faster than Y … compiler can do optimizations where it doesn't have to do "i < X" comparisons (loop unrolling)

const arguments to functions

- Functions can use const to guarantee to the calling function that they won't modify the arguments passed in.

```
struct Image
{
    int width, height;
    unsigned char *buffer;
};

ReadImage(char *filename, Image &);
WriteImage(char *filename, const Image &);
```

read function can't make the same guarantee

guarantees function won't modify the Image

const pointers

- Assume a pointer named “P”
- Two distinct ideas:
  - P points to something that is constant
    - P may change, but you cannot modify what it points to via P
  - P must always point to the same thing, but the thing P points to may change.

const pointers

```
int X = 4;
int *P = &X;
```

Idea #1:

violates const:

"*P = 3;"

OK:

"int Y = 5; P = &Y;"

pointer can change, but you can't modify the thing it points to

Idea #2:

violates const:

"int Y = 5; P = &Y;"

OK:

"*P = 3;"

pointer can't change, but you can modify the thing it points to

const pointers

```
int X = 4;
int *P = &X;
```

Idea #3:

violates const:

"*P = 3;"

"int Y = 5; P = &Y;"

OK:

none

pointer can't change, and you can't modify the thing it points to

const pointers

```
int X = 4;
int *P = &X;
```

Idea #1:

violates const:

"*P = 3;"

OK:

"int Y = 5; P = &Y;"

pointer can change, but you can't modify the thing it points to

```
fawcett:330 childs$ cat const3.C
int main()
{
    int X = 5;
    int Y = 6;
    const int *P;
    P  = &X;    // compiles
    P  = &Y;    // compiles
    *P = 7;     // won't compiles
}
fawcett:330 childs$ g++ const3.C
const3.C: In function 'int main()':
const3.C:8: error: assignment of read-only location
```

const goes before type

const pointers

```
int X = 4;
int *P = &X;
```

```
fawcett:330 childs$ cat const4.C
int main()
{
    int X = 5;
    int Y = 6;
    int * const P = &X; // must initialize
    *P = 7;     // compiles
    P  = &Y;    // won't compile
}
fawcett:330 childs$ g++ const4.C
const4.C: In function 'int main()':
const4.C:7: error: assignment of read-only variable 'P'
```

const goes after *

Idea #2:

violates const:

"int Y = 5; P = &Y;"

OK:

"*P = 3;"

pointer can't change, but you can modify the thing it points to

const pointers

```
int X = 4;
int *P = &X;
```

Idea #3:

violates const:

"*P = 3;"

"int Y = 5; P = &Y;"

OK:

none

pointer can't change, and you can't modify the thing it points to

const in both places

```
fawcett:330 childs$ cat const5.C
int main()
{
    int X = 5;
    int Y = 6;
    const int * const P = &X; // must initialize
    *P = 7;     // won't compile
    P  = &Y;    // won't compile
}
fawcett:330 childs$ g++ const5.C
const5.C: In function 'int main()':
const5.C:6: error: assignment of read-only location
const5.C:7: error: assignment of read-only variable 'P'
```

const usage

- class Image;
- const Image *ptr;
  - Used a lot: offering the guarantee that the function won't change the Image ptr points to
- Image * const ptr;
  - Helps with efficiency. Rarely need to worry about this.
- const Image * const ptr;
  - Interview question!!

Very common issue with const and objects

```
fawcett:330 childs$ cat const6.C
class Image
{
  public
    int

  priva
    int
};

unsigne
Allocat
{
    int
    unsi
    return rv;
}
```

How does compiler know GetNumberOfPixels doesn't modify an Image?

We know, because we can see the implementation.

But, in large projects, compiler can't see implementation for everything.

const functions with objects

```
fawcett:330 childs$ cat const7.C
class Image
{
  public:
    int    GetNumberOfPixels() const  { return width*height; };

  private:
    int    width, height;
};

unsigned char *
Allocator(const Image *img)
{
    int npixels = img->GetNumberOfPixels();
    unsigned char *rv = new unsigned char[3*npixels];
    return rv;
}
fawcett:330 childs$ g++ -c const7.C
fawcett:330 childs$
```

const after method name

If a class method is declared as const, then you can call those methods with pointers.

mutable

- mutable: special keyword for modifying data members of a class
  - If a data member is mutable, then it can be modified in a const method of the class.
  - Comes up rarely in practice.

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!!

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