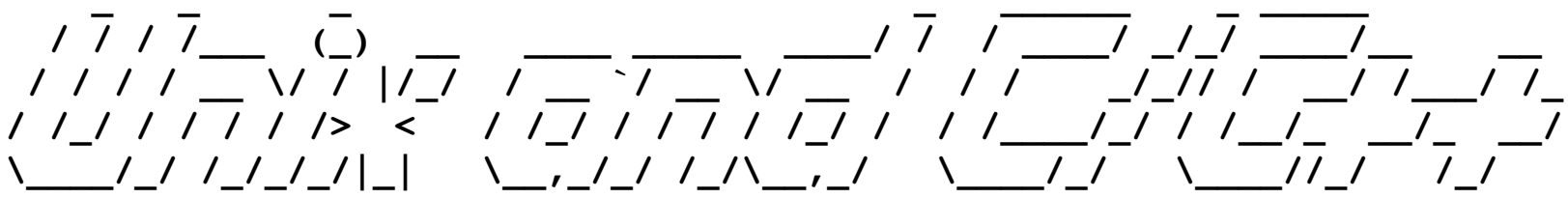
UNIVERSITY OF OREGON

CIS 330:



Lecture 17: performance analysis

May 25th, 2016

Hank Childs, University of Oregon

Announcements

- Weekend OH?
- Extra Credit

Announcements: Rest of Term

- 3G: assigned Monday (two days ago), due Monday (5 days from now)
- 3H, 4B, 4C: assigned this week, nominally due on Friday June 3rd
  - But: you can take until the Final to get them (3H, 4A, 4B) wrapped up (& they will not be late)

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 last week, due today

const

const

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

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 pointe

- 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.

nst pointer

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

Idea #1:

violates const:

“*P = 3;”

OK:

“int Y = 5; P = &Y;”

pointer <u>can</u> change, but you <u>can’t</u> modify the thing it points to

Idea #2:

violates const:

“int Y = 5; P = &Y;”

OK:

“*P = 3;”

pointer <u>can’t</u> change, but you <u>can</u> 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.

globals

globals

- You can create global variables that exist outside functions.

```
fawcett:Documents childs$ cat global1.C

#include <stdio.h>
int X = 5;

int main()
{
    printf("X is %d\n", X);
}

fawcett:Documents childs$ g++ global1.C
fawcett:Documents childs$ ./a.out
X is 5
fawcett:Documents childs$
```

Externs: mechanism for unifying global variables across multiple files

```
fawcett:330 childs$ cat file1.C

#include <stdio.h>

int count = 0;

int doubler(int);

int main()
{
    count++;
    doubler(3);
    printf("count is %d\n", count);
}
```

```
fawcett:330 childs$ cat file2.C
extern int count;

int doubler(int Y)
{
    count++;
    return 2*Y;
}
fawcett:330 childs$ g++ -c file1.C
fawcett:330 childs$ g++ -c file2.C
fawcett:330 childs$ g++ file1.o file2.o
fawcett:330 childs$ ./a.out
count is 2
```

extern: there's a global variable, and it lives in a different file.

static

- static memory: third kind of memory allocation
  - reserved at compile time
- contrasts with dynamic (heap) and automatic (stack) memory allocations
- accomplished via keyword that modifies variables

There are three distinct usages of statics

static usage #1: persistency within a function

```
fawcett:330 childs$ cat static1.C
#include <stdio.h>

int fibonacci()
{
    static int last2 = 0;
    static int last1 = 1;
    int rv = last1+last2;
    last2 = last1;
    last1 = rv;
    return rv;
}

int main()
{
    int i;
    for (int i = 0 ; i < 10 ; i++)
        printf("%d\n", fibonacci());
}
```

```
fawcett:330 childs$ g++ static1.C
fawcett:330 childs$ ./a.out
1
2
3
5
8
13
21
34
55
89
```

static usage #2: making global variables be local to a file

I have no idea why the static keyword is used in this way.

```
faw
#include <stdio.h>

static int count = 0;

int doubler(int);

int main()
{
   count++;
   doubler(3);
   printf("count is %d\n", count);
}
```

```
fawcett:330 childs$ cat file2.C
static int count = 0;

int doubler(int Y)
{
    count++;
    return 2*Y;
}
```

```
fawcett:330 childs$ g++ -c file2.C
fawcett:330 childs$ g++ file1.o file2.o
fawcett:330 childs$ ./a.out
count is 1
```

static usage #3: making a singleton for a class

```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

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

class MyClass
{
  public:
             MyClass()   { numInstances++; };
   virtual  ~MyClass()   { numInstances--; };

    int       GetNumInstances(void) { return numInstances; };

  private:
    int       numInstances;
};

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
fawcett:Downloads childs$ g++ static3.C
fawcett:Downloads childs$ ./a.out
Num instances = 1
Num instances = 0
fawcett:Downloads childs$
```

static usage #3: making a singleton for a class

```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

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

class MyClass
{
  public:
             MyClass()
   virtual  ~MyClass()

   int       GetNumInstan

  private:
   static int       numInstances;
};

i
{
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
```

```
fawcett:Downloads childs$ g++ static3.C
Undefined symbols:
  "MyClass::numInstances", referenced from:
      MyClass::MyClass()in ccoao8Hf.o
      MyClass::MyClass()in ccoao8Hf.o
      MyClass::GetNumInstances()      in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
      MyClass::~MyClass()in ccoao8Hf.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
```

We have to tell the compiler where to store this static.

What do we get?

static usage #3: making a singleton for a class

```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

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

class MyClass
{
  public:
             MyClass()    { numInstances++; };
   virtual  ~MyClass()    { numInstances--; };

    int       GetNumInstances(void) { return numInstances; };

  private:
    static int       numInstances;
};

int MyClass::numInstances = 0;

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << p[0].GetNumInstances() << endl;
}
```

static methods

```
fawcett:Downloads childs$ cat static3.C
#include <iostream>

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

class MyClass
{
  public:
             MyClass()    { numInstances++; };
   virtual  ~MyClass()    { numInstances--; };

   static int        GetNumInstances(void) { return numInstances; };

  private:
    static int        numInstances;
};

int MyClass::numInstances = 0;

int main()
{
    MyClass *p = new MyClass[10];
    cout << "Num instances = " << MyClass::GetNumInstances() << endl;
    delete [] p;
    cout << "Num instances = " << MyClass::GetNumInstances() << endl;
}
fawcett:Downloads childs$ g++ static3.C
fawcett:Downloads childs$ ./a.out
Num instances = 10
Num instances = 0
```

Static data members and static methods are useful and they are definitely used in practice

Scope

Scope Rules

- The compiler looks for variables:
  - inside a function or block
  - function arguments
  - data members (methods only)
  - globals

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];
}
```

- The compiler looks for variables:
  - inside a function or block
  - function arguments
  - data members (methods only)
  - globals

```
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;
}
```

Shadowing

- Shadowing is a term used to describe a "subtle" scope issue.
  - … i.e., you have created a situation where it is confusing which variable you are referring to

```
class Sink
{
  public:
    void SetInput(Image *i) { input = i; };
  protected:
    Image *input;
};

class Writer : public Sink
{
  public:
    void Write(void)  { /* write input */ };
  protected:
    Image *input;
};

int main()
{
   Writer writer;
   writer.SetInput(image);
   writer.Write();
}
```

Overloading Operators

More operator overloading

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

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

class Image
{
  public:
                        Image();

    friend ostream& operator<<(ostream &os, const Image &);

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

Image::Image()
{
    width  = 100;
    height = 100;
    buffer = NULL;
}

ostream &
operator<<(ostream &out, const Image &img)
{
    out << img.width << "x" << img.height << endl;
    if (img.buffer == NULL)
        out << "No buffer allocated!" << endl;
    else
        out << "Buffer is allocated!" << endl;
}
```

```
int main()
{
    Image img;
    cout << img;
}
fawcett:330 childs$ g++ oostream.C
fawcett:330 childs$ ./a.out
100x100
No buffer allocated!
```

Beauty of inheritance

- ostream provides an abstraction
  - That's all Image needs to know
    - it is a stream that is an output
  - You code to that interface
  - All ostream's work with it

```
int main()
{
    Image img;
    cerr << img;
}
fawcett:330 childs$ ./a.out
100x100
No buffer allocated!
```

```
int main()
{
    Image img;
    ofstream ofile("output_file");
    ofile << img;
}
fawcett:330 childs$ g++ oostream.C
fawcett:330 childs$ ./a.out
fawcett:330 childs$ cat output_file
100x100
No buffer allocated!
```

assignment operator

```
class Image
{
  public:
                    Image();
    void            SetSize(int w, int h);

    friend ostream& operator<<(ostream &os, const Image &);

    Image & operator=(const Image &);

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


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

```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated!
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated!
Image 2:200x200
Buffer is allocated!
```

```
Image &
Image::operator=(const Image &rhs)
{
    if (buffer != NULL)
        delete [] buffer;
    buffer = NULL;

    width  = rhs.width;
    height = rhs.height;
    if (rhs.buffer != NULL)
    {
        buffer = new unsigned char[3*width*height];
        memcpy(buffer, rhs.buffer, 3*width*height);
    }
}

int main()
{
    Image img1, img2;
    img1.SetSize(200, 200);
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
    img2 = img1;
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
}
```

let's do this again…

```
ostream &
operator<<(ostream &out, const Image &img)
{
    out << img.width << "x" << img.height << endl;
    if (img.buffer == NULL)
        out << "No buffer allocated!" << endl;
    else
        out << "Buffer is allocated, and value is "
            << (void *) img.buffer << endl;

    return out;
}
```

```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:200x200
Buffer is allocated, and value is 0x10081e600
```

(ok, fine)

let's do this again…

```
class Image
{
  public:
                        Image();
    void                SetSize(int w, int h);

    friend ostream& operator<<(ostream &os, const Image &);

    // Image & operator=(const Image &);

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

```
int main()
{
    Image img1, img2;
    img1.SetSize(200, 200);
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
    img2 = img1;
    cout << "Image 1:" << img1;
    cout << "Image 2:" << img2;
}
```

```
fawcett:330 childs$ g++ assignment_op.C
fawcett:330 childs$
```

it still compiled … why?

C++ defines a default assignment operator for you

- This assignment operator does a bitwise copy from one object to the other.
- Does anyone see a problem with this?

```
fawcett:330 childs$ ./a.out
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:0x0
No buffer allocated!
Image 1:200x200
Buffer is allocated, and value is 0x100800000
Image 2:200x200
Buffer is allocated, and value is 0x100800000
```

This behavior is sometimes OK and sometimes disastrous.

Copy constructors: same deal

- C++ automatically defines a copy constructor that does bitwise copying.
- Solutions for copy constructor and assignment operators:
  - Re-define them yourself to do “the right thing”
  - Re-define them yourself to throw exceptions
  - Make them private so they can’t be called

Project 3G

Please read this entire prompt!

Add 4 new filters:

1) Crop
2) Transpose
3) Invert
4) Checkerboard

Add 1 new source:

1) Constant color

Add 1 new sink:

1) Checksum

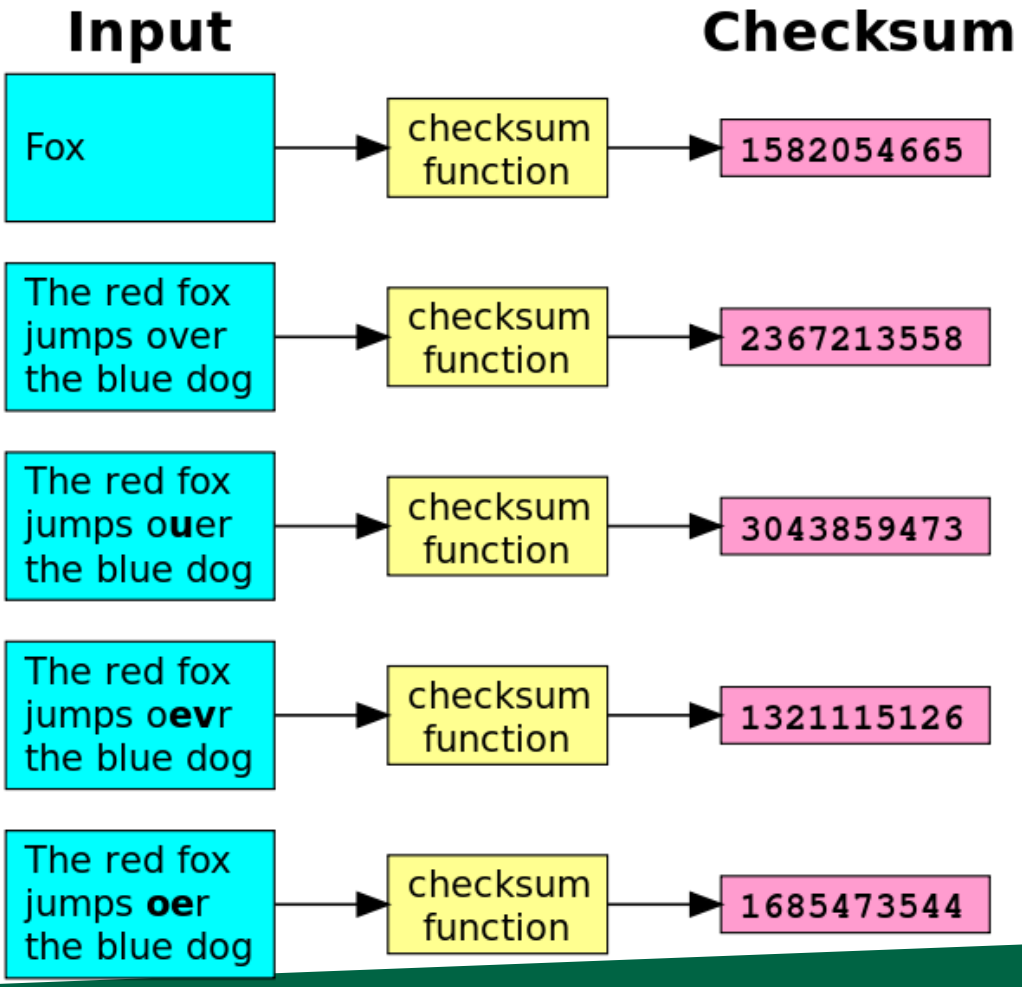
Plus: make the two image inputs in Sink be const pointers.

Assigned Monday (2 days ago), due Monday (5 days from now)

Stress Test Project (3H)

- We will have ~60 stress tests
- We can't check in 60 baseline images and difference them all
  - Will slow ix to a grind
- Solution:
  - We commit "essence of the solution"
  - We also complement that all images posted if needed.

Checksums



Most useful when input is very large and checksum is very small

From Wikipedia

Our “checksum”

- Three integers:
  - Sum of red channel
  - Sum of green channel
  - Sum of blue channel
- When you create a stress test, you register these three integers
- When you test against others stress tests, you compare against their integers
  - If they match, you got it right

This will be done with a derived type of Sink.

Should Checksums Match?

- On ix, everything should match
- On different architectures, floating point math won't match
- Blender: has floating point math
- → no blender

Performance Analysis

gettimeofday

GETTIMEOFDAY(2) BSD System Calls Manual GETTIMEOFDAY(2)

NAME

gettimeofday, settimeofday -- get/set date and time

SYNOPSIS

```
#include <sys/time.h>

int
gettimeofday(struct timeval *restrict tp, void *restrict tzp);

int
settimeofday(const struct timeval *tp, const struct timezone *tzp);
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the **gettimeofday**() call, and set with the **settimeofday**() call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in ``ticks.'' If tp is NULL and tzp is non-NULL, **gettimeofday**() will populate the timezone struct in tzp. If tp is non-NULL and tzp is NULL, then only the timeval struct in tp is populated. If both tp and tzp are NULL, nothing is returned.

The structures pointed to by tp and tzp are defined in <sys/time.h> as:

```
struct timeval {
        time_t       tv_sec;   /* seconds since Jan. 1, 1970 */
        suseconds_t  tv_usec;  /* and microseconds */
};

struct timezone {
        int     tz_minuteswest; /* of Greenwich */
        int     tz_dsttime;     /* type of dst correction to apply */
};
```

The timeval structure specifies a time value in seconds and microseconds. The values in timeval are opaque types whose length may vary on different machines; depending on

The timezone structure indicates om Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time a

Only the super-user may set the t er than 1 (see init(8)), the time may only be advanced. This limitation is imp y time stamps on files. The system time can still be adjusted backwards using

(there are lots of Unix system calls, which do lots of different things)

RETURN

A 0 return value indicates that the call succeeded. A -1 return value indicates an error occurred, and in this case an error code is stored into the global variable errno.

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
```

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
fawcett:330 childs$ g++ -O2 timings.C
fawcett:330 childs$ ./a.out
done executing, took 0.000000
fawcett:330 childs$
```

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    printf("Count was %d\n", count);  /* NEW LINE OF CODE */
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
```

gettimeofday example

```
fawcett:330 childs$ cat timings2.C
#include <sys/time.h>
#include <stdio.h>

int LoopFunction(int iteration, int &count)
{
    count += iteration;
}

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        LoopFunction(i, count);
    /* No longer need this: printf("Count was %d\n", count);  */
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
fawcett:330 childs$ g++ -O2 timings2.C
fawcett:330 childs$ ./a.out
done executing, took 0.213101
```

More performance analysis

- gprof: old program … I'm struggling to get it to work
- PAPI: library used widely to capture things like L1 cache misses, stalls, etc
- TAU: full performance analysis infrastructure
  - made right here at UO

Debugging

#1 complaint I hear from employers

- "students can't debug"
  - If you can debug, you can progress on their projects (even if slowly)
  - If not, they will have to hand hold you
- Think about your debugging approach.
  - How would you describe it during an interview?

This lecture describes how I would answer that question

Debugging Strategy

- (#1) Figure out _where_ the error is occurring
- (#2) Figure out _why_ the error is occurring
- (#3) Form a plan to fix the error

Terrible debugging strategy

- fix it by repeatedly making random changes
  - typically, code is pretty close to working in the first place
  - each random change creates a new problem that then has to be debugged
  - code ends up as a complete mess

This is a “bugging” strategy.

Always make sure you feel confident about what the problem is before you start changing code

Debugging as the scientific method

- Debugging involves the scientific method:
  - You have a hypothesis
    - ("the bug is because of this")
  - You form an experiment
    - ("I will test my theory by examining the state of some variables")
  - You confirm or deny the hypothesis
    - ("the state was OK, my hypothesis was wrong")
    - ("the state was bad, my hypothesis was right")

Backups

- The “scientific method” of debugging – which is good – can leave your code as a mess
- My recommendation:
  - when you have a bug, immediately make a copy of your program
  - apply the scientific method-style of debugging until you understand the problem and how to fix
  - then go back to your original program and make the fix there

Debugging Overview

- To me, effective debugging is about two things:
  - Challenging your own assumptions
  - Divide-and-conquer to find the problem

Challenging Assumptions

- you thought the code would work and it doesn't
  - so something you did is wrong, and you have to figure out what
- I find students are often turning under the wrong stones, since there are some stones they don't see
- the way to find these "hidden stones" is to start with the bad outcome and search backwards
  - why is this pointer NULL? why is this value bad?

Divide-and-Conquer

- There are lots of things that could be wrong with your program
  - This is a search problem!!
- Divide-and-Conquer: try to focus on hypotheses that will eliminate half the possibilities, no matter what the outcome is

Divide-and-Conquer

- "Halving" hypotheses:
  - The problem is in this module
  - The problem is in this function
- Non-halving hypotheses:
  - The problem is in this line of code
  - The problem is with this variable

As you divide the space smaller and smaller, you will eventually end up hypothesis that are small in scope

Good practice / Poor practice

- Good practice:
  - Write a few lines of code and test that it does what you want
- Poor practice:
  - Write a bunch of code and compile it all at the end

Why is it better to write smaller portions of code at a time?

Why is it better to write smaller portions of code at a time?

- If you have one bug
  - it is easier to figure out where that bug is
    - searching through tens of lines of code, not hundreds
- If you have many bugs
  - this is a disaster and you end up chasing your tail
    - and you are still searching through hundreds of lines of code, not tens

The extra effort of modifying the main program to test your new code pays off ten-fold (WAG)

Final thought: always initialize your variables!!

- Many crashes in your HW assignments due to uninitialized pointers.
  - If you assign it to NULL, then at least you know that is set to something not valid
  - Otherwise, you see an address and think it might have valid memory
- Initialize non-pointers too!
- Classic point that employers look for.

This practice becomes increasingly essential when you work in teams on large projects.

Debugging: Print Statements

Print statements

- Print statements (cerr, printf):
  - the world's most used debugger
  - very successful SW professionals are able to debug large SW projects using only print statements
- Pitfalls:
  - output buffering
  - too many print statements

Pitfall #1: output buffering

- output is sometimes buffered, and buffered output is dropped when a program crashes
- if you don't see a print statement
  - your program may have crashed before it got to that print statement
  - your program may have gotten to that print statement, but crashed before it got outputted

cerr: automatically flushes & mostly prevents this problem

```
cerr << "*(NULL) = " << *NULL << endl; // still doesn't work
```

Output buffering and cerr

- cerr: automatically flushes & mostly prevents output buffering problem
- Exception:
  - cerr << "*(NULL) = " << *NULL << endl;
    - (crashes before anything is printed out)
- Work-around:
  - cerr << "*NULL) = ";
  - cerr << *NULL << endl;

Pitfall #2: millions of print statements

```
void make_black(unsigned char *b, int width, int height,
                int buffer_size)
{
    for (int i = 0 ; i < width ; i++)
    {
        for (int j = 0 ; j < height ; j++)
        {
            int pixel_index  = j*width+i;
            int buffer_index = 3*pixel_index;
            cerr << "About to write to index"
                 << buffer_index << endl;
            b[buffer_index+0] = 0;
            b[buffer_index+1] = 0;
            b[buffer_index+2] = 0;
        }
    }
```

This will result in millions of print statements … hard to debug.

Reducing print statements

```
void make_black(unsigned char *b, int width, int height,
                int buffer_size)
{
    for (int i = 0 ; i < width ; i++)
    {
        for (int j = 0 ; j < height ; j++)
        {
            int pixel_index  = j*width+i;
            int buffer_index = 3*pixel_index;
            if (buffer_index < 0 || buffer_index >= buffer_size)
            {
                cerr << "About to write to index"
                     << buffer_index << endl;
                exit(EXIT_FAILURE);
            }
            b[buffer_index+0] = 0;
            b[buffer_index+1] = 0;
            b[buffer_index+2] = 0;
        }
    }
}
```

Make it easy on yourself…

```
128-223-223-73-wireless:330 hank$ cat big_print.C
#include <iostream>

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

#define PL cerr << "(PL): " << __FILE__ << ": " << __LINE__ << endl;

int main()
{
    PL
    int width = 100, height = 100;
    PL
    int buffer_size = width*height;
    PL
    unsigned char *b = new unsigned char[3*buffer_size];
    PL
    for (int i = 0 ; i < width ; i++)
    {
        PL
        for (int j = 0 ; j < height ; j++)
        {
```

```
128-223-223-73-wireless:330 hank$ ./a.out
(PL): big_print.C: 10
(PL): big_print.C: 12
(PL): big_print.C: 14
(PL): big_print.C: 16
(PL): big_print.C: 19
About to write to index10200
128-223-223-73-wireless:330 hank$
```

Debugging: Debuggers

Debuggers

- Allow you to set breakpoints
- Allow you to inspect variables
- Show you where a program crashed

Debuggers

- gdb:
  - GNU debugger
  - goes with gcc/g++
- lldb:
  - CLANG debugger

Debugging Symbols

- Object files:
  - by default, are compact and contain minimal information that connects to your original program
  - optionally, can be instructed to contain increased linkage
    - what line of code did these instructions come from?
    - what variable name is in this register?

You enable debugging symbols by adding "-g" to compile line "gcc –c file.C" → "gcc –g –c file.C"

Running with gdb

```
hank@ix: ~ 7$ cat myprogram.C
#include <stdlib.h>
int main()
{
    int *p = NULL;
    int X = *p;
}
hank@ix: ~ 8$ g++ -g myprogram.C
hank@ix: ~ 9$ gdb a.out
```

Running with gdb

```
(gdb) run
Starting program: /home/users/hank/a.out

Program received signal SIGSEGV, Segmentation fault.
0x0000000004004c4 in main () at myprogram.C:5
5               int X = *p;
(gdb) where
#0  0x0000000004004c4 in main () at myprogram.C:5
(gdb)
```

Arguments

- You are running "./proj3A 3A_input.pnm 3A_output.pnm"
- In gdb, you would do:

  % gdb proj3A

  (gdb) run 3A_input.pnm 3A_output.pnm

"core" files

- When a program crashes, it can create a "core" file
  - This file contains the state of memory when it crashes
  - It is very large, and can fill up filesystems
    - So system administrators often disable its generation
      - "ulimit –c 0" → "ulimit –c 10000000"
  - You can run a debugger on a program using a core file (tells you where it crashed)
    - gdb a.out core

Valgrind: a memory checker

```
hank@ix: ~ 14$ valgrind a.out
==13623== Memcheck, a memory error detector
==13623== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==13623== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==13623== Command: a.out
==13623==
==13623== Invalid read of size 4
==13623==    at 0x4004C4: main (myprogram.C:5)
==13623==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==13623==
==13623==
==13623== Process terminating with default action of signal 11 (SIGSEGV)
==13623==  Access not within mapped region at address 0x0
==13623==    at 0x4004C4: main (myprogram.C:5)
==13623==  If you believe this happened as a result of a stack
==13623==  overflow in your program's main thread (unlikely but
==13623==  possible), you can try to increase the size of the
==13623==  main thread stack using the --main-stacksize= flag.
==13623==  The main thread stack size used in this run was 8388608.
==13623==
==13623== HEAP SUMMARY:
==13623==     in use at exit: 0 bytes in 0 blocks
==13623==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==13623==
==13623== All heap blocks were freed -- no leaks are possible
==13623==
==13623== For counts of detected and suppressed errors, rerun with: -v
==13623== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
Segmentation fault (core dumped)
```

Valgrind and GDB

- Valgrind and gdb are available on ix
  - Older versions of Mac are possible, newer are not
  - Linux is easy
- You will have an assignment to have a memory error-free and memory leak-free program with valgrind.

Bonus Topics

Upcasting and Downcasting

- Upcast: treat an object as the base type
  - We do this all the time!
  - Treat a Rectangle as a Shape
- Downcast: treat a base type as its derived type
  - We don't do this one often
  - Treat a Shape as a Rectangle
    - You better know that Shape really is a Rectangle!!

Upcasting and Downcasting

```
class A
{
};

class B : public A
{
  public:
              B() { myInt = 5; };
    void      Printer(void) { cout << myInt << endl; };

  private:
    int       myInt;
};

void Downcaster(A *a)
{
    B *b = (B *) a;
    b->Printer();
}

int main()
{
    A a;
    B b;

    Downcaster(&b);  // no problem
    Downcaster(&a);  // no good
}
```

```
fawcett:330 childs$ g++ downcaster.C
fawcett:330 childs$ ./a.out
5
-1074118656
```

what do we get?

Upcasting and Downcasting

- C++ has a built in facility to assist with downcasting: dynamic_cast
- I personally haven't used it a lot, but it is used in practice
- Ties in to std::exception

Default Arguments

```
void Foo(int X, int Y = 2)
{
   cout << "X = " << X << ", Y = " << Y << endl;
}

int main()
{
    Foo(5);
    Foo(5, 4);
}

fawcett:330 childs$ g++ default.C
fawcett:330 childs$ ./a.out
X = 5, Y = 2
X = 5, Y = 4
```

default arguments: compiler pushes values on the stack for you if you choose not to enter them

Booleans

- New simple data type: bool (Boolean)
- New keywords: true and false

```
int main()
{
    bool b = true;
    cout << "Size of boolean is " << sizeof(bool) << endl;
}
fawcett:330 childs$ g++ Boolean.C
fawcett:330 childs$ ./a.out
```

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