Compiling and Bugs

PHYS2G03

McMaster University

Getting a working program

- Compiling
- Linking
- Bugs (errors)
- Fixing Bugs

Compiler Usage

Compile only: c++ file.cpp -c

Creates an object file: file.o

Link only: c++ file1.o file2.o –o exec

Combines 1 or more object files into an executable called exec

Compiler Usage

Compile and Link:

c++ file1.cpp file2.cpp –o myexec

Compiles 2 files and creates an executable called myexec

Not recommended – please use separate compiling and linking in your Makefile

Basic Makefile to Compile and Link

- Basic Makefile.
  To make myprogram just type: make

Also see the Unix handout

```
default: myprogram

myprogram: file1.o file2.o
	c++ file1.o file2.o –o myprogram

file1.o: file1.cpp
	c++ file1.cpp –c

file2.o: file2.cpp
	c++ file2.cpp -c
```

Compiler Usage: Flags

- Compilers have many options, e.g.

-c	**Create object file (COMPILE)**
-Wall	Warn about suspicious code
-O0	Don’t optimize
-O2	Default – mild optimization
-O3	Aggressive optimization
-g	Include debugging info
-I<dir>	Search this directory <dir> for files to include with #include

Linker Usage: Flags

Link Options

-o *name*	executable file called *name*
-L*<dir>*	search this directory for libraries
-l*library*	include this library

These are extra libraries – beyond basic stuff such as Fourier Transforms, parallel code or graphics, e.g.

c++ -o ploty ploty.o –lcpgplot -lpgplot -lX11

Photo # NH 96566-KN First Computer "Bug", 1945

92

Bugs

9/9

0800 Arctan started

1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct

13" sec (032) MP - MC ~~1.9826447000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10.00 test.

Relays changed

Relay 3145
Relay 3370

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

~~1645~~ 1630 Arctangent started.

1700 closed down.

The original bug

Bugs: Errors in Programs

1. Syntax Errors
2. Logic Errors (Bugs)
3. Linking Errors
4. Runtime Errors

Find the bugs…

I have put together a set of programs
that demonstrate the bugs I am talking about.
To start with get your own copy:

cp –r /home/2G03/bug ~/	
cd ~/bug	
ls	to see all the files
make *name*	to compile the program
name	run it
more *name*.cpp	Quick look
gedit *name*.cpp &	to look at the C++

Errors in Programs

1. Syntax Errors (Typos)
   The *dat* sat on the mat Compiler Finds it
   because dat is not recognized
2. Logic Errors (Bugs)
   The mat sat on the cat Compiler doesn't
   because the words are recognized but the outcome is not what you wanted

This is *why* variables must be declared
Otherwise the compiler thinks typing errors are just more undeclared variables

Bugs: Syntax

make syntax (syntax.cpp)

```
int man()

{

  / This program has a lot of issues

  std::cin >> a;

  std::cout << a;

}
```

Bugs: Syntax
compile syntax: Error messages

```
[wadsley@phys-ugrad ~/bug]$ make syntax
c++ -g -O0 syntax.cpp -c
syntax.cpp: In function ‘int man()’:
syntax.cpp:6:3: error: expected primary-expression before ‘/’ token
  / This program has a lot of issues
  ^
syntax.cpp:6:5: error: ‘This’ was not declared in this scope
  / This program has a lot of issues
    ^
syntax.cpp:6:10: error: expected ‘;’ before ‘program’
  / This program has a lot of issues
        ^
syntax.cpp:14:3: error: ‘cout’ is not a member of ‘std’
  std::cout << a;
  ^
syntax.cpp:14:16: error: ‘a’ was not declared in this scope
  std::cout << a;
             ^
make: *** [syntax.o] Error 1
```

Bugs: Syntax
compile syntax: Error messages

General rules on fixing bugs

Compilers start at the top

1) Look at the first bug first. The errors indicate the line and the character. Use an editor that tells you line numbers (e.g. see gedit preferences)

2) Don’t be discouraged by lots of error messages. Some types of errors (e.g. leaving off a “ } or ;) or forgetting to declare a variable can create many messages.

Bugs: Include

make include (include.cpp)

```
#include <iosteam>
int main()
{
  std::cout << "Hello world!\n";
}
```

A common generic typo is mistyping the name of an include file:

include.cpp:1:19: fatal error: iosteam: No such file or directory

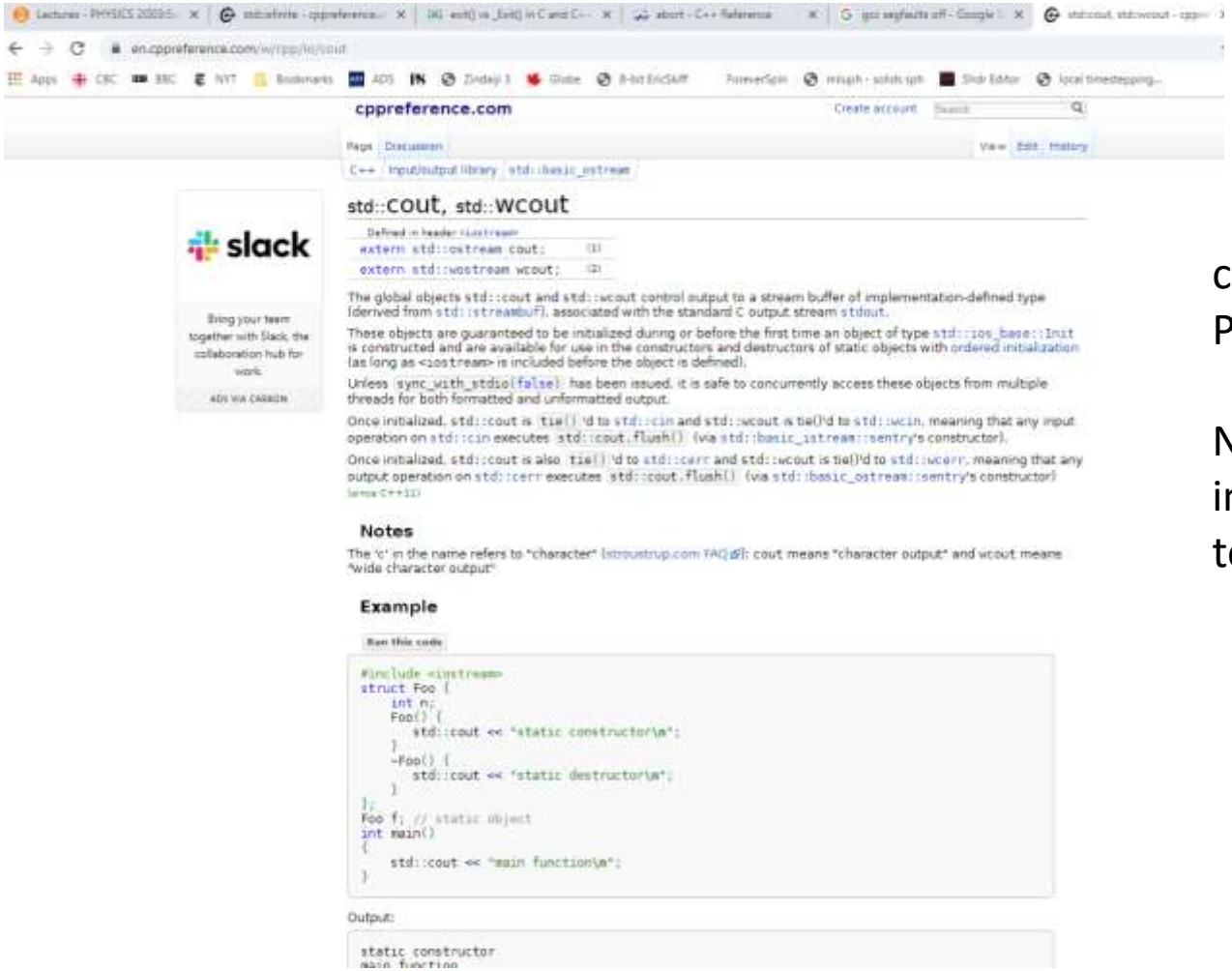
```
 #include <iosteam>
                  ^
```

Bugs: Include issues
compile syntax: Error messages

```
[wadsley@phys-ugrad ~/bug]$ make syntax
c++ -g -O0 syntax.cpp -c
syntax.cpp: In function 'int man()':
syntax.cpp:6:3: error: expected primary-expression before '/' token
  / This program has a lot of issues
  ^
syntax.cpp:6:5: error: 'This' was not declared in this scope
  / This program has a lot of issues
    ^
syntax.cpp:6:10: error: expected ';' before 'program'
  / This program has a lot of issues
        ^
syntax.cpp:14:3: error: 'cout' is not a member of 'std'
  std::cout << a;
  ^
syntax.cpp:14:16: error: 'a' was not declared in this scope
  std::cout << a;
            ^
make: *** [syntax.o] Error 1
```

These are also related to include problems. The compiler hasn't heard of cout
It needs #include <iostream>

Google c++ std::cout



cppreference . com
Page on std::cout

Note: it tells you what include is needed at the top!

Google c++ std::cout

cppreference . com
Page on std::cout

Note: it tells you which include header file is needed at the top!

Linker Errors

If you try to use a function that doesn't exist the Linker will give an error when you make the executable

e.g. undefined reference to "…"

Bug: Linking Error

make linkererror (linkererror.cpp)

```
int bogus_function(float);

int main()
{
  int i;

  i = bogus_function(1.0);
}
```

prototype

Bug: Linking Error

make linkererror (linkererror.cpp)

```
wadsley@phys-ugrad ~/bug]$ make linkererror
c++ -g -O0 linkererror.cpp -c
c++ -g -O0 linkererror.o -o linkererror
linkererror.o: In function `main':
/home/wadsley/bug/linkererror.cpp:8: undefined
    reference to `bogus_function(float)'
collect2: error: ld returned 1 exit status
make: *** [linkererror] Error 1
```

Compile works

Linker fails

Bug: Lack of Prototype

make prototype (prototype.cpp)

```
int main()
{
  int i;

  i = bogus_function(1.0);
}
```

Bug: Lack of Prototype

make prototype (prototype.cpp)

```
[wadsley@phys-ugrad ~/bug]$ make prototype
c++ -g -O0 prototype.cpp -c
prototype.cpp: In function ‘int main()’:
prototype.cpp:9:25: error: ‘bogus_function’ was not declared in this scope
  i = bogus_function(1.0);
                    ^
make: *** [prototype.o] Error 1
```

C++ complains at compile time if there is no prototype, it is not willing to guess

(Unfortunately C is willing to guess, the C++ standard is better here)

Real bugs

- Compile time problems are annoying but with the detailed line info and experience you will resolve them
- The hard bugs to fix are those that occur when the program compiles and runs, but gives the wrong answer

Bug: Logic Error, (a real bug)
make logic (logic.cpp)

```
#include <iostream>
int main()

{
  // This program prints the sum of two numbers

  float a,b,c;

  std::cout << "Enter two numbers: \n";

  std::cin >> a >> b;

  std::cout << "The sum of the two numbers is: " << b+c << "\n";
}
```

Compilers sometimes give useful **warnings** about this (even though they will still compile)

Bug: Logic Error, (a real bug)
make logic (logic.cpp)

```
wadsley@phys-ugrad ~/bug]$ make logic
c++ -g -O0 logic.cpp -c
c++ -g -O0 logic.o -o logic
[wadsley@phys-ugrad ~/bug]$ logic
Enter two numbers:
1 2
The sum of the two numbers is: 2
```

Compilers don’t always warn about bugs

Bug: Logic Error, (a real bug)
make logic (logic.cpp)

```
#include <iostream>
int main()
{
  // This program prints the sum of two numbers
  float a,b,c;

  std::cout << "Enter two numbers: \n";

  std::cin >> a >> b;

  std::cout << "The sum of the two numbers is: " << b+c << "\n";
}
```

Wrong things added!

Note: This bug may seem obvious but it is <u>the most common kind of bug</u> which wastes the most time for 2G03 students

Bug: Logic Error, (a real bug)
make logic (logic.cpp)

```
[wadsley@phys-ugrad ~/bug]$ c++ -Wall logic.cpp -c
logic.cpp: In function ‘int main()’:
logic.cpp:15:55: warning: ‘c’ may be used uninitialized in this
    function [-Wmaybe-uninitialized]
  std::cout << "The sum of the two numbers is: " << b+c << "\n";
                                                        ^
[wadsley@phys-ugrad ~/bug]$
```

Compilers can give useful **warnings** about this

-Wall is useful to make sure it does “Warn All”

(Note: for warnings, it will still compile – **read the warning!**)

Bug: Logic Error, (a real bug)
make logic (logic.cpp)

```
#include <iostream>
int main()
{
  // This program prints the sum of two numbers
  float a,b,c;

  std::cout << "Enter two numbers: \n";

  std::cin >> a >> b;

  std::cout << "The sum of the two numbers is: " << b+c << "\n";
  std::cout << “Debug: “ << b << c << “\n”;
}
```

If the results are different to what is expected – print the parts and make sure they make sense. **Print debugging works!**

Runtime Errors:
Math Errors (Floating Point Exceptions)

1. Math errors – e.g. Infinities and Non-numbers

   1/0

   1.0/0.0

   sqrt(-1.0)

The usual default for intel compilers is to ignore most math errors. This is the IEEE (Institute of Electrical and Electronics Engineers) standard behaviour for compilers. Even though **computing hardware detects these errors**, they decided to ignore it by default. Some language compilers, e.g. Fortran let you detect these errors. (Compile with ifort –fpe0)

Bugs: Math Exception

make mathbug (mathbug.cpp)

```
#include <iostream>

int main()
{
  float a,b;

  b = 0.0;

  std::cout << "b=" << b << "\n";

  a = 1.0/b;

  std::cout << "a=" << a << "\n";
}
```

Bugs: Math Exception

make mathbug (mathbug.cpp)

```
[wadsley@phys-ugrad ~/bug]$ make mathbug
c++ -g -O0 mathbug.cpp -c
c++ -g -O0 mathbug.o -o mathbug
[wadsley@phys-ugrad ~/bug]$ mathbug
b=0
a=inf
[wadsley@phys-ugrad ~/bug]$
```

Bugs: Math Exception (mathbugdetect.cpp)

```
#include <iostream>
#include <cmath>
#include <cstdlib>

int main()
{
  float a,b;

  b = 0.0;
  if (!std::isfinite(b)) {
    std::cout << "There is something wrong with b=" << b << "\n";
    exit(1);
  }

  std::cout << "b=" << b << "\n";

  a = 1.0/b;
  if (!std::isfinite(a)) {
    std::cout << "There is something wrong with a=" << a << "\n";
    exit(1);
  }

  std::cout << "a=" << a << "\n";
}
```

C/C++ provides functions to detect bad values (e.g. isfinite, isNaN)
It requires extra effort on behalf of the programmer for every single math operation

Runtime Errors:
Math Errors

- Ideally the process will crash with an error like a FLOATING EXCEPTION or

  DIVIDE BY ZERO
- Linux provides functions to turn on the error detection
- Otherwise bad values will quietly propagate through your program. In scientific computing you want to know as soon as bad values occur.

How to turn on floating exceptions

```
#include <fenv.h>

int main()
{
  /* At startup all exceptions are masked (turned off).
     Enable some exceptions  */

  feenableexcept (FE_INVALID|FE_DIVBYZERO|FE_OVERFLOW);
}
```

How to turn on floating exceptions

- We have provided a simple library –ltrapfpe that turns on floating exceptions at link time

Link with –ltrapfpe

e.g. c++ –ltrapfpe mathbug.o –o mathbugfpe

This is particularly useful when debugging – with a debugger it stops the program at the first error so you can instantly check the values that led to the issue

Bugs: Math Exception

make mathbug (mathbug.cpp)

```
#include <iostream>

int main()
{
  float a,b;

  b = 0.0;

  std::cout << "b=" << b << "\n";

  a = 1.0/b;

  std::cout << "a=" << a << "\n";
}
```

Runtime Errors: Math Errors

- Try mathbug

  2 flavours:

make mathbug

mathbug

make mathbugfpe

mathbugfpe

Default.
Ignore bad math

–ltrapfpe
Crash on bad math

Runtime Errors: Math Errors

make mathbugfpe

mathbugfpe ←

–ltrapfpe
Crash on bad math

Real applications (including your 2G03 projects), calculate millions of math operations – you cannot print them all!

It's very useful to have the program stop the first time it finds an issue

Bugs: Math Exception

make mathbugfpe (mathbug.cpp)

```
[wadsley@phys-ugrad ~/bug]$ mathbug
b=0
a=inf

[wadsley@phys-ugrad ~/bug]$ make mathbugfpe
c++ -ltrapfpe -lm -g -O0 mathbug.o -o mathbugfpe
[wadsley@phys-ugrad ~/bug]$ mathbugfpe
b=0
Floating exception
```

Not very informative!

Bugs: Math Exception with gdb

make mathbugfpe (mathbug.cpp)

```
wadsley@phys-ugrad ~/bug]$ gdb mathbugfpe
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-114.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /1/home/wadsley/bug/mathbugfpe...done.
(gdb) r
Starting program: /1/home/wadsley/bug/mathbugfpe
b=0

Program received signal SIGFPE, Arithmetic exception.
0x00000000004007fb in main () at mathbug.cpp:11
11    a = 1.0/b;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-260.el7_6.6.x86_64 libgcc-
    4.8.5-36.el7_6.2.x86_64 libstdc++-4.8.5-36.el7_6.2.x86_64
(gdb)
```

Debugger: Exact line!

Debugger: gdb

- All large computational science groups use debuggers and real time floating point error catching
- Anything else is hopelessly inefficient for a large program
- Note: you must compile with –g for the debugger to work properly (more later)

Runtime Errors: Memory Errors

- Memory usage errors:

  Accessing the wrong memory or data, such as memory that doesn't belong to your process

Typical Causes:

1. Incorrect modification of an array

   e.g. A[-1] = 0.0

2. Incorrect call to functions

   e.g. Too many arguments

Bug: Memory Error

make memoryerror (memoryerror.cpp)

```
#include <iostream>

int main()
{
  float a[4], b[4], c[4]; // vectors 4 long

  a[0] = 0.0;
  b[0] = 0.0;
  c[0] = 0.0;

  a[4] = 1;
  b[4] = 2;
  c[4] = 3;
  std::cout << "Now a[0] should be 0.0 but instead a[0] = " << a[0] << "\n";
  std::cout << "Now b[0] should be 0.0 but instead b[0] = " << b[0] << "\n";
  std::cout << "Now c[0] should be 0.0 but instead c[0] = " << c[0] << "\n";
}
```

In general errors like this are NOT detected by compilers!

Bug: Memory Error

make memoryerror (memoryerror.cpp)

```
[wadsley@phys-ugrad ~/bug]$ make memoryerror
c++ -g -O0 memoryerror.cpp -c
c++ -g -O0 memoryerror.o -o memoryerror
[wadsley@phys-ugrad ~/bug]$ memoryerror
Now a[0] should be 0.0 but instead a[0] = 2
Now b[0] should be 0.0 but instead b[0] = 3
Now c[0] should be 0.0 but instead c[0] = 0
```

If you accidentally change memory that you own the cpu/runtime system does not detect it!

In this case we are abusing the array bounds for arrays a, b and c.

In C an array of N things is indexed from 0 to N-1 (0 to 3 here)

Bug: Memory Error

make memoryerror (memoryerror.cpp)

```
#include <iostream>

int main()
{
  float a[4], b[4], c[4]; // vectors 4 long

  a[0] = 0.0;
  b[0] = 0.0;
  c[0] = 0.0;

  a[4] = 1;
  b[4] = 2;
  c[4] = 3;
  std::cout << "Now a[0] should be 0.0 but instead a[0] = " << a[0] << "\n";
  std::cout << "Now b[0] should be 0.0 but instead b[0] = " << b[0] << "\n";
  std::cout << "Now c[0] should be 0.0 but instead c[0] = " << c[0] << "\n";
}
```

In C/C++ indexes on vectors start at 0
So valid indices are 0,1,2,3 on a,b or c
However, like FPE, the program by default does not check if an index is valid

Runtime Errors:
Memory Errors

- Ideally the process will crash with a SEGMENTATION FAULT
- However this typically requires trying to change data outside the memory owned by your process
- If it doesn’t crash, you may be writing values randomly into your own data

Bug: Segmentation Fault
make segfault (segfault.cpp)

```
int main()
{
  float a[20];
  int i;

  i = 123456789;

  a[i] = 0.0;
}
```

Bug: Segmentation Fault
make segfault (segfault.cpp)

```
[wadsley@phys-ugrad ~/bug]$ segfault
Segmentation fault


[wadsley@phys-ugrad ~/bug]$ gdb segfault
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-114.el7
...

(gdb) r
Starting program: /1/home/wadsley/bug/segfault

Program received signal SIGBUS, Bus error.
0x0000000000400514 in main () at segfault.cpp:8
8       a[i] = 0.0;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-260.el7_6.6.x86_64 libgcc-
    4.8.5-36.el7_6.2.x86_64 libstdc++-4.8.5-36.el7_6.2.x86_64
(gdb)
```

Not very informative!

Debugger: Exact line

Runtime Errors:
Memory Errors

- Note: the compiler doesn't detect this error at all
- The OS detects it – when it runs
- The program tries to write outside its legal memory segment (i.e. to memory you don't own)

  SEGMENTATION FAULT

Symbols for debugging -g

-g includes symbols in your .o file

This lets the debugger know where in the program you are if it stops

```
[wadsley@phys-ugrad ~/bug]$ c++ -O0 segfault.cpp -c
[wadsley@phys-ugrad ~/bug]$ ls -alt segfault.o
-rw-rw-r-- 1 wadsley wadsley 1456 Sep 15 22:02 segfault.o

[wadsley@phys-ugrad ~/bug]$ c++ -g segfault.cpp -c
[wadsley@phys-ugrad ~/bug]$ ls -alt segfault.o
-rw-rw-r-- 1 wadsley wadsley 3008 Sep 15 22:02 segfault.o
```

Bug: Segmentation Fault
make segfault (segfault.cpp)

```
[wadsley@phys-ugrad ~/bug]$ c++ -O0 segfault.cpp –o segfault
```

No –g, no symbols

```
[wadsley@phys-ugrad ~/bug]$ gdb segfault
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-114.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /1/home/wadsley/bug/segfault...(no deb
(gdb) r
Starting program: /1/home/wadsley/bug/segfault

Program received signal SIGBUS, Bus error.
0x0000000000400514 in main ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-2
      36.el7_6.2.x86_64 libstdc++-4.8.5-36.el7_6.2.x86_64
(gdb)
```

Debugger: no idea
what line was
Only name of function
where it happened.
main()

Errors and Optimization

- Sometimes the compiler glosses over problems or creates problems due to over aggressive code **optimization**
- ***Optimizing can remove code the compiler thinks is pointless, including variables!***
- You can turn optimization off

  c++ –O0 No optimization

  c++ –O0 -g No optimization and debug info included

  (That is: minus Oh Zero)

Bug: Optimization problems

```
#include <iostream>
main() {
  float a,b,c;
  a=0;  c=2;   b=c/a;
  std::cout << "b=" << b << "\n";
}
[wadsley@phys-ugrad ~/bug]$ c++ -O3 -ltrapfpe -g mathbug2.cpp -o mb2
[wadsley@phys-ugrad ~/bug]$ gdb mb2
(gdb) r
Starting program: /1/home/wadsley/bug/mb2

Program received signal SIGFPE, Arithmetic exception.
0x00000000004006f6 in main () at mathbug2.cpp:4
4       a=0;  c=2;   b=c/a;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-260.el7_6.6.x86_64 libgcc-
      4.8.5-36.el7_6.2.x86_64 libstdc++-4.8.5-36.el7_6.2.x86_64
(gdb) print b
$1 = <optimized out>
(gdb)
```

With –O3 the compiler decided b was superfluous!

Bug: Optimization problems

```
#include <iostream>
main() {
  float a,b,c;
  a=0;  c=2;   b=c/a;
  std::cout << "b=" << b << "\n";
}

#include <iostream>
main() {
  float a,c;
  a=0;  c=2;
  std::cout << "b=" << c/a << "\n";
}
```

With –O3 (optimization level 3), the compiler rewrote program something like this

-O0 avoids this problem!