Making things go right
Fixing things that go wrong

Making things go right

- Defensive programming : *aim for the pit of success*
  - Creating abstractions : *manage and protect state*
  - Adding assertions : *what are you assuming is true?*
  - Creating test-benches : *what is the correct result?*
  - Incremental changes : *don't break existing functions*

- Other techniques
  - Compile with warnings turned on: ***fix*** compiler warnings
  - Specify your functions in the abstract
  - Develop high-level algorithms
  - Separate complex processes into smaller functions

Specifying functions : sorting

"Sort a range of integers"

```
void sort(int *begin, int *end);
void sort(vector<int> &begin);
vector<int> sort(const vector<int> &begin);
```

Specifying functions : sorting

"Sort a range of integers"

```
void sort(int *begin, int *end);
void sort(vector<int> &begin);
vector<int> sort(const vector<int> &begin);
```

Specifying functions : sorting

"Sort a range of integers"

```
bool is_sorted(const int *begin, const int *end);

// Pre:  begin <= end
// Post: is_sorted(begin',end')
void sort(int *begin, int *end)
{
    assert(begin <= end);

    // Something

    assert(is_sorted(begin, end));
}
```

Specifying functions : sorting

"Sort a range of integers"

```
bool is_sorted(const int *begin, const int *end);

// Pre:  begin <= end
// Post: is_sorted(begin',end')
void sort(int *begin, int *end)
{
    assert(begin <= end); // Pre-condition

    // Something

    assert(is_sorted(begin, end)); // Post-condition
}
```

Specifying functions : induction

```
// Pre:  is_sorted(begin,end-1)
// Post: is_sorted(begin',end') && end[-1] in (begin',end')
void insert(int *begin, int *end);

void sort(int *begin, int *end)
{
    assert(begin <= end); // Pre-condition

    if(begin==end){
        // An empty list is sorted
    }else{
        sort(begin, end-1); // Sort range [begin,end-1)
        insert(begin, end); // Produce sorted [begin,end)
    }

    assert(is_sorted(begin, end)); // Post-condition
}
```

Specifying functions : induction

```
// Pre:  is_sorted(begin,end-1)
// Post: is_sorted(begin',end') && end[-1] in (begin',end')
void insert(int *begin, int *end);

void sort(int *begin, int *end)
{
    assert(begin <= end); // Pre-condition

    if(begin==end){
        // An empty list is sorted
    }else{
        sort(begin, end-1); // Sort range [begin,end-1)
        insert(begin, end); // Produce sorted [begin,end)
    }

    assert(is_sorted(begin, end)); // Post-condition
}
```

Specifying functions : induction

```
// Insert end[-1] into sorted range[begin,end-1)
// Pre:  is_sorted(begin,end-1)
// Post: is_sorted(begin',end') && end[-1] in (begin',end')
void insert(int *begin, int *end)
{
    assert( is_sorted(begin,end-1) );

    int i=begin-end-1;
    while( 0 < i){
        if( begin[i-1] > begin[i] ){
            swap( begin[i-1], begin[i] );
        }
        i=i-1;
    }

    assert( is_sorted(begin,end) );
}
```

Is this code correct?

Specifying functions : testing

```
// Insert end[-1] into sorted range[begin,end-1)
// Pre:  is_sorted(begin,end-1)
// Post: is_sorted(begin',end') && end[-1] in (begin',end')
void insert(int *begin, int *end)
{
    assert( is_sorted(begin,end-1) );

    int i=begin-end-1;
    while( 0 < i){
        if( begin[i-1] > begin[i] ){
            swap( begin[i-1], begin[i] );
        }
        i=i-1;
    }

    assert( is_sorted(begin,end) );
}
```

Is this code correct?

What inputs to test?

- Designing test cases is a bit of an art
  - Think of special cases
  - Think of edge cases
  - Include stress tests
- Test cases for sorting a vector
  - Empty vector : []
  - Single element vector : [a]
  - Two element vector
    - Ordered : [a,b], a<b
    - Un-ordered: [a,b], a>b
    - Equal [a,b], a==b
  - Three element vector
    - All combinations of ordered, un-ordered, and equal?
  - Random vectors

Pre and post-conditions in practise

- Pre- and post-conditions are extremely useful
  - Documentation: what is the function supposed to do?
  - Testing: what are valid inputs and outputs?
  - Debugging: when does the function fail?

- Asserting pre/post-conditions *may* be useful
  - Are the conditions cheap or expensive?
    - Is checking the condition slower than the function?
  - Assert the cheap stuff: is this pointer non-null?
  - Comment out the expensive stuff: is this array sorted?
    - But leave it in as documentation and for future debugging

Fixing things that go wrong

- Defensive programming
  - Enforce abstractions : *check internal state is valid*
  - Adding more assertions : *how early does problem occur?*
  - Extend test-benches : *what is the simplest failing case?*

- Other techniques
  - Logging via stderr
  - Debugging
  - Exceptions

"My program doesn't work"

- Programs can "not work" in many different ways
- In what way is it not working?
  - **Compilation**: the source code is malformed
  - **Linking**: definitions are missing or repeated
  - **Crashing**: program does not complete execution
  - **Correctness**: program executes, but output is wrong

Running the wrong program

"*I keep changing the source code,*
but the program still fails in the same way"

1. Is compilation actually completing successfully?
2. Are you running the program/function you just compiled?
3. Is the code you are editing actually getting compiled?
4. Are you in the right directory?

Suggestion: make the program print something unique

```
int main(int argc, char *argv[])
{
    cerr << "Debug session 1456" << endl;

    assert(argc > 1);
    int x = atoi(argv[1]);
```

Dealing with compiler errors

- C++ produces incredibly long error messages
  - They are notoriously difficult to comprehend
  - A single error can result in hundreds of lines of error
  - Templates mean there is an explosion of `::` and `<>`
- **Tip 1** : fix the ***first*** error message first
  - *Ignore the last error*; it could be caused by the first one
  - Scroll up to the very first error message
- **Tip 2** : ***read*** the error messages (with eyes+brain)
  - The compiler is trying to give you all the information it can
  - Look at all the source locations it identifies very carefully
- **Tip 3** : google the error messages
  - Only do this after trying to solve the problem yourself
  - Actually read the explanation for what went wrong
  - Beware of "cargo-cult" programming

Compiling versus linking

- A program is compiled in two main phases:
  1. **Compiling**: turn source files (.cpp) into object files (.o)
  2. **Linking**: link object files (.o) together into one program
- Object file contains the "machine-code" for a source file
  - Meta-data about the declarations and definitions
  - CPU-specific binary instructions for the defined functions
  - Source file -> object file : easy; performed by the compiler
  - Object file -> source file : hard; no general solution
- Compilers will both compile and link by default
  - Compile and link: `g++ fileA.cpp fileB.o –o prog`
  - Compile to object file only: `g++ -c fileA.cpp –o fileA.o`
  - Link in existing object file: `g++ fileA.o fileB.cpp –o prog`

Dealing with linker errors

- Linker errors are generally easier to deal with
  - Linking is just about matching decls. and defns.

1. "`undefined reference`": declared but not defined
   - Something is declared somewhere and used
   - That thing is never defined in any source or object file
     1. You forgot to add a source/object file while compiling; or
     2. You forgot to add a definition; or
     3. The definition does not match the declaration
2. "`multiple definition`" : something is defined twice
   - You have compiled in the same source file twice
   - Multiple headers provide a definition of a function
     - Split function up: declaration in header; definition in one source

Crashes: "program go boom"

- A "crash" is a loose term for lots of things
  - The program did not exit in a controlled way; and/or
  - Some sort of exception or run-time error happened
  - It may not be obvious that the program has crashed

- We have four broad classes of crash
  1. ***Stack overflow***: a function recurses too many times
  2. ***Memory corruption***: reading or writing via invalid pointers
  3. ***Undefined behavior:*** stepping outside the rules of C++
  4. ***Unhandled exceptions***: C++ mechanism to indicate errors

Crashes : using debuggers

- A debugger allows you to explore program execution
  - *Where did the program crash?*
  - *What functions are on the stack?*
  - *What are the values of local variables?*
- There are many different debuggers available
  - Command-line: gdb (unix/linux), lldb (OSX)
  - GUI: Visual Studio, XCode, CLion, ...
  - Debuggers are very powerful and can be complicated
- *Recommendation*: use command-line gdb
  - Always available in Linux
  - Can answer the key "where" and "what" questions

Stack overflow : the problem

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ fib.cpp -o fib

dt10@LAPTOP-0DEHDEQ0:~
$ ./fib
Segmentation fault (core dumped)

dt10@LAPTOP-0DEHDEQ0:~
$
```

Stack overflow : adding debug info

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g fib.cpp -o fib

dt10@LAPTOP-0DEHDEQ0:~
$
```

Stack overflow : running in gdb

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g fib.cpp -o fib

dt10@LAPTOP-0DEHDEQ0:~
$ gdb ./fib
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git
...
Reading symbols from ./fib...done.
(gdb)
```

Stack overflow : running in gdb

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g fib.cpp -o fib

dt10@LAPTOP-0DEHDEQ0:~
$ gdb ./fib
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git
...
Reading symbols from ./fib...done.
(gdb) run
```

Stack overflow : running in gdb

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g fib.cpp -o fib

dt10@LAPTOP-0DEHDEQ0:~
$ gdb ./fib
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git
...
Reading symbols from ./fib...done.
(gdb) run
Starting program: ~/fib

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000603 in f (...) at
fib.cpp:2
2       {
```

Stack overflow : viewing the stack

```
int f(int n)
{
  if(n==0)
    return 1;
  return f(n-1)+f(n-2);
}

int main()
{
  return f(10);
}
```

```
(gdb) run
Starting program: ~/fib

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000603 in f (...) at
fib.cpp:2
2  {

(gdb) bt
#0  0x0000000008000603 in f (n=<error>)
at fib.cpp:2
#1  0x0000000008000620 in f (n=-174672)
at fib.cpp:6
#2  0x0000000008000620 in f (n=-174671)
at fib.cpp:6
#3  0x0000000008000620 in f (n=-174670)
at fib.cpp:6
#4  0x0000000008000620 in f (n=-174669)
```

Corruption: the problem

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb 0 Hello
H

dt10@LAPTOP-0DEHDEQ0:~
$ gdb 1 Hello
e

dt10@LAPTOP-0DEHDEQ0:~
$ gdb 100000 Hello
Segmentation fault (core dumped)

dt10@LAPTOP-0DEHDEQ0:~
$
```

Corruption: adding debug info

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$
```

Corruption: running in gdb

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb --args ./a.out 10000 Hello
```

Corruption: running in gdb

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb --args ./a.out 10000 Hello
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git

(gdb)
```

Corruption: running in gdb

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb --args ./a.out 10000 Hello
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git

(gdb) run
Starting program: ~/a.out 10000 Hello

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000862 in f (off=10000,
p=0x7fffffffee451 "Hello") at tmp.cpp:7
7         return p[off];
(gdb)
```

Corruption: running in gdb

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb --args ./a.out 10000 Hello
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git

(gdb) run
Starting program: ~/a.out 10000 Hello

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000862 in f (off=10000,
p=0x7fffffffee451 "Hello") at tmp.cpp:7
7         return p[off];
(gdb)
```

Corruption: running in gdb

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
dt10@LAPTOP-0DEHDEQ0:~
$ g++ -g tmp.cpp

dt10@LAPTOP-0DEHDEQ0:~
$ gdb --args ./a.out 10000 Hello
GNU gdb (Ubuntu 8.1-0ubuntu3)
8.1.0.20180409-git

(gdb) run
Starting program: ~/a.out 10000 Hello

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000862 in f (off=10000,
p=0x7fffffffee451 "Hello") at tmp.cpp:7
7         return p[off];
(gdb)
```

Corruption: viewing the stack

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) run
Starting program: ~/a.out 10000 Hello

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000862 in f (off=10000,
p=0x7fffffffee451 "Hello") at tmp.cpp:7
7           return p[off];
(gdb)
```

Corruption: viewing the stack

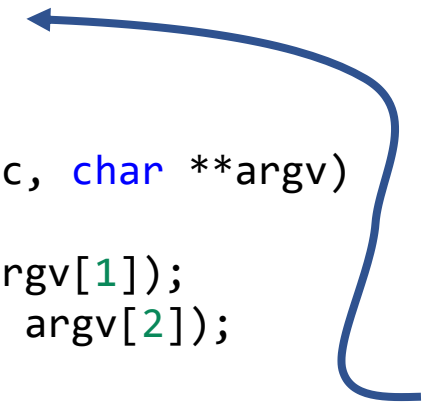
```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) run
Starting program: ~/a.out 10000 Hello

Program received signal SIGSEGV,
Segmentation fault.
0x0000000008000862 inf (off=10000,
p=0x7fffffee451 "Hello") at tmp.cpp:7
7          return p[off];
(gdb) bt

#0  0x0000000008000862 in
f (off=10000, p=0x7ffffee451 "Hello")
at tmp.cpp:7

#1  0x00000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13
```

Corruption: viewing variables

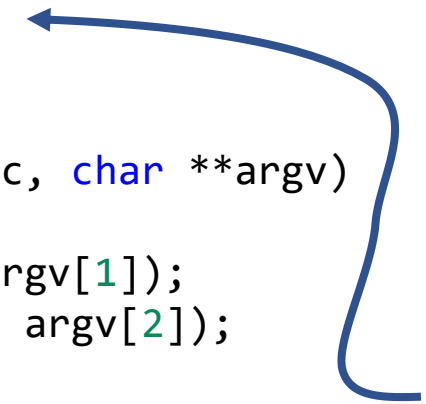
```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) bt

#0  0x0000000008000862 in
f (off=10000, p=0x7ffffee451 "Hello")
at tmp.cpp:7

#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13

(gdb)
```



Corruption: viewing variables

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```



```
(gdb) bt

#0  0x0000000008000862 in
f (off=10000, p=0x7ffffee451 "Hello")
at tmp.cpp:7

#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13

(gdb) print off
$1 = 10000

(gdb) print p
$2 = 0x7fffffee451 "Hello"

(gdb)
```

Corruption: moving up the stack

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) bt

#0  0x0000000008000862 in
f (off=10000, p=0x7ffffee451 "Hello")
at tmp.cpp:7

#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13

(gdb) up
#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13
13          cout << f(off, argv[2]);

(gdb)
```

Corruption: viewing variables

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) up
#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13
13          cout << f(off, argv[2]);

(gdb)
```

Corruption: viewing variables

```
char f(int off, char *p)
{
  return p[off];
}

int main(int argc, char **argv)
{
  int off=atoi(argv[1]);
  cout << f(off, argv[2]);
}
```

```
(gdb) up
#1  0x0000000080008a4 in
main (argc=3, argv=0x7ffffee1c8)
at tmp.cpp:13
13          cout << f(off, argv[2]);

(gdb) print off
print off
$3 = 10000

(gdb) print argv[2]
$4 = 0x7ffffee451 "Hello"

(gdb)
```

Using a debugger

- Gdb is good for working out "where" and "what"
  - `run` : where did the program fail?
  - `bt` : (back-trace) what functions were on the stack?
  - `print` : what are the current values of variables?
  - `up` : moving to an outer function call
- Debuggers will not give you the "why"
  - It's up to you to work out what the actual root cause is
- Debuggers are usually a last resort
  - You've already thought about the code
  - You've already added assertions to check assumptions
  - You've already tried adding logging with cerr

Undefined behavior

- C++ allows you to shoot yourself in the foot
  - Not all behavior is specified
  - Some behavior is explicitly undefined
  - This allows compilers to create faster programs
- Some undefined behaviour *may* crash
  - Dereferencing a null pointer
  - Accessing memory after it is freed
  - Not returning a value from a function
- Some undefined behaviour *may not* crash
  - Dereferencing a null pointer
  - Accessing memory after it is freed
  - Not returning a value from a function

Undefined behavior : avoidance

- The compiler can *try* to help you
  - Compile with warnings turned on : `g++ -W -Wall`
  - Use run-time sanitisers: `g++ -fsanitize=undefined`

- There is no guaranteed method here
  - Create multiple test inputs
  - Examine every compiler warning
  - Try to diagnose and understand errors; don't use patch over