An F-16 fighter plane on the northern hemisphere. Why the northern hemisphere, you ask?

Because this is what an F-16 on the southern hemisphere would look like. (BTW, interesting effect if you drop a bomb :-)

From risks.digest, volume 3, issue 44:
From risks.digest, volume 3, issue 44:

- One of the first things the Air Force test pilots tried on an early F-16 was to tell the computer to raise the landing gear while standing still on the runway. Guess what happened? Scratch one F-16. (my friend

Retrieved by a technician from the Harvard Mark II machine on September 9, 1947. Now on display at the Smithsonian, Washington
Facts on Debugging

- Software bugs cost ~60 bln US$/yr in US
- Improvements could reduce cost by 30%
- Validation (including debugging) can easily take up to 50-75% of the development time
- When debugging, some people are three times as efficient than others

A Sample Program

$ sample 9 8 7
Output: 7 8 9

$ sample 11 14
Output: 0 11
How to Debug
(Sommerville 2004)

Locate error → Design → Repair error → Re-test program

The Traffic Principle
- Track the problem
- Reproduce
- Automate
- Find Origins
- Focus
- Isolate
- Correct

The Traffic Principle
- Track the problem
- Reproduce
- Automate
- Find Origins
- Focus
- Isolate
- Correct
1. The programmer creates a defect – an error in the code.
2. When executed, the defect creates an infection – an error in the state.
3. The infection propagates.
4. The infection causes a failure.

This infection chain must be traced back – and broken.

The Curse of Testing

• Not every defect causes a failure!
• Testing can only show the presence of errors – not their absence. (Dijkstra 1972)

Debugging

• Every failure can be traced back to some infection, and every infection is caused by some defect.
• Debugging means to relate a given failure to the defect – and to remove the defect.
The defect must be searched in \_\textit{space}\_ and \_\textit{time}\_.

Search in Space + Time

The Defect

Search in Time
Search in Time

Search in Space
State of the GNU compiler (GCC)
42991 vertices
44290 edges - and 1 is wrong :-)
An actual GCC execution has millions of these states.
A Sample Program

$ sample 9 8 7
Output: 7 8 9
$ sample 11 14

```c
int main(int argc, char *argv[])
{
    int *a;
    int i;
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);
    shell_sort(a, argc);
    printf("Output: ");
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");
    free(a);
    return 0;
}
```

Find Origins

- The 0 printed is the value of a[0]. Where does it come from?
- Basic idea: Track or deduce value origins
- Separates relevant from irrelevant values
- We can trace back a[0] to shell_sort
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}

Observing a Run

 variables

 time

 Specific Observation

static void shell_sort(int a[], int size)
{
    fprintf(stderr, "At shell_sort");
    for (i = 0; i < size; i++)
        fprintf(stderr, "a[%d] = %d\n", i, a[i]);
    fprintf(stderr, "size = %d\n", size);
    int i, j;
    int h = 3;
    ...}

The state is infected at the call of shell_sort!

FIXME: argv[0] should be “sample”, not “11”
Fixing the Program

```c
int main(int argc, char *argv[]) {
    int *a;
    int i;
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);
    shell_sort(a, argc - 1);  // $ sample 11 14
    shell_sort(a, argc);      // Output: 11 14
    ...
}
```

Finding Causes

- Infected state
- Sane state

The difference causes the failure

Search in Space

- Infected state
- Sane state
- Mixed state

Test

argc = 3
int main(int argc, char *argv[]) {
    int *a;
    // Input array
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (int i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);
    // Sort array
    shell_sort(a, argc);
    // Output array
    printf("Output: ");
    for (int i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");
    free(a);
    return 0;
}
**Concepts**

★ A failure comes to be in three stages:
  1. The programmer creates a defect
  2. The defect causes an infection
  3. The infection causes a failure — an externally visible error.
★ Not every defect results in an infection, and not every infection results in a failure.

**Concepts (2)**

★ To debug a program, proceed in 7 steps:

T rack the problem
R eproduce
A utomate
F ind Origins
F ocus
I solate
C orrect

**Concepts (3)**

★ A variety of tools and techniques is available to automate debugging:

- Program Slicing
- Observing & Watching State
- Asserting Invariants
- Detecting Anomalies
- Isolating Cause-Effect Chains