Chapter - 17 Debugging and Optimization

Debugging Techniques

- Divide and conquer
- Debug only code
- Debug Command Line Switch

  Note: Use I/O redirection and the editor to browse large volumes of debug output.
- Interactive Debuggers

Gnu Debugger (gdb) commands:

`run` Start execution of a program.

`break` *`line-number`*

Insert a breakpoint at the given line number.

`break` *`function-name`*

Insert a breakpoint at the first line of the named function.

`cont` Continue execution after a breakpoint.

`print` *`expression`*

Display the value of an expression.

`step` Execute a single line in the program.

`next` Execute a single line in the program, skip over function calls.

`list` List the source program.

`where` Print the list of currently active functions.

`status` Print out a list of breakpoints

`delete` Remove a break point.

A program to debug

When we run this program with the data 3 7 3 0 2 the results are:

GDB run:.

First Debugging Session

run

next

next

next

All right how did seven_count get to 2?

Figuring out what happened

The value got change in get_data. Let's look through it.

```
break get_data
```

```
run
```

```
next
```

Binary Search

```
/**************************************************************

 **************************************************************/
```

```
main()
{









    }
    /*
     */



            break;



            break;
        ++max_count;
    }
```

```
                break;

                }

                    break;
                }

                else

            }
        }

    }
```

Data

```
4
6
14
16
17
-1
```

When we run this program on UNIX, the results are:

```
% search
Segmentation fault (core dumped)
```

Debugging Session

```
(gdb)
```

```
(gdb) where
```

```
(gdb)
```

```
(gdb) quit
```

```
list main
```

```
(gdb)
```

```
run
```

```
step
step
step
step
step
step
step
```

```
quit
y
```

If at first you don't succeed, play second base.

We try again:

```
Enter number to search for or -1 to quit:4
Found at index 0
Found at index 0
Not found
Enter number to search for or -1 to quit:^C
```

```
run


step

step

step

step

step

step

step
```

The Mistake

```
    }
```

Changes to:

```
        break;
    }
```

Try again:

```
search
```

```
4
6
3
5
```

Debug Session III

```
        run
                                        5
^C


 (gdb)

        step
        step
        step
        step
        step
        step
        step
        step
        step
        step
        step
        step
```

Debugging Cont.

```
step

step

step

step

step

step

print low

print middle

print high

print search



quit
                                                    y
```

Final Fix

```
if (data[middle] < search)
    low = middle;
else
    high = middle;
```

Should be:

```
if (data[middle] < search)
    low = middle +1;
else
    high = middle -1;
```

Tricking Interactive Debuggers to Stop when you want them to

```
{

                green*correction);
}
```

Runtime Errors

Segmentation Violation

- Bad pointer
- Indexing off the end of an array

Stack Overflow

- Too many local variables (big problem in DOS/Windows).
- Infinite recursion

Divide by 0

Floating exception (core dumped)

- On UNIX this is caused by floating point and *integer* divides.

Buffering problem

```
main()
{




    return(0);
}
```

When run, this program outputs:

```
Starting
Floating exception (core dumped)
```

Problem Solved

```
int main()
{




    std::cout.flush();

    std::cout.flush();


    std::cout.flush();
    return(0);
}
```

Confessional Method of Debugging

Programmer explains his program to an interested party, an uninterested part, a wall. The programmer just explains his program.

Typical session:

“Hey Bill, could you take a look at this. My program has a bug in it. The output should be 8.0 and I’m getting -8.0. The output is computed using this formula and I’ve checked out the payment value and rate and the date must be correct unless there is something wrong with the leap year code which — Thank you Bill, you’ve found my problem.”

Bill says nothing.

Optimization

And now a word about optimization:

Don't!

Getting a faster machine is the most economical way to speed up your program.

Unoptimized Program

```
{

                }
        }
}
```

Register Variables

```
{

                }
        }
}
```

The **register** keword is a hint that tells the compiter to put a frequently used variable in a machine register (which is faster than stack memory).

But most modern compilers ignore this hit because they can do register allocation better than an human anyway.

With loop ordering

```
{
                    }
            }
}
```

Powers of 2

Indexing an array requires a multiply. For example to execute the line:

```
matrix[x][y] = -1;
```

the program must compute the location for placing the -1. To do this the

1) Get the address of the `matrix`.
2) Compute `x * Y_SIZE`.
3) Compute `y`.
4) Add up all three parts to form the address. In C++ this code looks like:

```
*(matrix + (x * Y_SIZE) + y) = -1;
```

If we change Y_SIZE from 30 to 32, we waste space but speed up the computation.

Using Pointers

```
{

        ++matrix_ptr;
    }
}
```

Can the loop counter and the `matrix_ptr` be combined?

Using the library function

```
{

}
```

Our function is one line long. We might want to make it an **inline** function.

Optimizing techniques

- Remove invariant code from loops
- Loop ordering
- Reduction in strength
- Use reference parameters
- Powers of 2
- Pointers
- **inline** functions

Optimization Costs

Operation	**Relative Cost**
file input and out (`<<` and `>>`) Also includes the C functions `printf` and `scanf`.	1000
new and **delete**	800
trigonometric functions (sin, cos...)	500
floating point (any operation)	100
integer divide	30
integer multiple	20
function call	10
simple array index	6
shifts	5
add/subtract	5
pointer dereference	2
bitwise and, or, not	1
logical and, or, not	1