ECS 36A, May 17, 2023

Announcements

1. We haven’t graded the midterms yet; our target date is by Friday

gdb

- A dynamic debugger
- To run it, compile your program with the `–g` option
  - This adds in debugging information *gdb* uses
  - You can use *gdb* without it but it simplifies the use greatly
- Then load it into *gdb* by:

```
gdb executable
```

- Note you use the executable file and *not* the source code file
  - You can also load the executable once *gdb* starts

Inside the *gdb* Shell

- Once started, you get a prompt "(gdb)"
- If you forgot to name the executable in the command line:

```
(gdb) file executable
```

- One other handy feature

```
(gdb) help
```

- You will get a list of commands you can ask for help on
- Then type

```
(gdb) help command
```

Executing the program

- Type:

(gdb) run arg_1 . . . arg_n

- This runs the program with command line arguments arg_1 through arg_n
  - If there are no command line arguments, just type ``run''
- If there are no problems, the program runs to completion
- If the program stop with a message like this, there's a problem

```
Program received signal SIGSEGV, Segmentation fault.
0x00005555555551b5 in nfact (n=<error reading variable: Cannot access
memory at address 0x7fffff7fefec>) at nfact2.c:12
```

Stopping the Program Before It Ends

- A *breakpoint* causes the execution to stop at that point
- Here's an example:

```
(gdb) break 15
Breakpoint 1 at 0x5555555551b8: file nfact2.c, line 15.
```

- This causes execution to stop when it reaches line 15
  - If you have multiple source files, name the file before the number:

```
(gdb) break nfact2.c:15
```

- It shows some useful information

```
Breakpoint 1, nfact (n=15) at nfact2.c:15
15 x = nfact(n+1);
```

Conditional Breakpoints

- Causes a breakpoint to stop execution when a condition is met
- Here's an example:

```
(gdb) break 15 if n >= 20
Breakpoint 1 at 0x5555555551b8: file nfact2.c, line 15.
```

- This causes execution to stop when it reaches line 15 *and* n is 20 or more
  - If you have multiple source files, name the file before the number:

```
(gdb) break nfact2.c:15 15 if n >= 20
```

What Can You Do When Stopped?

- You can continue the execution from the breakpoint:

```
(gdb) continue
```

- You can execute one statement at a time to step through the program
  - If it encounters a function, it goes into that function and executes one statement at a time

```
(gdb) step
```

  - n (next) is like s but treats the function as part of the statement and does not go into it

```
(gdb) next
```

Printing Values

- You can print the value of an expression

```
(gdb) print expression
```

- If you prefer hexadecimal

```
(gdb) print/x expression
```

Watchpoints

- Like breakpoints, but keyed to variables

```
(gdb) watch x
```

- Whenever x changes values, the program stops and *gdb* prints old and new values of x

Other Useful Commands

- backtrace
- where
  - These show the stack, that is, the functions that have been called and not yet returned
- delete 2
  - Delete breakpoint 2 (or watchpoint 2)
- info breakpoints
  - List the breakpoints (and watchpoints)
- info frame
  - Show the *current* frame

And now a Word About argv

```
void main(int argc, char *argv[])
```

- Program name is `argv[0]`
- One way to go down the arguments (`j` is declared as `int j`):

```
for(j = 1; j < argc; i++)
    printf("Argument: %s\n", argv[1]);
```

- And the same thing, but using pointers (`a` is declared as `char **a`):

```
for (a = argv+1; *a; a++)
    printf("Argument: %s\n", *a);
```

How Numbers and Letters Are Represented

- The computer stores these in binary representations
- Examples:
  - 345 in binary is 0000 0000 0000 0000 0000 0001 0101 1001
  - –345 in binary is 1111 1111 1111 1111 1111 1110 1010 0111
    - This is two's complement; flip the bits, add 1, and ignore overflow
  - If you add these, you get 0000 0000 0000 0000 0000 0000 0000 0000
  - 'a' is 97, which is 0110 0001
  - Floats use a different format:
  - 2.456 is 0100 0000 0001 1101 0010 1111 0001 1011

sign bit exponent mantissa

Type Coersion

```
int n;
float j = 2.456;
. . .
n = (int) j;
printf("float is %f, int is %d\n", j, n);
```

prints

```
float is 2.456000, int is 2
```

Representation of Data

- But if we want the bitwise representation of 2.456, we need to use a union

Unions

- Allows data to be viewed as multiple types
- Syntax is like a structure:

```
union intfloat {
    int un;
    float uj;
} t;
```

Unions

- So to get the representation of 2.456 in binary:

```
t.uj = 2.456
printf(“bit representation is 0x%x\n”, t.un);
```

- And this prints

```
bit representation is 0x401d2f1b
```