Tolerable performance on this quiz:
answering more than half of the
questions correctly in half an hour.
Carefully review any incorrect answers;
errors are bad in most contexts
but particularly bad in this course.

Everyone else:
Learn C or drop the course.

Frank Tiboni, Federal Computer Week, 2004.08.31:

“Army CIO asks for better security

“The Army’s chief information officer wants service and industry information technology officials to do a better job of protecting networks and building more secure products.

“ ‘Guys, this is eating us up,’ said Lt. Gen. Steve Boutelle, the Army’s CIO, speaking here today at the Directorate of Information Management/Army Knowledge Management Conference sponsored by AFCEA International.

“Army systems are under attack, Boutelle said. He said he wants to buy commercial products that maintain information assurance over time.

“On Aug. 15, Gen. Peter Schoomaker, Army chief of staff, issued a directive to all service commanders stating that they must practice computer security and not expect officials in the CIO’s office to do everything. The five-point plan detailed steps IT officials can take to improve information assurance at their installations including installing antivirus software and getting personnel to use unique passwords, Boutelle said.”

Assignment due 2004.08.25: read foreword and preface of textbook.

Assignment due 2004.08.27: read textbook Chapter 1 pages 1–14, up to “The Trinity of Trouble.”

Assignment due 2004.08.30: read the rest of Chapter 1.

Assignment due 2004.09.03: Read Gaim.
`http://cr.yp.to/2004-494/gaim.html`

What is inside a computer?

A typical computer has these variables:

```
  char RAM[4294967296];
  char *ip;
  int *sp;
  int *bp;
  char *si;
  char *di;
  int ax, bx, cx, dx;
  int flags;
  int segments[6];
  int control;
  long double st[8];
```

C variables are assigned space inside `RAM`.

Pointer values are locations in `RAM`.

The non-`RAM` variables are “registers.”

Computer reads an instruction by reading `*ip++`, reading `*ip++`, etc. until it has a complete instruction.

Computer modifies a few variables as specified by the instruction. Jumps ("goto") are instructions that specify modifications to `ip`.

e.g. Assume that `*ip` is `0xc3` on an x86-architecture computer. Computer reads byte `*ip++`, sees that it's `0xc3`, and then does `ip = (char *) *sp; sp = sp + 1`. In other words, it does `goto *sp++`.

After computer follows the instruction, it reads another instruction, etc.

Example:

```
int target = 5;
char payload[] = {
  0xb8, 0xed, 0x5f, 0x84, 0x00
, 0xa3, 0x28, 0x95, 0x04, 0x08
, 0x83, 0xc4, 0x1c
, 0xc3
} ;
void smash(int a)
{ (&a)[-1] = payload; }
void middle(void)
{ smash(11); }
int main(int argc,char **argv)
{ middle();
  printf("%d\n",target);
}
```

Compiler on this computer decides to put `target` at location `0x08049528` in `RAM`, `*smash` at location `0x080484a0`, etc. When program starts,
`RAM[0x08049528]` is set to `0x05`,
`RAM[0x08049529]` is set to `0x00`,
`RAM[0x0804952a]` is set to `0x00`,
`RAM[0x0804952b]` is set to `0x00`;
`RAM[0x080484a0]` is set to `0xc7`,
`RAM[0x080484a1]` is set to `0x04`,
`RAM[0x080484a2]` is set to `0x24`,
`RAM[0x080484a3]` is set to `0x2c`,
`RAM[0x080484a4]` is set to `0x95`,
`RAM[0x080484a5]` is set to `0x04`,
`RAM[0x080484a6]` is set to `0x08`,
`RAM[0x080484a7]` is set to `0xc3`; etc.

More compact picture:

```
0x080484a0  c7 04 24 2c
0x080484a4  95 04 08 c3
...
0x08049528  05 00 00 00
```

Picture with some C names for locations:

```
0x080484a0==smash    c7 04 24 2c
0x080484a4           95 04 08 c3
...
0x08049528==&target 05 00 00 00
```

Picture with instructions translated:

```
0x080484a0  smash:
            *sp = payload
0x080484a7  goto *sp++
...
0x08049528  05 00 00 00
```

`smash` ends up doing `goto payload`,
i.e. `ip = payload`.

So computer reads instructions from
`payload[0]`, `payload[1]`, etc.

`payload` was written in C as an array,
not a function, but computer doesn't care.

What are the `payload` instructions?

Program instructions, including `payload`:

```
payload:
  ax = 0x845fed;
  target = ax;
  sp += 7; goto *sp++;
smash:
  *sp = payload;
  goto *sp++;
middle:
  sp -= 6;
  *--sp = 11; *--sp = t95;
  goto smash;
  t95: sp += 7; goto *sp++;
main: ...
```

So `target` changes to `0x845fed`.

Program prints 8675309, not 5.

Changing the bytes in `payload`
can tell the computer to do anything,
just like changing the C code
in `main`, `middle`, etc.

For comparison: Previous example
printed `two two` by following
instructions that were in the C code,
but inserting an unexpected jump
from `three` to `two`.

Question: Are unexpected jumps
powerful enough to do anything?
Depends on the program;
yes for most large programs.
Will come back to this question.

Example:

```
int main(int argc,char **argv)
{ char line[512];
  line[0] = 0;
  gets(line);
  ...
}
```

This is the original version of the
UNIX `fingerd` network server.
Its standard input reads from the network;
its standard output writes to the network.

The first Internet worm,
released by Robert T. Morris Jr. in 1988,
spread primarily by
seizing control of this program.

The `gets` function writes to `line`.
How does it know the number of bytes available in `line`? It doesn't!

Here's the code inside `stdio`:

```
char *gets(char *s)
{ int ch;
  int i = 0;
  for (;;) {
    ch = getchar();
    if (ch == EOF) ...
    if (ch == '\n') break;
    s[i++] = ch;
  }
  s[i] = 0;
  return s;
}
```