CSC 180 lab 7A: Command-line arguments

Thurs Oct 25 or Mon Oct 29, 2001

Note: For week 7, there are two half-labs; this week's lab session is a combination of two separate, short topics. Please also see lab 7B, ‘‘Structs’’.

As a systems-oriented programming language, C must provide a facility to pick up ‘‘command-line arguments’’. For example, when you type ‘‘cat file’’, the ‘‘cat’’ command has to be able to find that string ‘‘file’’.

C does this by providing an alternate allowable definition of main(), whose declaration looks like this:

```
int main(int argc, char **argv)
```

‘‘argc’’ stands for ‘‘argument count’’. It is the number of elements in ‘‘argv’’, ‘‘argument vector’’ (‘‘vector’’ is another word for ‘‘array’’, in computer programming). While these are theoretically arbitrary parameter names, the use of the names ‘‘argc’’ and ‘‘argv’’ is *so* standard that you should not use any other name.

1. Write a program using this extended main() declaration syntax, and simply print the value of argc. Run it with various numbers of arguments, e.g.

```
./a.out x y z a b c
./a.out
./a.out x a
```

The value is always one more than you might expect. This is because that ‘‘./a.out’’ is included in the count, and is available as argv[0].

2. Since a string can be passed around as a value of type pointer-to-char, argv should be an array of strings. And when it is passed to main() by the operating system, it will decay into a pointer to its zeroth element, so the data will be of type pointer-to-pointer to char.

Write a program which first checks that argc is at least 2, then prints the value of argv[1] (using %s). A session with this program might look like this, where ‘%’ is your shell prompt:

```
% ./a.out hello, world
hello,
%
```

Since "world" is argv[2], it didn't print that part. The shell (command interpreter) divides up the arguments based on spaces.

3. Write a program which prints all of the argv values in a loop. Try it out.