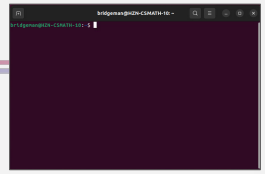
Pathnames

- pathnames
  - absolute – start with `/` or `~`
    - `/classes/cs331/handin`
    - `/home/bridgeman/cs331`
    - `~/cs331`
  - relative – don't start with `/` or `~`, interpreted relative to the current directory
    - `cs331/handin`
    - `../cs331`
  - shortcuts and special notations
    - `/` (root)
    - `~` (home directory), `~arthur` (arthur's home directory)
    - `..` (parent directory)
    - `.` (current directory)

The Command Line



- Applications Menu → Terminal
- basic commands
  - show current directory – `pwd`
  - change directories – `cd` *`dir`*, `cd ..`
  - list contents of a directory – `ls`, `ls` *`path`*
    - `ls -l` – long listing
    - `ls -lrt` – long listing, reverse order, by time
  - delete (permanently!)
    - a file – `rm` *`file`*
    - an empty directory – `rmdir` *`dir`*
    - a directory and all of its contents – `rm -r` *`dir`*
  - create a directory – `mkdir` *`dir`*

The Command Line

- tab completion
  - tab once for max unambiguous completion
  - tab twice for possible completions
- history
  - browse history – up arrow to go back, down arrow to go forward
  - search previous commands – `ctrl-R`, then start typing the search string; repeat ctrl-R for earlier matches

The Command Line

- process management
  - terminate a running program – `ctrl-C`
  - start a program in the background – *`cmd`* `&`
  - stop a running program – `ctrl-Z`
  - background a stopped program – `bg`, `bg %`*`n`*
  - foreground a program – `fg`, `fg %`*`n`*
  - list running programs – `jobs`

C for Java Programmers

2.1 What's The Same?

Since Java is derived from C, you will find many things that are familiar:

- Values, types (more or less), literals, expressions
- Variables (more or less)
- Conditionals: `if`, `switch`
- Iteration: `while`, `for`, `do-while`, but not `for`-in-collection ("colon" syntax)
- Call-return (methods in Java, functions in C): parameters/arguments, return values
- Arrays (with one big difference)
- Primitive and reference types
- Typecasts
- Libraries that extend the core language (although the mechanisms differ somewhat)

C for Java Programmers

2.2 What's Different?

On the other hand, C differs from Java in some important ways. This section gives you a quick heads-up on the most important of these.

- No classes or objects: C is not an object-oriented language, although it has structured types (`struct` and `union`) and there are ways of doing things like inheritance, abstraction, and composition. C also has a mechanism for extending its type system (`typedef`).
- Arrays are simpler: there is no bounds checking (your program just dies if you access a bad index), and arrays don't even know their own size!
- Strings are much more limited, although the C standard library helps.
- No collections (lists, hashtables, *etc.*), exceptions, or generics.
- No memory management: You must explicitly allocate (`malloc`) and release (`free`) memory to use dynamic data structures like lists, trees, and so on. C does almost nothing to prevent you from messing up the memory used by your program. **This is the number one cause of problems for new C programmers (and old ones also).**
- Pointer arithmetic: You can, and often have to, use addresses of items in memory (called pointers). C allows you to change the contents of memory almost arbitrarily. This is powerful magic, but also sometimes dangerous magic.

Bottom line: When you program in C you are closer to the machine. This gives you more flexibility but less protection. The next chapter explains this in more detail by comparing the development and execution models of Java and C.

https://www.cs.rochester.edu/u/ferguson/csc/c/c-for-java-programmers.pdf

Some Gotchas / Things to Know

- there is no `boolean` type in C
  - use `int` instead – 0 means false, non-zero means true
- `stdout`, `stderr`, `stdin`
  - predefined streams for output (typically goes to the screen), error messages (typically goes to the screen), input (typically comes from the keyboard but they can be redirected in the shell
  - `./prog < infile` – stdin for `prog` reads from `infile`
  - `./prog > outfile` – stdout for `prog` is written to `outfile`
- `printf("format",arg1,arg2,…)`
  - format specifiers – `%d` int, `%f` double, `%c` char, `%s` string (`char *`)
  - formatting – `%.2f` (2 decimal places), `%6d` (6 total characters)
  - escapes – `\n` newline
    - must explicitly include newline – there's no `println` version like Java
  - e.g. `printf("x=%d y=%.2f\n", x, y);`

Some Gotchas / Things to Know

- `return` vs `exit`
  - `return` returns from a subroutine
  - `exit` ends the program
  - `return` and `exit` largely equivalent in `main` but there are some subtleties if the caller of `main` does stuff after `main` exits and before terminating
- functions must be defined before they are used
  - order of definitions matters
- pointers and pointer variables
- dynamic memory allocation
- arrays
  - see sections 2.2-2.5 of *Dive Into Systems* https://diveintosystems.org/book/C2-C_depth/