Pointers

A pointer is a variable that contains the address of another variable.

Since a pointer contains the address of an object, it is possible to access the object “indirectly” through the pointer. For example,

```c
int x;
int* px;
px = &x;
```

means px contains the address of x, or “points” to x.

Similarly,

```c
int y = *px;
```

means that y gets the value stored at the address in px (the value px “points” to).

You can do dereference more than once with the use of multiple operators:

```c
char** commandA;
```

means `commandA` is a `pointer` to an array of `pointers to chars` (which we can think of as strings).
Another way to say this is that it references an address to an array of addresses.

Assuming that space in memory has been allocated, draw a memory diagram showing the result of executing the following statements:

```c
commandA[0] = "emacs";
commandA[1] = "strings.c";
```

**Pointer Arithmetic**

If p is a pointer, then p++ increments p to point to the next element of whatever kind of object p points to. So, the actual number by which p gets increments is a multiple of the size in bytes of the object pointed to.

```c
int *p;
p++; p++;
```

results in p being incremented by the size of an integer in bytes on the particular machine on which the operation is performed.

If the word size is 32 bits, p is incremented by 4.

If the word size is 64 bits, p is incremented by 8.
Important Data Types

The result of subtracting two pointers in C is always an integer, but the precise data type varies from C compiler to C compiler. Likewise, the data type of the result of `sizeof` also varies between compilers.

**ptrdiff_t**

This is the signed integer type of the result of subtracting two pointers. For example, with the declaration:

```c
Char* p
```

the expression `p2 - p1` is of type `ptrdiff_t`. This will probably be one of the standard signed integer types (short int, int or long int), but might be a nonstandard type that exists only for this purpose.

Data Type: **size_t**

This is an unsigned integer type used to represent the sizes of objects. The result of the `sizeof` operator is of this type.

**Usage Note:** `size_t` is the preferred way to declare any arguments or variables that hold the size of an object.

Memory Allocation
GNU Debugger (gdb)

Tutorials and manuals:

http://wellesleycs240.bitbucket.org/tools.html

Commands
Can be shortened to a single letter, or repeated by entering <return> at the prompt):

- Compile C program with –g option to create debugging information
- Run the program under gdb

$ gdb testprog

(gdb) run

- Set breakpoints

(gdb) break main

- Step/next statement by statement through your program

(gdb) step
(gdb) next
(gdb) cont -- continue execution
• Display/print code or values of variables and arguments

(gdb) list
(gdb) print x
(gdb) info locals
(gdb) info args

• (gdb) quit or Ctrl-d -- to exit.

• To find a bug:

  1. Set breakpoints at the start of every function
  2. Restart the program and step line-by-line until you locate the problem exactly.
  3. If program is stuck (infinite loop) Ctrl-c terminates the action of any gdb command that is in progress and returns to the gdb prompt.

• Execute statements/expressions during execution to tweak program execution state

(gdb) set var i = 2

• Display/print binary and hexadecimal representation of variables and arguments

(gdb) print /x result -- uses hex representation
(gdb) print /t result -- uses binary representation