

Tips on Pointers and Pointer Arithmetic in C, or How to Succeed in 0019 CW2

Brad Karp
UCL Computer Science



CS 0019
28th January 2020

Pointers in C: Basics

■ Pointer declaration

- e.g., a pointer to long is declared as:
`long *longp;`
- general form: for type **T**, pointer to value of type **T** is:
`T *Tp;`

■ Taking a variable's address

- the `&` operator, applied to a variable
- e.g., the address of a `long` may be taken as in third line of example in yellow box above

■ Dereferencing a pointer

- the `*` operator, applied to a pointer
- e.g., setting the contents of memory at the address stored in `xp` as in fourth line of example in yellow box above

■ Output of example at top right: 9

```
long x = 7;
long *xp;

xp = &x;
*xp = 9;

printf("%d\n", x);
```

Arrays vs. Pointers in C: Basics

- **The variable name for an array also functions as a pointer to first element of that array**
 - i.e., in below code, `x` by itself in an expression is of type `(long *)`
- **Compiler implements C indexing into array in assembly by computing address of desired array element from address of array's first element**
- **...which brings us to [pointer arithmetic](#) in C**

```
long x[3];
long *xp;

x[0] = 17;
xp = x;

printf("%d\n", *xp);
```

- **output: 17**

Pointer Arithmetic in C

- **C allows one to construct expressions in which one adds and/or subtracts integers to/from a pointer**

- e.g., as in 4th line of example at right

- **C's rule for pointer arithmetic:**

- when adding integer *i* to pointer to type *T*, advance address by *i* * `sizeof(T)` bytes
- in example at right, we have *x* of type `(long *)`, a pointer to type `long`
- `sizeof(long)` is 8 bytes
- so address *x* is increased by the number of bytes in memory taken up by 2 `long`s, or **16 bytes**

- Never forget: C pointer arithmetic on a `(foo *)` doesn't interpret the added value in bytes, but in number of chunks of `sizeof(foo)` bytes!

- Of course, `sizeof(char)` is 1, so pointer arithmetic on `(char *)` is in bytes, and also in units of `sizeof(char)`

- **The path to C pointer arithmetic madness:**

- Add pointer to type *T* where `sizeof(T) > 1` byte (e.g., `short`, `int`, `long`, or a `struct`) to integer values computed in bytes, rather than in number of type *T*s

```
long x[3];
long *xp = x;

x[2] = 42;
xp = x + 2;

printf("%d\n", *xp);
```

Pointer Casts in C

- Sometimes it's handy to manipulate memory region holding data of one (or multiple!) types by using a pointer of different type
- C construct for converting a pointer to type T to a pointer to some other type U , where $U \neq T$ is a **cast**
 - To cast a pointer type to another pointer type, prepend desired pointer type in parentheses to original pointer
 - ...as in example at right

- **Output: 42**

```
long x[3];
char *cp;

x[2] = 42;
cp = (char *) x;
cp += 2*sizeof(long);
printf("%d\n", *(long *)cp);
```

Type `(void *)` in C

- **Pointers of type `(void *)` in C point to data of unknown type**
 - Sometimes convenient when type of pointed-to data unknown
 - e.g., the return type for `malloc()` is `void *`, as `malloc()` doesn't know what type you will store in the memory to which it returns a pointer!
- **Illegal to dereference a `(void *)` pointer**
 - Compiler has no idea what type is pointed to!
- **Can cast a `(void *)` pointer to any other pointer type, but result is undefined behavior if cast is to incompatible type**
 - e.g., `(float *) → (void *) → (int *)` yields undefined behavior
 - We will discuss the perils of undefined behavior in future lecture
 - Don't write code that exhibits UB!
- **C99 spec disallows arithmetic on `(void *)` pointers; gcc compiler allows by default as "extension"**
 - treats as `(char *)`, i.e., increment of 1 to `(void *)` pointer is 1 byte

Extremely Useful Reading

- **CS:APP/3e 3.10.1 (assigned for 23rd Jan)**
- **Goes through several rules discussed in previous slides**
- **These slides and above required textbook reading are crucial background to doing CW2**
 - CW2 requires you to allocate memory, cast pointers and do pointer arithmetic...

The 0019 Scoreboard

- Available at link in Piazza post @42
- Anonymized
- Shows CDF of all scores of all students who've checked out each CW
- Scores don't include lateness penalties or late days
- Shows git commit hash that grading server graded, so lets you check that right version of your code (right commit) has been graded
- Lets you see progress of whole class on CWs
- Available all term for CW2-CW5