CSE 220: Systems Programming

Memory and Pointers

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo

Memory

Memory on POSIX systems is data storage identified by address.

All of the data accessible to your C program has an address. ¶

The CPU uses this address to retrieve data from memory.

C/POSIX Memory Model

On a POSIX system, every process appears to have its own memory.

This memory ranges from address zero to the maximum allowable address.

It may be the case that not all of it is available, however!

On Unix systems, the usage of that memory is somewhat predictable.

Pointers

C pointers are variables that hold memory addresses.

This lets your program interact with memory explicitly.

Pointers are very powerful but inherently unsafe tools.

The C compiler doesn't know which pointers are valid!

Most non-trivial data structures in C use pointers.

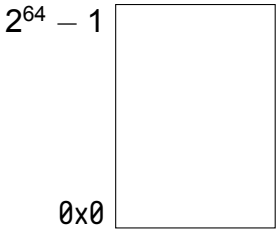
Memory Addresses

On our platform, you can consider memory as a large array.

A pointer is an index into that array.

If memory starts at address 0, a pointer with value `p` is the p'th byte of that array.

Note that any given byte may not exist!



Pointer Concepts

A pointer:

- Contains an address
- Allows the memory at that address to be manipulated
- Associates a type with the manipulated memory

Remember, to the computer, memory is just bits.
Programmers supply the meaning.

The special pointer value `NULL` represents an invalid address.

Pointer Syntax — Declaration

A pointer variable is marked with *.

```
char *str;
```

This is a pointer to char.

(`char *` is the idiomatic string type in C.)

A pointer may be marked const, in which case the memory it points to is const.[1]

```
const char *str;
```

It is a good idea to mark pointers const if you don't intend to modify their contents.

[1] There is another type of constant pointer that we won't talk about now.

Pointer Types

What is a pointer to char anyway?

An address of a character-size integer.

```
char *str = "Hello";
```

This says:

- `str` contains an address
- The data at the object stored in `str` is of type `char`

Addresses

Pointers must store a valid address to be used. ¶

There are limited opportunities to create valid addresses:

- Acquire the address of a variable
- Request new memory from the system
- Create a string or array constant
- Calculation from other addresses

Pointers created in other manners probably are not valid.

Pointer Syntax — Taking Addresses

A pointer may be created from a variable using unary &.
This is sometimes called the address-of operator.

```
int x = 42;
int *px = &x;
```

px is now a pointer to x.
(More on the implications of this later.)

Dereferencing a Pointer

Dereferencing a pointer is accessing the data it points to.

It can be dereferenced to read or modify that data.

Dereferencing an invalid pointer is undefined behavior.

This will often result in a segmentation fault, but may silently corrupt memory!

Pointer Syntax — Dereferencing

A pointer is dereferenced with *, ->, or [].
(More on -> when we get to structures.)

The * notation reads the value at the pointer address.

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

- The variable px is created as a pointer to x, an integer.
- The variable y is created as an integer.
- y is assigned the value of x by dereferencing px with *.

Pointer Syntax — Dereferencing

A pointer can also be dereferenced like an array, with `[]`.

`y = px[0];`

- This is exactly the same as `y = *px;`.

`y = px[1];`

- This treats `px` like an array, and retrieves the second element.
- We will explore how this works in more detail later.

Pointers and Arrays

Arrays and pointers are closely related in C.

You can often think of an array variable as a pointer to the first array element, and a pointer variable as an array.

However, they are not the same.

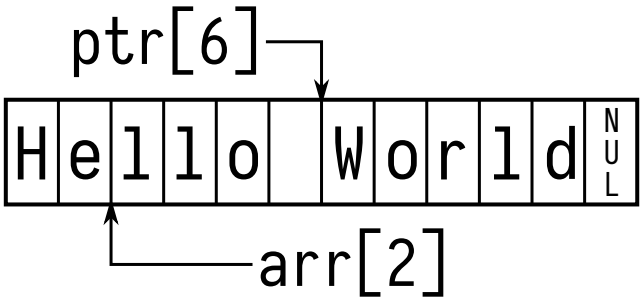
In both cases, dereferencing with `[i]` says

...add `i` *times the size of the type of this variable to the base address (first element of the array or pointer value), then treat the memory at that location as if it is of the type of this variable.*

Pointers and Arrays

Consider:

```
char arr[] = "Hello World";
char *ptr = arr;
```



Arrays Are Not Pointers

```
char arr[] = "string";
char arr2[] = arr;
```

“error: invalid initializer”

```
char arr[] = "Hello World";
char *ptr = arr;
```

`ptr` points to arr[0].

Exploring Pointers

We will explore pointers in a program and the debugger.

Summary

- Memory locations are identified by addresses.
- Addresses are integers.
- Our system's memory is like one large array.
- POSIX processes appear to have their own dedicated memory.
- Pointers hold addresses and have types.
- Unix processes are divided into sections.
- Pointers and arrays are closely related, but not the same.

References I

Required Readings

[1] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Second Edition. Chapter 5: Intro, 5.1-5.4. Prentice Hall, 1988.

License

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.