Memory and Pointers

CSE 220: Systems Programming

Ethan Blanton & Carl Alphonce

Department of Computer Science and Engineering
University at Buffalo
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 predictable.
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.
Time Management

Make progress by setting a timer.

Set a timer for 15 minutes. When it expires:

- Are you still working?
  - If not, why not? Fix the problem!
- Are you making progress?
  - If not, why not? Fix the problem!

Time spent is different from results achieved.
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:
- Is 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.
A pointer variable is marked with *.

```c
char *str;
```

`str` is a variable which stores a pointer to `char`.

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

A pointer variable may be marked `const`, in which case the memory it points to is `const`.

```c
const char *str;
```

It is a good idea to mark pointer variables `const` if you don’t intend to modify their contents.

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

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

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

px now stores a **pointer to x**. (More on the implications of this later.)
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!
A pointer is dereferenced with *, ->, or [].
(More on -> when we get to structures.)

The * notation reads the value at the pointer address.

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

- The variable px is initialized with 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 *.
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:

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

```
ptr[6]  
Hello World   NUL
```

```
arr[2]
```
Arrays Are Not Pointers

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

“error: invalid initializer”

```c
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**.
- Pointers and arrays are **closely related, but not the same**.
References I

Required Readings

License

Copyright 2019–2023 Ethan Blanton, All Rights Reserved. Copyright 2022, 2023 Carl Alphonce, All Rights Reserved. Copyright 2019 Karthik Dantu, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

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