Pointers and Arrays

Friday, October 2, 2020 11:58 AM

Pointers

Defn. A pointer is a variable that stores a memory address.

Properties. Can pass around objects by pointers

Size - 8 bytes (on 64-bit machines)

Useful for allocating heap memory

Let us refer to memory generically

Memory Arr. of bytes - referred to w/ mem. address

Syntax.

```
int i = 0;
int *ptr = &i; // pointer
*ptr = 1;      // dereference and set
int j = *ptr;  // dereference
```

Parameters Don't want changes to input - pass data type

Want to modify - pass location (pointer to) data

char * Pointer to a single character.

Can be used to signify a string by having char it points to having more characters following it up to a '\0'

Double Pointers

Why are they useful? Can directly modify memory address to which a pointer points to

Arrays

Memory Refers to a single block of memory
Cannot reassign existing array to be equal to a new one

Block of memory is contiguous
- var is array itself, not pointer
- sizeof returns size of the array

Passing as parameter When array passed as a parameter, a copy of a pointer to the first element of the array is passed

This means we need to pass array length as a separate parameter.

Arrays of pointers

Why? Group multiple pointers (e.g. strings) together

ex. `char *stringArray[5]; //stores 5 char*'s, // not all chars for 5 strings`

Pointer Arithmetic

Why? Advance pointer forwards by some places

Notes Doesn't work in bytes - instead, in the size of the type it points to

ex.
```
int *nums = ...,      // 0xff0
int *nums1 = nums+1;  // 0xff4
int *nums2 = nums+3;  // 0xffc
```

Sizes determined at compile time

Const, Structs, and Ternary

Const. Use to declare global constants — indicates variable can't be changed after being created

Can also be used w/ pointers to indicate that underlying data can't be changed

ex.
```
const char *foo = "bar";
foo[1] = 'c'; // invalid
```

Struct. A way to define a new variable type that is a group of other variables.

ex.

```
struct node {
    int val;
    node * next;
}
```

```
struct node cur;
cur.val = 3;
struct node next = {4, NULL};
cur -> next = next;
```

Passing as param - by default, passed as copy

Sizeof - yields sum of sizeofs of each element