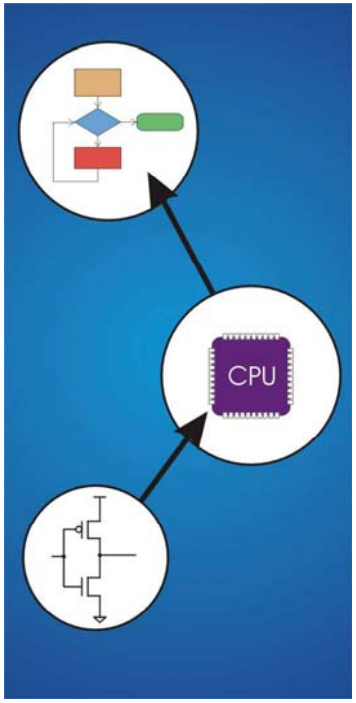


COMPSCI 210
Part II
Pointers Arithmetic

Agenda & Reading

Agenda:

- Pointers Arithmetic
- Common Mistakes
- 2D Array & Pointers
- Arrays of Strings

Exercise:

- Which of the following pointer expressions affects the value of the pointer?

Example: 01arithmetic.c

Pointers Arithmetic

Integer math operations can be used with pointers. But not all operations are legal.

Legal:

- Add an integer to a pointer
- Subtract 2 pointers (in the same array)
- Comparing pointers

Illegal:

- Adding 2 pointers
- Multiplying 2 pointers
- Dividing 2 pointers
- Adding float to a pointer
- Shifting pointers
- etc

Make no sense in a program

Pointers Arithmetic

Adding an integer (positive/negative)

- If you increment/decrement a pointer, it will be increased/decreased by the size of whatever it points to.
  - Adding one (for instance) to an address should direct it to the next object, not necessarily the next byte.
    - If the address points to integers, then it should end up pointing to the next integer, actually increase the address by four.

```
p1 = array;
*(p1+1) = 30;
```

*p1	
*p2	
array	10
	20->30
	30 -> 60
	40 -> 50
	50

  - Subtracting one to an address should direct it to the previous object

```
p1 += 3;
*p1 = 50;
p1 -= 1;
*p1 = 60;
```

Pointer Arithmetic

***Address calculations depend on size of elements

- In our LC-3 code, we've been assuming one word per element.
  - e.g., to find element at index 4, we add 4 to base address
- It's ok, because we've only shown code for int and char, both of which take up one word.
- If double, we'd have to add 8 to find address of the element.

C does size calculations under the covers, depending on size of item being pointed to:

```
double x[10];
double *y = x;
*(y + 3) = 13;
```

allocates 20 words (2 per element) – LC-3 Code
Allocates 10*8 bytes (C compiler on Chaos)

same as x[3]
base address plus 6 (LC-3 code)
Base address + 3 * 8 bytes (C compiler on Chaos)

Pointers Arithmetic

Subtracting 2 pointers (in the same array)

- = the difference of two addresses.
- The result is interpreted as the number of objects between the two addresses

```
p1 = array;
p2 = &array[2];
printf("difference=%d\n", p2-p1);
```

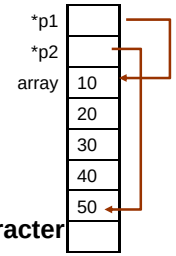
Returns 2

Comparing Pointers

Comparing Two Pointers

- Note: Only addresses within a single array should be compared.
- Example: Given an array, the size is n. We want to access element from 0 to n-1. Use a loop and test for exit by comparing to the address one element past the array

```
p1 = array;
p2 = &array[5];
while ( p1 != p2) {
  printf("%d ", *p1);
  p1++;
}
```



Comparing value pointed by a pointer to zero

- Example: Given a string, we want to access each character

```
char string[] = "Good morning";
cp = string;
while (*cp) {
  printf("%c", *cp);
  cp++;
}
```

If the value is nonzero, continue

Zero = Null terminator character

Example: 02stringExample.c

Example

Convert all characters to uppercase

- Convert all characters in a string to uppercase
- Parameter: pointer to the beginning of string
- Return: a pointer to the beginning of a string

```
char *string =  "good monring" ;
printf("%s\n", stringToUpper(string));
```

```
char *stringToUpper(char *string)
{
  char *cp;
  cp = string;
  while (*cp != '\0') {
    *cp = toupper(*cp);
    cp++;
  }
  return string;
}
```

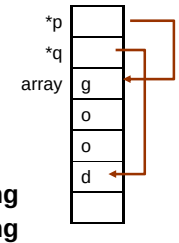
Increment the pointer

Get the character and convert it to upper case

Example

Reverse a string

- Reverse a string
- Parameter: pointer to the beginning of string
- Return: a pointer to the beginning of a string




```
char *string =  "good" ;
printf("%s\n", stringReverse(string));
```

```
char *stringReverse(char *string)
{
  char *p, *q;
  char temp;
  p = string;
  if ( *p == '\0')
    return NULL;
```

p points to the first character

```
  q = string;
  while (*q != '\0')
    q++;
  --q;
  while (p < q) {
    temp = *p;
    *p = *q;
    *q = temp;
    p++;
    q--;
  }
  return string;
}
```

q points to the last character

Swap the values

Common Problem

Error 1:

- What is wrong with the following code?

```
int *p;
*p = 10;
```

  - What is the address in p? (NULL)
    - Writes "10" into a random location in memory
  - C defines that memory location 0 must not be a valid address for pointers

Error 2:

- What do the following statements do?

```
int *p, q;
int* p, q;
int *p; int q;
```

SAME

  - Declare an integer pointer and an integer (not a pointer)
- Declare two integer pointers

```
int *p, *q;
```

```
int *p;
int *q;
```


Common Problem

Error 3:

- What's wrong with:

```
int* func() {
  int x = 1;
  return &x;
}
```

  - Answer: Storage for "x" disappears on return, so the returned pointer is dangling.
  - A dangling pointer is a pointer to storage element that is no longer allocated.

Error 4:

```
int *p, *q, a=100, b=100;
p = &a;
q = &b;
```

- Copy pointers? Copy Values?
  - What does the following statement do? Copy values pointed

```
*ip = *iq;
```

  - What does the following statement do? Copy the address stored

```
ip = iq;
```


Example: 03pointers.c

Common Problem

5:

- What do the following statements do?

```
*ptr+=1;
```

  - Value pointed increment by 1

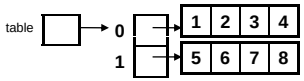
```
(*ptr)+=1;
```

  - Value pointed increment by 1

```
*(ptr+1)
```

  - Address increment by 1 (e.g. next integer)

2D Array & Pointers



Note:

- When we create 2D array to a function we must specify the number of columns
- C needs to know how many columns in order that it can jump from row to row in memory.

Example:

- Creating Pointers
  - ➢ int (*p)[4]; declares a pointer to an array of 4 ints.
    - – Note:int *p[4]; declares an array of 4 pointers to ints.

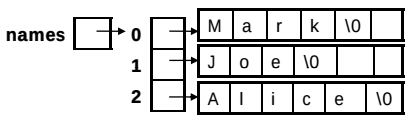
```
int table[2][4] = { {1,2,3,4}, {5,6,7,8}};
int (*p1)[4], (*p2)[4], *x;
p1 = table;
```

  - ➢ Addresses
    - – Return the address of the first pointer (a pointer to an array of 4 ints) `p1`
    - – Return the address of the i^{th} pointer (i^{th} row) `p1 +i`
    - – Return the address of the i^{th} row and j^{th} column `*(p1 + i ) + j`
    - – Return the value of the i^{th} row and j^{th} column `*(*(p1 + i ) + j)`

Arrays of Strings

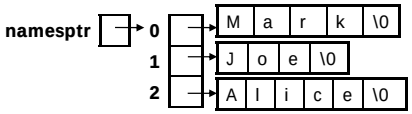
In C, you cannot store an array of strings, but you can store an array of character pointers, and each character pointer can point to a string in memory

Example: using a 2d array – waste of space




```
char names[3][6] = {"Mark", "Joe",
"Alice"};
```

Example: using array of character pointers




```
char *namesptr[3] =  {"Mark", "Joe",
"Alice"};
```

- To print the i^{th} element

```
printf("%s", *(namesptr+i));
```


Tips:

Common Programming Error

- Using pointer arithmetic on a pointer that does not refer to an element in an array.
- Subtracting or comparing two pointers that do not refer to elements in the same array.
- Running off either end of an array when using pointer arithmetic.