LECTURE 16: POINTERS / ARRAYS (K&R §§ 5.1–5.5)

VOCABULARY

```
int k;
k = 2;
```

lvalue	an expression referring to a named region of storage (the memory address of that storage) in the above example, k is the lvalue
rvalue	the value stored in the above example, 2 is the rvalue
pointer	a type of variable that holds an lvalue

POINTER EXAMPLE

```
/* assume these are automatic variables */
int x = 1;
int y = 2;
int z[10];
int *ip;  /* ip is a pointer to an int */
ip = &x;  /* ip is a pointer to int x */
```

how to read a pointer declaration

Read it right-to-left. For example:

```
int *ip
```

variable ip
is a pointer (*)
to a variable of type int

memory address	contents				var. name
0Xff1054	0x00	0x00	0x00	0x01	x
0Xff1050	0x00	0x00	0x00	0x02	y
0Xff104C	0x??	0x??	0x??	0x??	z[9]
...	...				
0Xff1028	0x??	0x??	0x??	0x??	z[0]
0Xff1024	0x00	0xFF	0x10	0x54	ip

POINTER OPERATORS

&

value of &x is address of x, i.e., lvalue of x

```
ip = &x; /* ip now points to x */
```

*****

*ip dereferences a pointer, i.e., accesses the rvalue stored at the lvalue stored in the pointer

```
y = *ip; /* set y = x (the int at address ip) */
```

two meanings:
- in a declaration or formal parameter: "I am a pointer"
- inline: "the value stored where I point"

INCREMENTING POINTERS

A pointer is a number corresponding to the address of the byte used to store the variable.

When you increment a pointer, the address is incremented by the number of bytes used to store the type associated with that pointer.

I.e., the pointer is incremented by size of the data type that it points to.

Example:

```
char *cp;
cp++;     /* byte address is incremented by 1 */
```

Example:

```
int *ip;
ip++;     /* byte address is incremented by 4 */
```

*ip + 1;		add 1 to the int pointed to by ip
*ip += 1;	same	adds one to the int pointed to by ip
++*ip;	same	pre increments int pointed to by ip
++(*ip);	same	same as above, binds right-to-left
*ip++;		point to int at pointer ip, post increment ip binds right to left as *(ip++)
(*ip)++;		post increments int pointed to by ip need (), otherwise binds as *(ip++)

POINTER USE EXAMPLE (SWAP)

Does not work.

```
void swap (int a, int b) {
  int dummy;
  dummy = a;
  a = b;
  b = dummy;
}
```

Variables are not swapped in calling function.

Pointer version.

```
void swap (int *pa, int *pb){
  int dummy;
  dummy = *pa;
  *pa = *pb;
  *pb = dummy;
}
```

Memory addresses are passed in the pointers.

POINTERS AND ARRAYS

C treats an array name (without a subscript value) and a pointer in the same way.

```
int a[10];  /* declare an array of 10 int elements */
int *pa;    /* declare an int pointer */
pa = a;     /* same as pa = &a[0]; */
```

The array name a acts as a specially initialized pointer pointing to element 0 of the array.

a is known as a constant pointer or unmodifiable lvalue.

a is not an lvalue.

```
a = a + 1; /* not possible */
7 = 7 + 1; /* not possible */
```

defining an array	defining a pointer
Allocates the required space for contents of all array elements.	Allocates memory for the pointer but not for the data that the pointer points to.

Since pointers increment based on the size of the data type they point to, we can use a pointer to access successive elements of the array it points to.

array subscript	pointer equivalent
a[0]	*pa
a[1]	*(pa + 1)
a[m]	*(pa + m)

An array name can be used in an expression the same way that a pointer can be in an expression (however, its actual value cannot be changed permanently).

a + m is the same as &a[m]

if pa = a, *(a + m) is the same as *(pa + m)

*(pa + m) can be written as pa[m]

POINTER/ARRAY EXAMPLE

```
int i, a[] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
int *pa = &a[3];
```

What is the value of *(pa + 3) ?	12
What is the value of *pa + 3 ?	9
What happens when i = *pa++ evaluated?	pa = &a[4]
What is the value of i?	6
What happens when i = ++*pa evaluated? (after above statement)	++a[4]
What is the value of i?	9

USE CASE EXAMPLE: STRLEN

strlen as we saw it earlier

```
int strlen(char s[ ]) {
  int n;
  for (n = 0; s[n]; n++)
    ;
  return n;
}
```

strlen with pointers

```
int strlen(char *s) {
  char *cp;
  for(cp = s; *s; s++)
    ;
  return s - cp;
}
```

The pointer version is the more common approach.

We can make it even more compact:

```
int strlen(char *s) {
  char *p = s;
  while(*p++) ;
  return p - s - 1;  /* we don't count the '\0' */
}
```

EXAMPLE: STRCOPY (COPY t TO s)

```
void strcpy ( char *s, char *t) {
  while ( *s++ = *t++ )  /* stops when *t = '\0' */
    ;                    /* look at page 105 examples  */
}
```

EXAMPLE: STRCMP (LEXICOGRAPHIC STRING COMPARE)

```
/* space between * and pointer name is OK */
int strcmp ( char * s, char * t) {
  for (  ; *s == *t;  s++, t++)
    if ( *s == '\0')
      return 0;     /* have compared entire string and
                       found no mismatch */
  return *s - *t;  /* on the first mismatch, return */
}  /* (*s - *t is < 0 if *s < *t and is > 0 if *s > *t) */
```

GRAPHICAL REPRESENTATION OF POINTERS