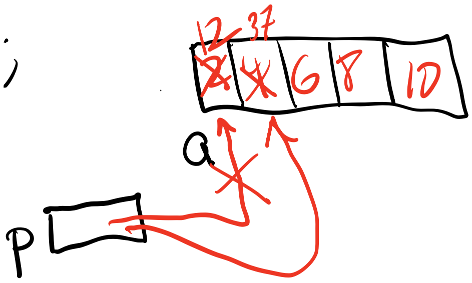
# Pointers and Arrays

```
int a[5]={2,4,6,8,10};
int *p=a;
```



```
*p = *p+10;
p = p+1;     // or p++
             // p's contents is actually incremented
*p=37;       // by 4, to point to the next int.
```

Notice that:

```
int *q;
*q=47;
```

The two *'s are completely different!

```
int *q;
```

* is part of the type of q. "int*" - pointer to int.

```
*q=47;
```

* this is an operator - it says to dereference q.

Pointer arithmetic depends on the *type* of the pointer.

1 byte

```
char *p;
p++;   // just increments p by 1
```

4 bytes

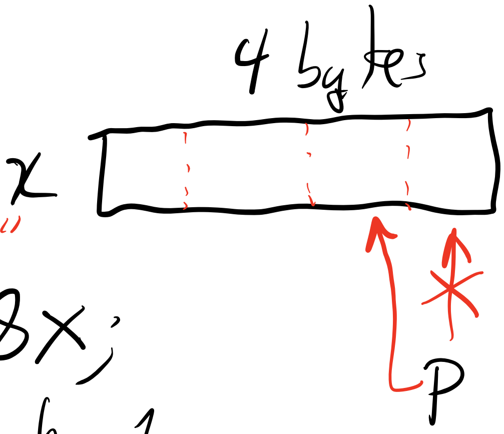
```
int *q;
q++;   // increments q by 4
```

Example:

```
int x = 93;
```

"cast"

```
char *p = (char*) &x;
p++; //increments p by 1.
```



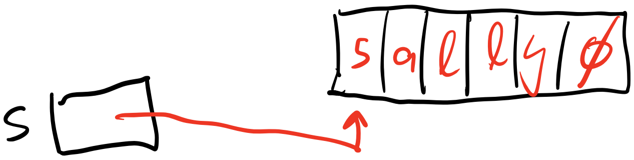
Strings are just arrays of chars

- lots of library functions that operate on strings
  - expect the last element of the string to be the number 0.
    - not the character '0'.

"hello"

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| h | e | l | l | o | 0 |

```
char *s = "sally";
```



|   |   |   |   |   |   |
|---|---|---|---|---|---|
| s | a | l | l | y | 0 |

To print:

```
printf("%s\n", s);
```

specifies the address of the array.

format specifier for a string

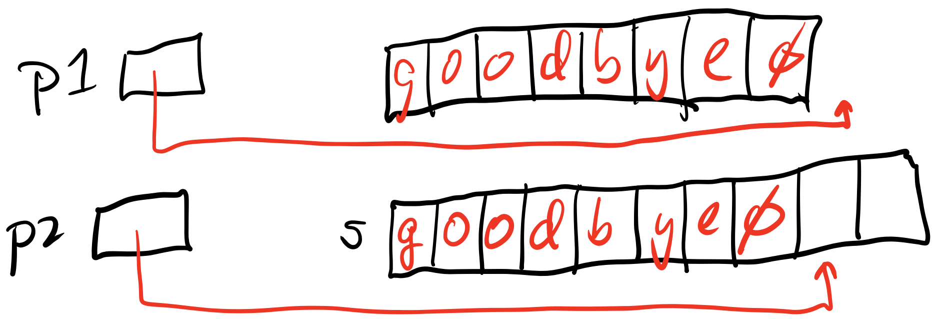
```
printf("There is %d strings, the first is %s\n",
        1, s);
```

# What does this strange code do?

```
char *p1 = "goodbye";
char s[10];
char *p2 = s;
while (*p2++ = *p1++);
printf("%s\n", s);
```

```
while (cond)
    {...}
```

Answer: It copies "goodbye" into s.



- The value of an assignment statement is the value being assigned, so the above loop will terminate after the ∅ is copied
  - since ∅ means false.

Never write code like this!!!

Instead, write:

```
while (*p1 != 0) {
    *p2 = *p1;
    p1++;
    p2++;
}
*p2 = 0;
```

much clearer and no less efficient.

---

## Structures in C

- "struct"
- like a class in C, but no methods, only data fields

```
struct person {
    char name[100];
    int age;
    int salary;
}
```

```
struct person me;   //allocates space for the struct
```

type

variable

```
strcpy(me.name, "Ben Goldberg");
```

library function for copying strings

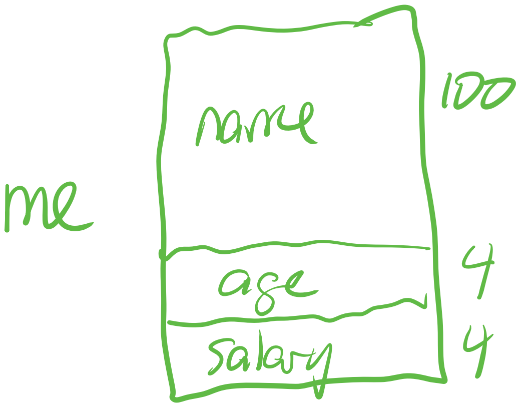
```
me.age = 61;
me.salary = 75;
```

Why not say:

```
me.name = "Ben Goldberg";
```

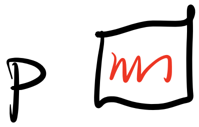
can't assign to the name of an array.

= does not perform a copy.



# Pointers to structs

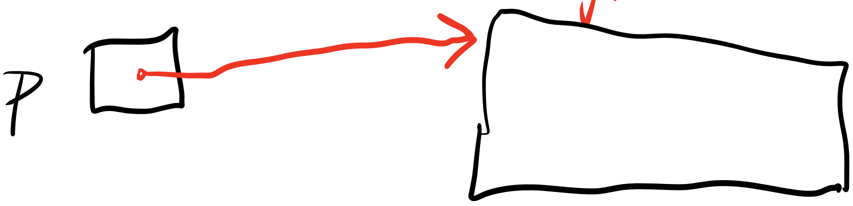
$$\underbrace{\texttt{struct person *}}_{\text{pointer type}}\texttt{p;}$$ // the pointer is allocated, but <u>not</u> the structure.



Need to call "malloc" to allocate space for a struct.

- malloc returns the address of the allocated space.

$$\texttt{p = malloc}\underbrace{\texttt{(sizeof(struct person))}}_{\text{gives the number of bytes in a person.}}\texttt{;}$$



## Filling in the fields

```
(*p).age = 19;  // OK
```

The equivalent syntax that everyone uses:

```
p->age = 19;  // better
```

instead of the "*" and "."

```
p->salary = 500000;
strcpy(p->name, "Sally Field");
```