

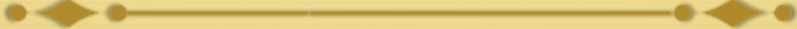


# Same Basic Syntax



- ✦ Data Types: int, char
  - ✦ void - (untyped pointer)
  - ✦ Can create other data types using typedef
- ✦ No Strings - only char arrays
  - ✦ Last character needs to be a 0
    - ✦ Not '0' , but '\0'

# struct – C's object



```
✦ typedef struct foo {  
    int a;  
    void *b;  
    void (*op)(int c); // function pointer  
} foo_t; // <-----type declaration
```

- ✦ Actual contiguous memory
- ✦ Includes data and function pointers

# More on Function Pointers

- ✦ C allows function pointers to be used as members of a struct or passed as arguments to a function
- ✦ Continuing the previous example:

```
void myOp(int c){ /*...*/ }  
  
/*...*/  
foo_t *myFoo = malloc(sizeof(foo_t));  
myFoo->op = myOp; // set pointer  
  
/*...*/  
myFoo->op(5); // Actually calls myop
```

# No Constructors or Destructors

- ✦ Must manually allocate and free memory - No Garbage Collection!
  - ✦ `void *x = malloc(sizeof(foo_t));`
    - ✦ `sizeof` gives you the number of bytes in a `foo_t` - DO NOT COUNT THEM YOURSELF!
  - ✦ `free(x);`
    - ✦ Memory allocator remembers the size of `malloc`'ed memory
- ✦ Must also manually initialize data
  - ✦ Custom function
  - ✦ `memset(x, 0, sizeof(x))` will zero it

# Memory References



✦ ‘.’ - access a member of a struct

✦ `myFoo.a = 5;`

✦ ‘&’ - get a pointer to a variable

✦ `foo_t * fPointer = &myFoo;`

✦ ‘->’ - access a member of a struct, via a pointer to the struct

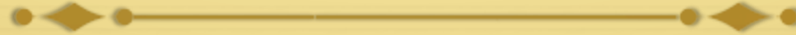
✦ `fPointer->a = 6;`

✦ ‘\*’ - dereference a pointer

✦ `if(5 == *intPointer){...}`

✦ Without the \*, you would be comparing 5 to the address of the int, not its value.

# Memory References, cont.

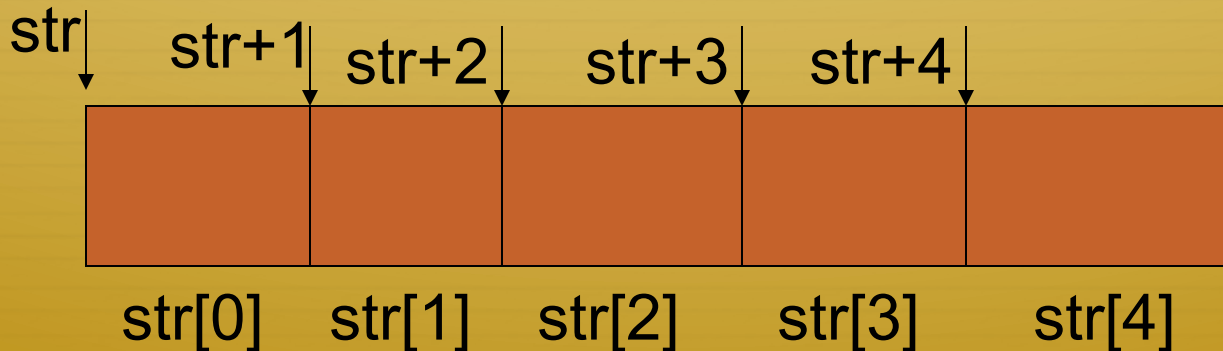


- ✦ '[]' - refer to a member of an array

```
char *str = malloc(5 * sizeof(char));
```

```
str[0] = 'a' ;
```

- ✦ Note: `*str = 'a'` is equivalent
- ✦ `str++`; increments the pointer such that `*str == str[1]`



# The Chicken or The Egg?



- ✦ Many C functions (printf, malloc, etc) are implemented in libraries
- ✦ These libraries use system calls
- ✦ System calls provided by kernel
- ✦ Thus, kernel has to “reimplement” basic C libraries
  - ✦ In some cases, such as malloc, can’t use these language features until memory management is implemented



# Referring to Assembly from C



- ✦ “extern” keyword imports a variable or function
- ✦ Can call a labeled code region as a function if it implements proper calling convention
  - ✦ In most cases, though, you will just inline a “call” instruction

# For more help



- ✦ man pages are your friend!
  - ✦ (not a dating service)!
  - ✦ Ex: 'man malloc', or 'man 3 printf'
    - ✦ Section 3 is usually where libraries live - there is a command-line utility printf as well
- ✦ Use 'apropos *term*' to search for man entries about *term*
- ✦ *The C Programming Language* by Brian Kernighan and Dennis Ritchie is a great reference.