



# C Minicourse

(Day 1)

## **Introduction to C**

“It’s not OOP, but it’s okay!”

- Stacy



# Outline

- Data types
- Structs and unions
- Arrays
- Strings
- printf()
- Enums
- Typedef
- Pointers



# This is C!

```
#include <stdio.h>
```

```
int main(int argc, char **argv) {  
    printf("Hello, world!\n");  
    return 0;  
}
```



# Basic Data Types

- Variables declared with *data types*

- e.g. int, char, float, long

```
int a, b, c;
```

```
a = b = c = 3;
```

- No boolean data type

```
while(true) → while(1)
```



# Structs (1)

- Special data type, groups variables together

## To declare:

```
struct sesamestreet {
    int    the_number;
    char   the_letter;
} ; /* note the semicolon */
```

## To set values:

```
struct sesamestreet st1 = {3, 'a'};
/* the order must be correct */
```

```
struct sesamestreet st2;
st2.the_number = 7;
st2.the_letter = 'm';
```



## Structs (2)

- You can also declare a struct for one-time use

```
struct { /* don't need type name */
    int    length;
    int    width;
    int    height;
} box ;
```

```
/* now we can use it... */
int vol = box.length * box.width *
box.height;
```



# Unions very quickly

- Like a struct, but only *one* member is allowed to have a value.
- You almost never need to use these

```
union grade {  
    int    percent;  
    char   letter;  
};
```

```
union grade stud1 = {100, 'a'}; /* not allowed */
```

```
union grade stud2 = {'a'}; /* unclear */
```

```
union grade stud3;  
stud3.percent = 100; /* okay! */
```



# Arrays

- Declaration is similar to Java
- Array size must be known at compile time (size should never be specified by a variable)

```
int nums[50];  
nums[10] = 123;
```

```
/* multidimensional arrays */  
int morenums[10][10];  
morenums[0][3] = nums[1];
```

```
/* initialized array (size declared  
implicitly) */  
int somenums[] = {1, 2, 3};
```



# Strings (or lack thereof)

- A string in C is a char array.

```
char firstname[10];  
char fullname[20];
```

```
firstname = "george"; /* won't work! */  
fullname = firstname + "bush" /* won't  
work! */
```

- Null termination is important.
  - Beware: char foo[50] can only hold 49 characters because last char should be '0'.



# printf ( ), your new best friend

- Equivalent of System.out.println ( )

```
#include <stdio.h>
```

```
char name[10] = "yoda";
```

```
int age = 900;
```

```
printf("name: %s      age: %d\n", name, age);
```

**OUTPUT** → name: yoda age: 900

- Use *format specifiers* for printing variables

```
%s - string
```

```
%d - int
```

```
%f - float
```



# enum, a special type

- Used to declare and assign numerical values to a list of constants
- Start number defaults to '0'

```
enum { MON, TUE, WED, THU, FRI };  
/* now, MON = 0, TUE = 1, etc.. */
```

```
enum colors { RED, BLUE, YELLOW };  
/* name your enum and use it as a type */
```

```
enum colors mycolor = BLUE;  
/* can make statements like: if(mycolor == RED) */
```

```
mycolor = 17; /* is this allowed? */
```



## enum (cont'd)

- Can initialize your own enum values

```
enum { JAN=1, FEB, MAR, APR, MAY, JUN };  
/* now, JAN = 1, FEB = 2 etc.. */
```

```
enum { RED=3, BLUE=2, YELLOW=1 };
```

- But it could be problematic...

```
enum { CIRCLE, SQUARE, TRIANGLE=0, RECTANGLE };
```

Now...

```
CIRCLE = 0;  
SQUARE = 1;  
TRIANGLE = 0;  
RECTANGLE = 1;
```



# typedef (yay for less typing)

- Giving an alternative name to a type – especially useful with `structs`

- `typedef <type> <name>`

```
typedef int age_t;
```

```
typedef struct student {  
    char name[8];  
    int age;  
} student_t;
```

```
/* declaring variables is much easier */  
age_t myage = 12;  
student_t stud1;  
stud1.age = myage;
```



# First look at pointers

- When a variable is declared, space in memory is reserved for it.
- A pointer represents the memory address of a variable, NOT its *value*.
- **&** (address operator) gives the address of a variable
- **\*** (dereference operator) value of the memory referred to by the pointer



# Some pointer examples

```
int a, b;

a = 3;
b = a;

printf("a = %d, b = %d\n", a, b);
printf("a located at %p, b located at %p\n", &a, &b);

int *myptr;
int x = 7;

myptr = &x;

printf("x = %d, myptr points to %d\n", x, *myptr);
printf("x located at %p, myptr is at %p\n", &x, myptr);
```