

C LANGUAGE INTRODUCTION, PART 2 (INPUT VIA SCANF, WHILE LOOPS)

POINTERS, PART 1

Some ways in which C differs from Python:

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

#include instead of **import**

```
void printRootTable(int n);
```

Prototypes

```
int main() {
    printRootTable(10);
    return EXIT_SUCCESS;
}
```

Execution starts in **main**

Curly braces { } indicate start and end of blocks, instead of using indentation.

Simple statements end in **semicolon**.

```
void printRootTable(int n) {
    int k;
```

Every variable has its **type declared** when the variable is first introduced. Return type of functions is declared – **void** for does not return a value.

```
    for (k = 1; k <= n; ++k) {
        printf("%2d %7.3f\n",
               k, sqrt(k));
```

for-loops: (**init**; **while** <test>;
update at end of each iteration)

```
    }
```

printf – %d, %f, %c, %s ... \n
double quotes for strings – "go fish"
single quotes for characters – 'G'

```
}
```

Recap: Comments in C

- Python comments begin with `#` and continue until the end of the line
- C comments begin with `/*` and end with `*/`.
- They can span any number of lines
- Some C compilers (including the one we are using) also allow single-line comments that begin with `//`

Using if and else

- **if m % 2 == 0:**
 print "even"
else:
 print "odd"

- Python:
 - ▣ Colons and indenting

- **if (m % 2 == 0) {**
 printf("even");
} else {
 printf("odd");
}

- C:
 - ▣ Parentheses, braces
 - ▣ C allows you to omit the braces when the body is a single statement, but don't (leads to errors)

else if

- **if** gpa > 2.0:
 print "safe"
elif gpa >= 1.0:
 print "trouble"
else:
 print "sqrt club"

- Python:
 - ▣ Colons and indenting
 - ▣ elif

- **if** (gpa > 2.0) {
 printf("safe\n");
} **else if** (gpa >= 1.0) {
 printf("trouble\n");
} **else** {
 printf("sqrt club");
}

- C:
 - ▣ Parentheses, braces
 - ▣ else if
 - ▣ Nested if's also allowed
(as in Python)

C does not have a Boolean type

- Instead:
 - **0 means False** and
 - **any other number means True**
- So if you want a function to return True or False, instead have it return 0 (for False) or 1 (or any other non-zero number) (for True)

Boolean operators in C

- Python uses the words **and**, **or**, **not** for these Boolean operators. C uses symbols:

&& means “and”

|| means “or”

! means “not”

- Example uses:

```
if (a >= 3 && a <= 5) {  
    ...  
}
```

```
if (! same(v1, v2)) {  
    ...  
}
```

While loops


- How do you suppose the following Python code would be written in C?

```
n = 10
while n >= 0:
    n = n - 1
    print n
```

```
n = 10;
while (n >= 0) {
    n = n - 1;
    printf("%d\n", n);
}
```

- How do you break out of a loop in Python?
- How do you suppose you break out of a loop in C?
 - Use **break**, same as Python
 - Here's the loop-and-a-half pattern in C

```
while (1) {
    ...
    if (...) {
        break;
    }
    ...
}
```



To read input from user in C, use `scanf()`

```
float x;  
double y;  
int z;
```

In this use of `scanf`, user can enter the numbers separated by any whitespace (e.g. all on one line or on separate lines).

```
printf("Enter two real numbers and an integer:");  
fflush(stdout);
```



`fflush`: Pushes prompt string to user before asking for input.

```
scanf("%f %lf %d", &x, &y, &z);
```

```
printf("Average: %5.2f\n", (x + y + z)/3.0);
```

Note `%lf` in `scanf` for double's.

Note `&`'s – more on them soon.

`scanf` is not resilient – if you misuse it, the compiler will generally not complain (your compiler is better than most) but the program will crash or simply give wrong results.

`scanf` has lots of options that are powerful but perhaps confusing – see pages 355-359 of your text if you need more structured input, and meanwhile stick to the above form, with spaces between the `%`'s.

Next 30 minutes

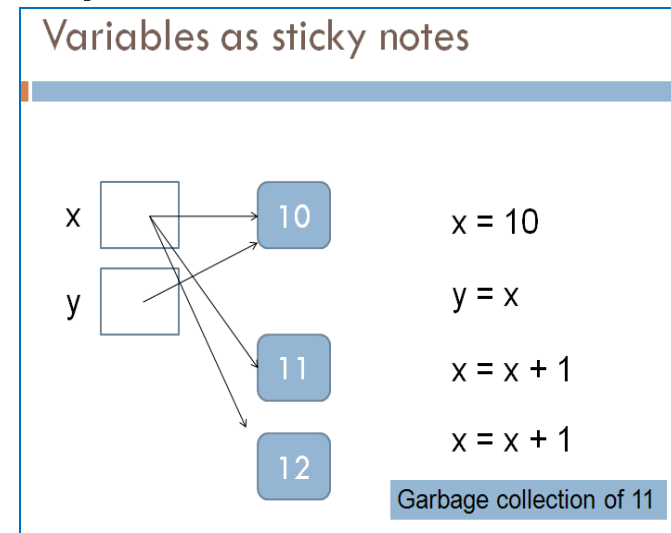
- ❑ Checkout **24-CWhileLoopsAndInput**
- ❑ Work through the TODO's, as numbered.
- ❑ Ask questions as needed!
 - ❑ Don't merely make the code "work". Make sure you understand the C notation and how to use it.
 - ❑ Pay close attention to error messages – learn how to decipher them.
- ❑ If you:
 - ❑ finish early, return to your **23-CForLoops** project and continue your work in it.
 - ❑ don't finish in class, then finish the exercise for homework

Outline

- Last time: C basics
 - ▣ Functions and variables, with types
 - ▣ For loops
 - ▣ If statements
- So far today:
 - ▣ Input, via scanf
 - ▣ While loops
- Rest of today: **Pointers**

Variables and parameter passing in Python

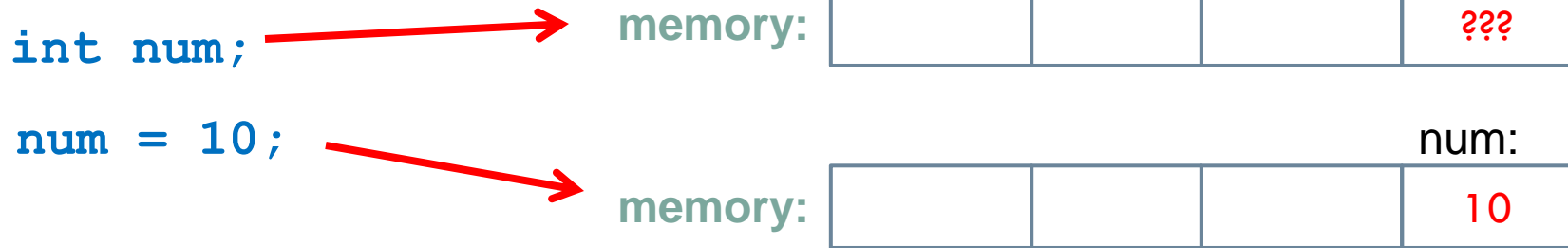
- Recall that in Python “everything is an object” and hence all variable names are **references** to objects
 - They act like sticky notes
 - When we pass a variable to a function, we are passing a reference to an object.
 - This is efficient (fast) – we copy only the reference, not all the data that is referenced. For example, when we pass a list, we pass a reference to the list, not all the data in the list.
 - If the object is mutable, we can mutate it in the function – this is convenient and efficient. If the object is not mutable, we are assured that it is unchanged when we return from the function – this makes it easier to write correct code. So both mutable and immutable objects have their place.



Variables in C

- Variables are stored in memory

- ▣ We call the place in memory the variable's *address*



- C has several types of variables:

- ▣ Integers – their bits are interpreted as a whole number
 - ▣ Doubles – their bits are interpreted as a floating point number
 - ▣ ...
 - ▣ **Pointers** – their bits are interpreted as an *address in memory*
 - As such, they are *references* to other data

The three notations for pointers in C

pNum is a **pointer** to an int

pNum is set to the **address** of num

The **thing at pNum** is set to 99

```
int num;
```

```
num = 4;
```

```
int* pNum;
```

```
pNum = &num;
```

```
*pNum = 99;
```

memory:



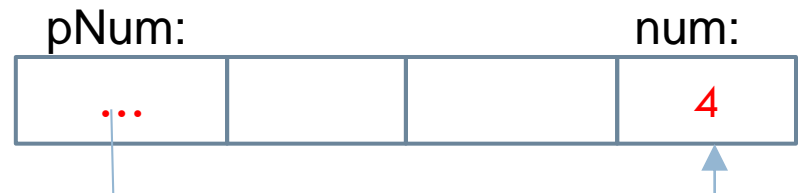
memory:



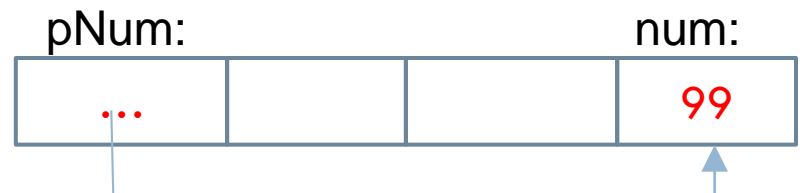
memory:



memory:



memory:



pNum is the **pointer** and **num** is the **pointee**.

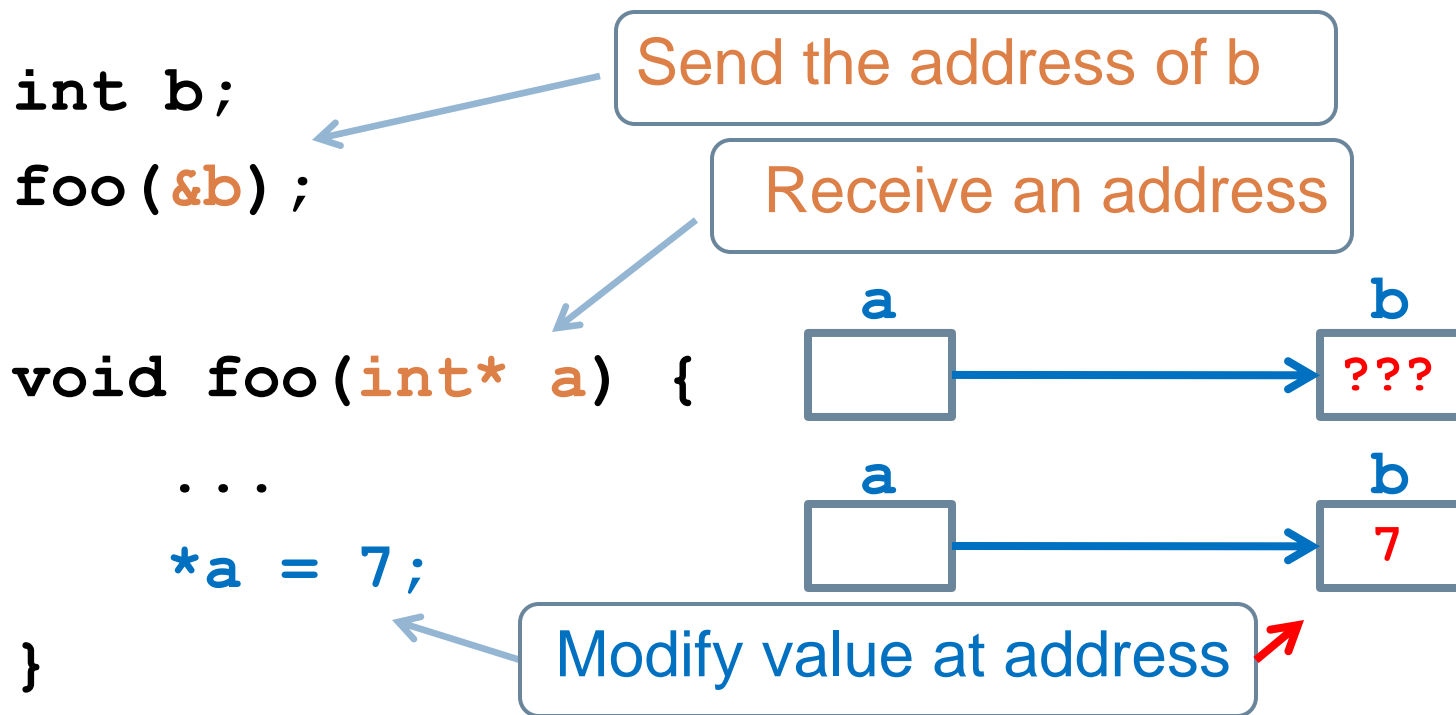
***pNum** **deferences** the pointer, which means that it obtains the pointee.

Here's Binky!

- Ignore *malloc* in the video for now
- Vocabulary
 - ▣ *Pointee*: the thing referenced by a pointer
 - ▣ *Dereference*: obtain the pointee
- See <http://cslibrary.stanford.edu/104/>
- What name did we give pointer “sharing” in Python?
 - ▣ Answer: *aliasing*

Using pointers as parameters

Box and Pointer Diagrams



Now `b` has the value 7 that was established in `foo`!

This is useful for:

- sending data back from a function via the parameters, and for
- passing large amounts of data to a function.

Thus pointers in C give us the same advantages as references-to-objects in Python.

Rest of today

- Checkout **24-CPointers**
- Work through the TODO's, as numbered.
 - ▣ We'll do the first few together
- Ask questions as needed!
 - ▣ Don't merely make the code “work”. Make sure you understand the C notation and how to use it.
- If you:
 - ▣ finish early, return to your **24-CWhileLoopsAndInput** or your **23-CForLoops** project and continue your work in it.
 - ▣ don't finish in class, then *finish the exercises for homework*
 - *Get help from the assistants in F-217 Sunday evening as needed!*