POINTERS IN C, PASSING POINTERS TO FUNCTIONS

CSSE 120—Rose-Hulman Institute of Technology

Parameter Passing Styles

- Most programming languages offer several means of passing parameters.
- We have seen pass-by-value, in which the argument is evaluated and a copy of the value is assigned to the formal parameter of the called function.
- We will now explore a second style

An Analogy

- Suppose a friend asks you for 5 bucks.
- You have at least two choices:
  1. You reach into your wallet and give your friend 5 bucks.
  2. You give your wallet to your friend and ask them to take 5 bucks.
- Most people will use the second method only with true friends.

Evaluation of the Analogy

- In the second case, the person receiving the wallet may:
  - Take 5 bucks.
  - Take all the money you have.
  - Take no money
  - Put money into your wallet

Passing by pointers - Part 1

- We will now introduce you to *passing a parameter by pointer.*
- If you think of variables as wallets, then we need to have a way of passing wallets rather than the contents of a wallet.
- If you prefer to think of variables as sticky notes, then we need to have a way of passing a sticky note rather than what is written on the sticky note.

Passing by pointers - Part 2

- We pass a parameter by pointer by adding the '&' symbol in front of the variable name.
- If you have a variable called x, then you pass it by pointer like so: &x

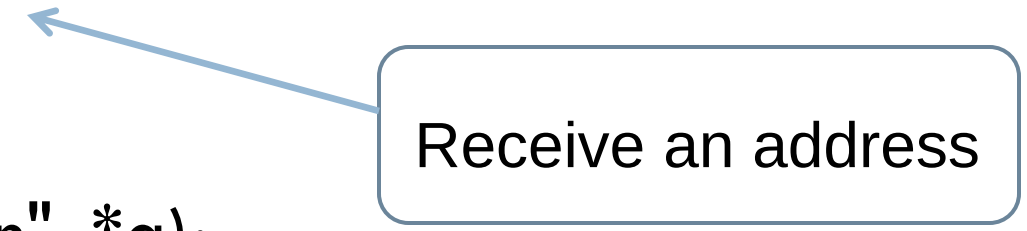
Passing by Point - Part 3

- The called function needs to know that it receives a wallet/sticky note rather than a value.
- As such, we need to tell it.
- We do this by adding the symbol '*' in front of the parameter name.
- If you have a parameter called y, then you change it to: *y

A simple example

```
void foo(int *a){
        *a = 7;
        printf("%d\n", *a);
}

int b = 3;
foo(&b);
printf("%d\n", b);
```



A Second Example

- Consider this C function:

```
void downAndUp(int takeMeHigher, int putMeDown){
    takeMeHigher = takeMeHigher + 1;
    putMeDown = putMeDown - 1;
 }
```

- Given: `int up = 5, down = 10;`
- Invoke: `downAndUp(up, down);`
- After we return from downAndUp, will the values of **up** and **down** be changed?

Second Example - Cont'd

- How do we modify downAndUp() so that it changes the values of its parameters?
- Together, implement a function that passes pointers to values to be changed
  - Use project PointersInclass that you already checked out
  - Implement **`downAndUpthatWorks()`**
  - Use function **`testdownAndUpthatWorks()`** to test **`downAndUpthatWorks()`**

Pointers

- Variables are names of memory addresses.
- Variables that we have seen so far hold values such as integers, floats, and characters.
- A **pointer** is a variable that holds the *address* of some variable.
- To continue our analogy, a pointer is a variable that holds a wallet or a sticky note.

Pointers Cont'd

- We use the "*" to indicate that a variable is a pointer.
- Examples:

```
int *aVariableThatHoldsAPointerToAnInt;
int aVariableThatHoldsAnInt = 4;
int *pNum;
int num = 6;
```

Assignments to Pointers

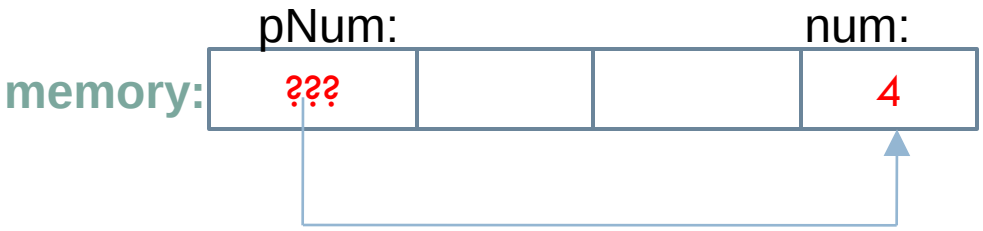
- We need to have a way of obtaining the address of a variable, rather than its contents.
- We obtain the address of a variable by using the '&' (address) operator.
- Example continued:

```
int *pNum;
int num= 4;
pNum = &num;
```

Visualizing Pointers

Box and Pointer Diagrams

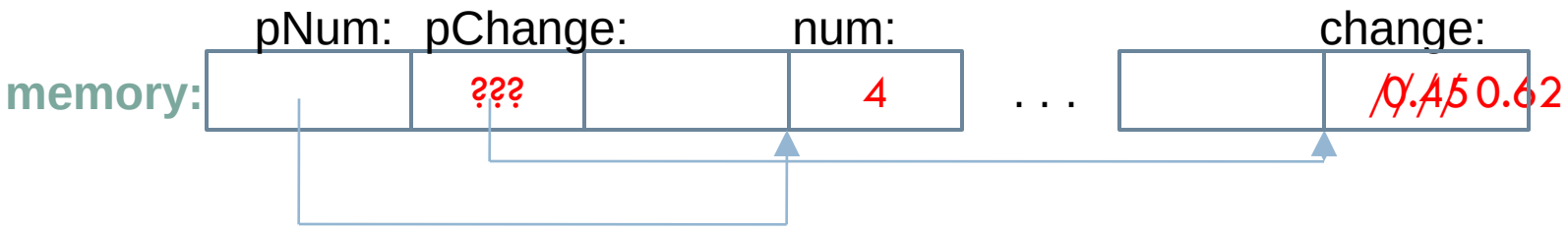
```
int num = 4;
int *pNum;
pNum = &num;
```



Both, num and *pNum are 4.

Visualizing Pointers – Part 2

```
int num = 4;
int *pNum;
pNum = &num;
double change = 0.45;
double *pChange;
pChange = &change;
*pChange = .62;
```



Summary of Pointers

- Example of a pointer variable: *pNum
- Example of a integer variable: num
- Assigning a value to an int: num = 4;
- Obtaining the address of a variable: &num
- Assigning an address to a pointer variable:

  pNum = #
- Assigning a value to the variable that a pointer variable points to:

  *pNum = 7;

Summary of Pointers

- What happens, if we do:

  pNum = 7 instead of:

  pNum = &num

- What happens, if we do:

  *pNum = &num instead of:

  *pNum = 7

Proof that Pointers are Memory Addresses

- Try the following code in the PointersInClass project:

```
printf("num = %d and is stored at %p\n", num, &num);
printf("pNum = %p and is stored at %p\n", pNum, &pNum);
```

Practice with Pointers

```
1.     int x = 3, y = 5;
2.     int *px = &x;
3.     int *py = &y;
4.     printf("%d %d\n", x, y);
5.     *px = 10;
6.     printf("%d %d\n", x, y);  /* x is changed  */
7.     px = py;
8.     printf("%d %d\n", x, y);  /* x not changed */
9.     *px = 12;
10.    printf("%d %d\n", x, y);  /* y is changed  */
```

Break

- Starring **Binky**!
- (See http://cslibrary.stanford.edu/104/)

Pointer Pitfalls

- Don't try to dereference an unassigned pointer:
  - ```
    int *p;
    *p = 5;
    ```
  - ```
    /* oops! Program probably dies! */
    ```
- Pointer variables must be assigned *address* values.
  - ```
    int x = 3;
    int *p;
    p = x;
    ```
  - ```
    /* oops,  RHS should be &x */
    ```
- Be careful how you increment
  - ```
    *p += 1;      /* is not the same as … */
    ```
  - ```
    *p++;
    ```

In-class exercise on pointer pitfalls

- The rest of today's quiz lets you see some pointer pitfalls in action. These make great exam questions!
- Do it now

- When you are done, start the homework:
  - More pointer output
  - Writing functions to change variables
    - doubleMe
    - swap
  - scanf revisited