Questions about HW4

Recap from Lab 7

- Writing a README and comments
- Function prototypes (but I am still not sure everyone gets it)
- Preprocessors
  - #include
  - #define
- Bit Operators
- Debugging
Recap from Lab 8

- preprocessors
- struct
- union
- typedef
- enum

Pointer Basics

- A pointer is a variable in C that contains a memory location.
- Pointers are used in programs to access memory and manipulate addresses.
  - We have already seen it briefly in scanf() where usage was scanf("%d", &v);

Pointer Basics II

- Declaration
  - int *p;
  - This creates ‘p’, which is of type “pointer to int”
  - The legal range of values for any pointer always includes the special address 0 and a set of positive integers that are interpreted as machine addresses on the system
- & is used to “point to” the address of a variable
  - This is used to dereference a variable’s memory location
  - Officially - & is an operator that retrieves the memory address of a variable
Pointer Basics III

- Examples
  - `p = &i;` // `p` has the memory location of `i`
    // therefore `*p` points to `i`
  - `p = 0;` // shows assignment of `p` to 0
  - `p = NULL;` // same as `p = 0;`
  - `p = (int *) 1307;` // `p` now has an absolute
    // address in memory
    // We do this by using a cast
    // This is typically not done, why?

Pointer Basics IV

- Typical example (ptrexample0.c)

```c
int var; // Declare an integer var
int *p; // Declare p as a pointer to an integer
var = 4; // Set the value of var to be 4
p = &var; // Set p to be the address of var
printf("%d", p); // Is this accurate?

*p = 5; // Sets the value of the thing p is pointing to, to 5
p = 5; // What will this do?
```

Pointer Addressing/Dereferencing

```c
int a, b;
int *p;
a = b = 7;
p = &a;
printf("%d\n", *p); // What is printed?
*p = 3;
printf("%d\n", a); // What is printed?
```
**Pointer Addressing/Dereferencing**

\[ \text{p} = \&b; \]

\[ \text{*p} = 2 \times \text{*p} - \text{a}; \]

```
printf("b = %d\n", b); // What does this print?
```

---

**& and * relationship**

- Simply put, the dereference operator (*) is the inverse of the address operator (&).
- `double x, y, *p;`
- `p = \&x;`
- `y = \*p;`

// Here, p is assigned to address of x. Then y is assigned to the value of object pointed to by p
- `y = \*\&x;`
- `y = x;`

// How do these two statements relate to the above two?

```
(ptrexample1.c)
```

---

**Multiple pointers can point to one location**

```
int something;
int *first_ptr;
int *second_ptr;
something = 1;
first_ptr = \&something;
second_ptr = first_ptr;
```

---

---

---
Convince yourself

<table>
<thead>
<tr>
<th>Declarations and Initializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i=3, j=6, *p=&amp;i, *q=&amp;j, *r;</td>
</tr>
<tr>
<td>double x;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>p == &amp;i</td>
<td>p == (&amp;i)</td>
<td>1</td>
</tr>
<tr>
<td>p = i + 7</td>
<td>p = (i+7)</td>
<td>illegal</td>
</tr>
<tr>
<td>* * &amp; p</td>
<td>(* (&amp;p))</td>
<td>3</td>
</tr>
<tr>
<td>r = &amp;x</td>
<td>r = (&amp;x)</td>
<td>illegal</td>
</tr>
<tr>
<td>8 * * p / * q + 7</td>
<td>(((8 * (* p)) / (* q)) + 7</td>
<td>11</td>
</tr>
<tr>
<td>* (r &amp; j)</td>
<td>*= &quot;p</td>
<td>18</td>
</tr>
</tbody>
</table>

Call by reference

- Pointers can be used as function arguments
- We have been typically using call by value
- Remember the swap function

```c
#include <stdio.h>

int swap (int a, int b);

int main () {
    int x=3, y=7;
    printf("%d %d\n", x, y);
    swap (x,y);
    printf("%d %d\n", x, y);
    return 0;
}

int swap (int a, int b) {
    int tmp;
    tmp=a;
    a=b;
    b=tmp;
    return a;       // I can return only one value, what do I return?
}
```

Call by reference II

- Note that the call-by-value has problems in that only the method’s local values are affected.
- Therefore we need something else
  - Pointers to the rescue
  - We call other functions and pass parameters by reference
  - New code looks like
Call by reference III

#include <stdio.h>

int swap (int *, int *);

int main() {
    int x=3, y=7;
    printf("%d %d\n", x, y);
    swap (&x,&y);
    printf("%d %d\n", x, y);
    return 0;
}

int swap (int *p, int *q) {
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}

//ptrexample3.c

Call by reference IV

- Another example

#include <stdio.h>

void inc_count (int *count_ptr)

int main () {
    int count = 0;
    while (count < 10)
        inc_count(&count);
    return 0;
}

void inc_count(int *count_ptr) {
    (*count_ptr)++;
}

Assignment

- Read Ch. 13 from the Practical C Programming book
- HW4