YOU MAY FIND IT HELPFUL TO DRAW OUT THE PROGRAMS LIKE I DID IN CLASS

For the first part below, I'm making two different headers-one for recursive and one for iterative versions of the same function.

Note that one of the main advantages of headers is that we don't pay attention to the implementation (abstraction). That means it would be more realistic to have one header with a function named one thing and wouldn't care about the implementation (meaning we could have either implementation inside and just call the function name).

No makefile:

```
1100(base) Computers-Air:C computer$ gcc practice.c functions_iterative.c functions_recursive.c
(base) Computers-Air:C computer$ ./a.out
5
5

120
120

1100
1100
```

Makefile:

```
1100(base) Computers-Air:C computer$ make
(base) Computers-Air:C computer$ ./a.out
5
5

120
120

1100
1100
```

makefile

```
CC=gcc

recursionmake: functions_iterative.c functions_recursive.c
    @ $(CC) main.c functions_iterative.c functions_recursive.c
```

main.c

```
#include <stdio.h>
#include "functions_recursive.h"
#include "functions_iterative.h"

int main(int argc, char **argv)
{
    printf("%d\n", fib_rec(5));  //in functions_recursive
    printf("%d\n\n", fib_iter(5));

    printf("%d\n", factorial_rec(5));
    printf("%d\n\n", factorial_iter(5));

    dec_binary_rec(12);
    printf("\n");
    dec_binary_iter(12);
}
```

functions_recursive.h

```
#ifndef FUNC_REC /*guards*/
#define FUNC_REC
int fib_rec(int n);
int factorial_rec(int n);
void dec_binary_rec(int num);

#endif
```

functions_recursive.c

```
#include "functions_recursive.h"
#include <stdio.h> /*don't forget to include this since we use printf in here*/

int fib_rec(int n) /*location is n*/
{
    if(n==1 || n==2)
    {
        return 1; /*one of the base cases*/
    }

    int stuff=fib_rec(n-1)+fib_rec(n-2);
    return stuff;
}

int factorial_rec(int n)
{
    if(n==1)
    {
        return 1;
    }
    int num= n*factorial_rec(n-1);
    /*printf("value returned: %d\n", num);*/ /*you can add this to see the values as the
function moves along*/
    return num;

}


void dec_binary_rec(int num) /*notice you could have this return 0 or 1 instead of
printing*/
{
    if (num == 0)
    {
        return ;
    }

    dec_binary_rec(num / 2);
    printf("%d", num % 2);
}
```

functions_iterative.h

```
#ifndef FUNC_ITER /*guards*/
#define FUNC_ITER
int fib_iter(int n);
int factorial_iter(int n);
void dec_binary_iter(int num);
#endif
```

functions_iterative.c

```
#include "functions_iterative.h"
#include <stdio.h> /*don't forget to include this since we use printf in here*/

int fib_iter(int n)
{
        int i;
        int f1 = 0;
        int f2 = 1;
        int fi=0;

        if(n == 0)
            return 0;
        if(n == 1)
            return 1;

        for(i = 2 ; i <= n ; i++ )
        {
            fi = f1 + f2;
            f1 = f2;
            f2 = fi;
        }
        return fi;

}

int factorial_iter(int n)
{
    int cur=1, i=0;

    for(i=2;i<=n;i++) /*so if n=2, this wont execute*/
    {
        //printf("%d, %d\n", cur, i);
        cur*=i;
    }

    return cur;
}

void dec_binary_iter(int num)
{
    int number[50];
    int i=0;

    while(num>0)
    {
        number[i]=num%2;
        num=num/2;
        i++;
    }

    i--; /*since i is one higher than we want*/

    while(0<=i) /*print out numbers backwards*/
    {
        printf("%d",number[i]);
        i--;
    }

}
```