Instructions

There are two deliverables for this assignment; both are IA32 assembly programs: `fib.s` and `swap.s`.

Group Work

Please commit the `.s` files to GitHub and submit via Gradescope. Only one member of each pair needs to do this, though do remember to add the other person's name to the submission. You are expected to practice pair programming, where both partners are working together on a shared screen to complete this assignment. See the explanation of `pair_programming.pdf` in the syllabus for more-detailed guidelines.

Work Location

Some of the details of this assignment depend critically on the version of the `gcc` being used. It is therefore *strongly* recommended that you work on this homework while connected to the Watson 132 lab to ensure that the compiler you are using behaves as expected. You should clone a copy of your repository on your Watson 132 computer, in addition to any copies you clone on your own computer(s). That way you can push to GitHub from whatever computer you are coding on, and then pull from your repo on the Linux computer to test.

Writing Simple Functions in IA32

In this assignment, you will write two simple functions in IA32 assembly. The first function, in `fib.s` takes an integer N and uses a loop to compute the N^{th} Fibonacci number. The Fibonacci sequence starts with 1,1, then each subsequent element is the sum of the two previous elements. The following table shows the first few Fibonacci numbers:

`n`	1	2	3	4	5	6	7	8	9	10	11	12
`fib(n)`	1	1	2	3	5	8	13	21	34	55	89	144

The second function, in the file `swap.s` takes in two memory addresses and exchanges the integers at those two locations.

In C, the function signatures are as follows:

```
int fib(int n);
void swap(int *x, int *y);
```

Note that the `fib` function's argument is an integer, while the `swap` function's arguments are pointers to integers. For `swap`, this means that the parameters each store an address, pointing to the storage location of an integer. See below for more information about pointers.

Both `fib.s` and `swap.s` already contain some IA32 instructions that set up up the function's memory scope when called and tear that memory down upon returning. You will add IA32 instructions to complete the body of these functions.

In `mainfib.c` and `mainswap.c` is code to test your implementations. Run `make` at the command line to build the executables, `mainfib` and `mainswap`, that link in your assembly code and call it. Then run these executables to test your solutions for correctness:

```
$ make
$ ./mainfib 12
$ ./mainswap -2 2
```

Fib Requirements

The job of the `fib` function is to find the N^{th} Fibonacci number. In C, the `fib` function might look like this:

```
int fib(int n) {
  int prev = 0, curr = 1;
  for(int i=1; i < n; i++) {
    int next = curr + prev;
    prev = curr;
    curr = next;
  }
  return curr;
}
```

The C code in `mainfib.c` calls your `fib` function. The assembly code that the compiler generates will expect to find the function's return value in the `%eax` register. This means your function needs to ensure that return value is placed in the `%eax` register at before the assembly instructions `leave` and `ret`. The argument `n` is located in memory at `8(%ebp)`.

For the `fib` function you may use any of the general-purpose IA-32 registers to store data, but *you may not store any data in memory* (including modifying the argument n). The general-purpose registers are `%eax`, `%ecx`, `%edx`, `%ebx`, `%esi`, and `%edi`.

Swap Requirements

The job of the `swap` function is to exchange the integers stored at two given memory locations. In C, the `swap` function might look like this:

```
int swap(int *x, int *y) {
  int temp;
  temp = *x; //copy the value x points to into temp
  *x = *y;   //copy the value y points to into the location x points to
  *y = temp; //copy the value in temp into the location y points to
}
```

In this function, `x` and `y` are variables of type pointer-to-int. Pointer variables point to data by storing the memory address where data is located. Pointers of type `int*` specifically hold addresses where integer data is stored.

The result of executing the `swap` function is to exchange the values in the memory locations pointed to by `x` and `y`. The C code in `mainswap.c` calls your swap function. A call to swap would look like this (see `mainswap.c` for another example):

```
int a, b;
a = 10;
b = 8;
printf("%d %d\n", a, b);  // prints: 10 8
swap(&a, &b);             // swap the values stored in a and b
printf("%d %d\n", a, b);  // prints: 8 10
```

The variables `x` and `y` are stored at `8(%ebp)` and `12(%ebp)` respectively. However, note that since both variables are pointers, these locations actually store other memory addresses where the integer values are actually located, so you will need multiple memory accesses to actually bring the integers into CPU registers.

For the `swap` function *you may only use the first three general-purpose registers* (`%eax`, `%ecx`, and `%edx`), and you may not modify any memory locations other than the two you are

exchanging. It may be easier to first implement a draft version where you use four registers (adding `%ebx` to the above), but then you should think about how you can re-use some of the first-three registers to eliminate the need for the fourth.

Further Tips and Requirements

- **Briefly comment each of your IA32 instructions with what they are doing**. Feel free to explain them in relation to the C code. As an example:

```
movl 0x8(%ebp), %eax   # load n into register %eax
addl $1, %eax          # increment %eax
```

- Implement and test incrementally. For example, in `swap`, first see if you can set the value pointed to by `x` to the value pointed to by `y`. Once that works, then see if you can complete the function.

- You can also use gdb to debug your solution. Set a `breakpoint` in `fib` or `swap`, and use `disass`, `ni`, `info registers`, etc. to see values as you go. Look back at the reading, video, and in-class exercises for more about using `gdb` for IA-32.