1. Some of the code fragments below have a problem. For each fragment indicate whether the code works as intended or whether there is an error (logical error, compile-time error/warning, or runtime error). Assume all programs are compiled using the C99 standard. For this question, we'll assume programs which do not terminate are errors as well. If there is an error in a fragment, explain **briefly** what is wrong in the box. We have intentionally omited the error checking of the system calls to simplify the examples. Do not report this as an error.

   (a)
   ```
   int x = 5;
   // checking whether x equals 6
   if ( x = 6 ) {
           printf("x equals 6\n");
   }
   ```

   ☐ Works as intended ☒ Error
   The `( x = 6 )` statement, does not check if `x` is `6`, but rather assigns `x` to `6`, so `"x equals 6\n"` is always printed.

   (b)
   ```
   int x;
   //reading a value for x
   scanf("%d", &x);
   ```

   ☒ Works as intended ☐ Error

   (c)
   ```
   struct student {
           int age;
           char *name; }
   // Increase the age of a student by amt.
   void increase_age(struct student s, int amt) {
           s.age += amt;
   }
   int main() {
           struct student rob;
           rob.age = 10;
           increase_age(rob, 5);
           printf("%d should be 15\n", rob.age);
   }
   ```

   ☐ Works as intended ☒ Error
   In `increase_age`, local changes are made to student `s`, so they are not reflected in `main`, therefore `rob`'s age will still be `10`.

(d)
```
char * st = malloc(31);
// reading a string into st
// you may assume that not more than 30 characters are read
scanf("%s", &st);
```

☐ Works as intended ☒ Error
`scanf` gets a pointer to a `char` pointer. `scanf("%s", st)` should be used instead.

(e)
```
#include <string.h>
int main()
{
        char * st;
        // copying "abc" into st
        strcpy(st, "abc");
        return 0;
        Winter 2017
}
```

☐ Works as intended ☒ Error
`st` does not point to any allocated memory for a `char` array so `strcpy` should fail.

(f)
```
char st1[] = "abc";
char st2[] = "abc";
if ( st1 == st2 )
printf("Strings are identical");
else
printf("Not identical");
```

`( st1 == st2 )` is only true when both pointers point to the same address. Even though both arrays have the same string, "`Not identical`" would be printed. `(strcmp(st1, st2) == 0)` should be used instead to check if they are identical.

2. ***I didn't do this one but here it is:***
In this question you will write a function hash that will take as arguments the name of a file filename and a hash block size blocksize. It will then read the contents of file filename byte by byte and compute (and return) a hash with the size specified in blocksize. The hash you implement should be based on xor and your function should be able to handle text files as well as binary files. You can assume that blocksize passed to the function is a valid block size and that no error occur during any of the system calls you might make. You can also assume that any libraries you need have been included.

```
char *hash(char *filename, int blocksize) {









}
```

3. Suppose you want write a program with two processes that communicate through a pipe. Decide for each of the following statements whether they are correct or not:

You need to call pipe() before you call fork().

☒ True ☐ False

You need to call fork() before you call pipe().

☐ True ☒ False

You need to call pipe, but you don't necessarily have to call fork.

☒ True ☐ False

Pipes are uni-directional, which means that only the parent can read and only the child can write.

☐ True ☒ False Child can read and parent write instead.

It is important to close the unused end of a pipe because otherwise the read end of the pipe will block STDIN.

☐ True ☒ False Read end will block process until all writing ends are closed.

It is important to close the unused end of a pipe because otherwise the tobacco will spill out of the pipe ends left open.

☒ True ☐ False

A read call will block if the pipe is empty.

☒ True ☐ False

A write call will block if the pipe is full.

☒ True ☐ False

A child inherits all the open file descriptors from the parent, including those corresponding to pipes.

☒ True ☐ False

4. Consider the following makefile:

```
FLAGS = -Wall -std=gnu99
DEPENDENCIES = hash.h ftree.h

all: fcopy

fcopy: fcopy.o ftree.o hash_functions.o
gcc ${FLAGS} -o $@ $^

%.o: %.c ${DEPENDENCIES}
gcc ${FLAGS} -c $<

clean:
rm *.o fcopy
```

Assume the directory that contains the makefile contains the following files (and no others):

hash.h ftree.h ftree.c fcopy.c hash functions.c.

Suppose the following commands are run one after the other. Fill in the table to show what files are created, deleted or modified as a result of running each command. If no files are created, deleted or modified after a particular command, write "NO CHANGE".