CSC209H Worksheet: Function Pointers and System Call Error Checking

1. Remember that we can use the name of a function as the pointer to the function. This allows us to create variables that are pointers to functions. The syntax can be a little confusing.

   For the statements below, identify whether the statement is A) a function signature, B) declaration of a function pointer variable, C) assigning the return value of a function to a variable, or D) assigning a pointer to a function to a variable.

   Then label the relevant parts of the statement: variable name, return value, argument(s). Explain to your neighbour what each line means.

A (a) `int simple(char *str, int length);`

return type; ↳ function name; ↳ parameters

B (b) `int (*x)(char *s, int l);`

variable name. Type of x is a pointer to a function that takes two arguments - one that is char *, and one that is int, and returns an int

C (c)
```
int z;
z = simple("abc", 30)
```

call simple and store its return value in z

D (d) `x = simple;` use the name of a function as a pointer to the function

Now we can use x the same way as simple `int q = x("def", 25);`

A (e) `int (*complex(int index))(char *s, int l);`

complex: Name; argument; return type

complex is a function that takes an int as an argument and returns a pointer to a function

B (f) `int (*y)(char *s, int z) = complex(2);`

var. name; variable type; ↳ call complex which returns a pointer to a function

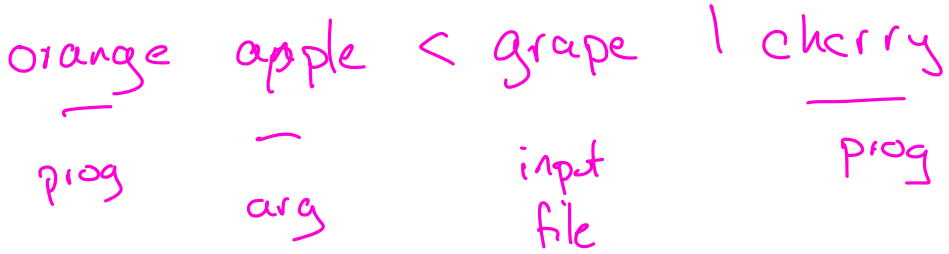
+C

2. Add the error checking for the following calls. Discuss with your neighbour what the possible errors from the following system calls. (Feel free to cheat by reading the man page.) How important is it to check for errors? Should the program exit immediately?

```
FILE *fp;
fp = fopen(argv[1], "r");
if (fp == NULL) {
    perror("fopen");
    exit(1);
}
int num;  int result;
result = fread(&num, sizeof(int), 1, fp);
if (result == 0) {
    if (ferror(fp)) {
        // may be exit?
    } else {
        // we have finished processing the
        // file so continue with the program
char *str;
str = malloc(sizeof(char) * 1024);
if (str == NULL) {
    perror("malloc");
    exit(1);
    // not much you can do when you
    // are out of memory
}
```

Remember errno is a global variable

perror prints a message based on the value of errno




```
qsort( void *list , compare )

int compare(void* a, void*b) {
    int x = (int) a
    int y = (int) b
    return (x > y)
```

System Calls + Error Checking

- A system call requests a service from the OS
- fgets, fopen, fread etc are not system calls, but they do make system calls (open, read...)
- System calls return -1 or NULL on error and set errno.
- errno - integer code for the cause of the error
- ALWAYS check return value of system calls (and relevant library calls)!!!