Standard C Library:

#include <stdio.h> <stdlib.h> // general purpose
    <string.h> //String: strcpy, strcat, strlen, strcmp (true=0), strchr, strstr
    <unistd.h> <fcntl.h> <stdlib.h> // Syscall
    <sys/types.h> // process: fork, exec, execl
    <sys/wait.h> // wait (), wait(&status)
    <pthread.h> // Thread library

Format specifier:

%i or %d = **int**
%d = **boolean** [true=1, false=0]
%c = **char** %f = **float** %s = **string**

#Note for input: have to add &variable
scanf ("%d %f %c", **&i, &f, &c**);

System Calls:

int **open** (file_name, mode);
mode=O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, __O_LARGEFILE, O_TRUNC
eg. int dest_file = **open** ("./out.txt", O_CREAT | O_RDWR | __O_LARGEFILE | O_TRUNC, 0600);

read/write/lseek: (#include <unistd.h> #include <fcntl.h> #include <stdlib.h>)
int fd;
char buffer[num];
unsigned transfer_size; // transfer_size = **sizeof**(buffer) or **strlen**(buffer)

read (fd, buffer, transfer_size); **write** (fd, buffer, transfer_size); **close**(fd);
lseek (fd,5, SEEK_SET); //+5 forward, -5 backward
  // SEEK_SET from beginig, SEEK_CUR from current pointer position
# Get the current location of the file descriptor
  int current_location = **lseek** (source_file, 0, SEEK_CUR);
# Move the file descriptor to the start of the file: **lseek** (source_file, 0, SEEK_SET);
# Move the file descriptor to the end of the file: **lseek** (source_file, -1, SEEK_END);
# Get the length of the file: off_t fileLength = **lseek** (source_file, 0, SEEK_END);

Process: #include → <stdio.h> <unistd.h> <sys/types.h> <sys/wait.h>
pid_t pid = **fork** ();
int id = **getpid** (); **#Note:** be careful of this one you forget it exists!
wait (&status);
int **execl** (const char *path, const char *arg, ..., NULL);
  eg. int ret = **execl** ("/bin/ls", "ls", "-1", (char *)0);

fork e.g.
```
int parent () {
    pid_t pid;
    pid = fork ();
    if (pid == 0) {
        child ();       }
    else if (pid > 0) {
        int status;
        wait(&status);
```

```
                printf ("I am parent process. %d\n", getpid () ); }
        else {
                printf ("Forking failed.\n");
                exit (EXIT_FAILURE);} }
```

excel e.g.:

```
Program 1:
#include<stdio.h>
int main (int argc, char* argv []) {
    printf (“Program-1 arguments passed: %d”,
argc);
    for (int i=0; i<argc; i++) {
        printf (“%s”, argv[i]); } }
```

```
int main (){
    printf (“Program-2 Running...”);
    pid_t pid, status;
    pid = fork ();
    if (pid == 0) {
    execl (“home/john/Desktop/”,
“program1”, ‘a’, ‘b’, ‘c’, ‘d’, NULL);}
    else if(pid>0) {
        wait(&status);
        execl (“bin/pwd/”,“pwd”, NULL); } }
```

Thread:

```
        pthread_exit(&i);
        pthread_create (&thread_id, NULL, FunctionName, argument);
        pthread_join (thread_id, (void**) &ptr); → printf ("%i\n", *ptr);
```

# Multithreading without parallel execution {put inside main}

```
        pthread_t thread [3];
        for (int i=0; i<3; i++) {
                pthread_create (&thread[i], NULL, block, NULL);
                pthread_join (thread[i], NULL);         }
```

# Creating multiple threads using loop with parallel execution {put inside main}

Same as before except create and join in separate loops

# Return - pthread_exit () and argument passing- *arg

```
void* block (int *n) {
    printf ("Entered the Thread.\n");
    if (*n%2==0) {
        pthread_exit("odd");
    } else {
        pthread_exit("Even");} }
int num=10;
void* thread_return;
```

```
int main () {
    pthread_t thread1;
    pthread_create (&thread1, NULL, (void*) block, &num);
    pthread_join (thread1, &thread_return);
    printf ("Thread returned; %s\n", (char*) thread_return);
    return 0;     }
```

Note: for integer return:

```
void *block (int *n) {... int number [3];
        pthread_exit ((void **) &number[*n]);}
void main () {
        pthread_create(&t[i], NULL, (void *) block, &i);
        pthread_join(t[i], &x);
        valx = *(int *) x;)         }
```