**C**

**What Is C?**

* C is versatile (easy to adapt) due to its design.

**Input & Output**



* For giving character input.



* Output for the same.

******

* **scanf()** stops scanning when a space appears.



* Putting gaps between two won’t affect, but putting anything else will.
* **\’** – single quote
* **\”** – double quotes
* **\\** - backslash
* **\b** – backspace
* **%e** – scientific notation (exponent)



* %3.2 = %-3.2
* In %3.2, 2 is number of digits after decimal point.

**Conditionals**

* A **non-zero** value inside parenthesis of conditional statement is considered **true**.

**Logical Operators**

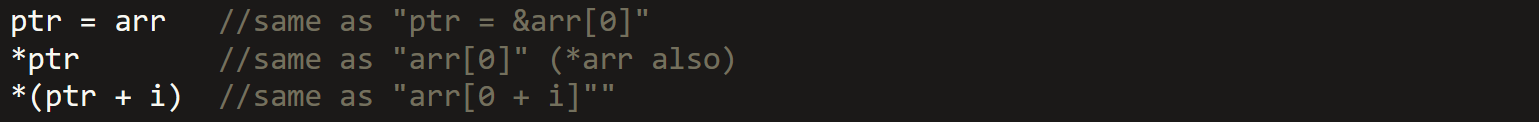
* Boolean expressions are evaluated from **left to right**.
* ***if (!(n==’x’ || n==’x’))*** [Remember these kind of expressions.]

**do-while Loop**

* We add semicolon at the end of do-while block.

**Pointers**

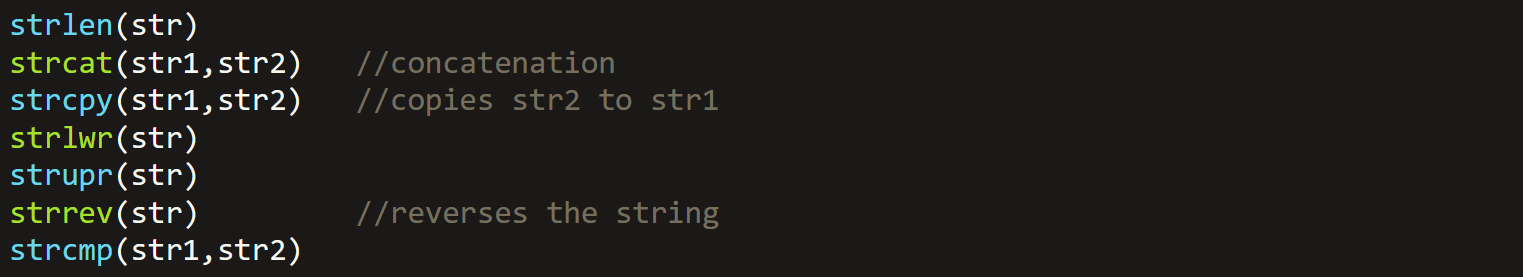
* Memory addresses are saved as **hexadecimal** values.
* **\*** is called indirection/ **dereference** operator.
* **Dereferencing**: Process of checking what a pointer is pointing to.
* We use parentheses around ***(\*ptr)++*** unary operations.
* We use pointers for arrays in a weird way:



* **ptr + i = ptr + i\*sizeof(array\_data\_type)**
* It is good practice to use **%u** for addresses.

**String**

* Ends with a **NULL** character **‘\0’** always.
* It is recommended to declare the string array size **one character longer**.
* Can be declared as an array of characters, just **explicitly add ‘\0’** as last element.
* Like an array, name of a string acts as a **pointer**.
* A string pointer is **constant** and thus can’t be changed.
* We access any memory block and fill it with data.

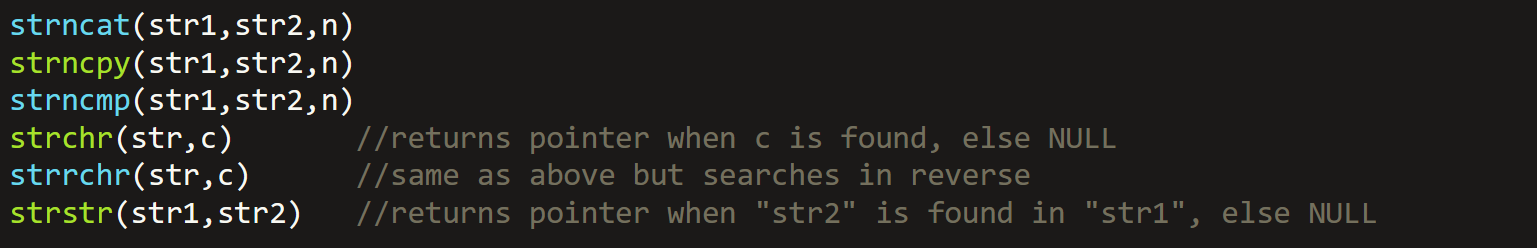


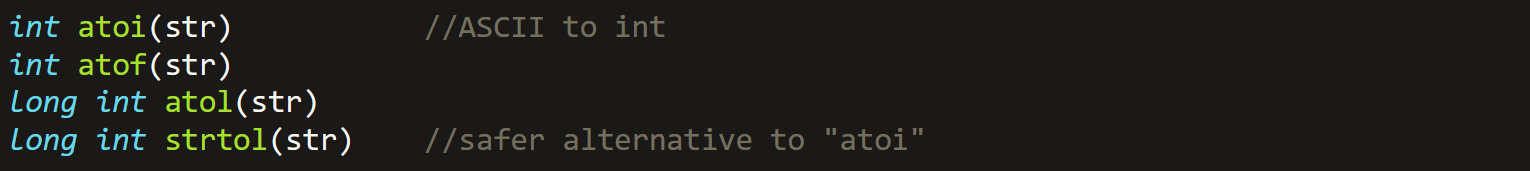
* String in **C** can’t be assigned a value using **=.** So, we use **strcpy()**.
* **strcmp()** returns **0** when both strings **match** & **1** when they **don’t**.
* **scanf()** stops reading a string when space is encountered, unlike **gets()**.



* **sprint** and **sscanf** are used for formatted strings.

**String Functions**





* In **atoi()** and **atof()**, str must be a string and **not** char.
* **atof()** returns pointer to **str**.
* **strtol()** is still a mystery.
* Array of strings is basically **2D** array of characters.
* But when we create a pointer to array of strings, we have to define array as **1D**.

**Function Pointers**

* Void pointers to functions are like:-



* ***\*Mistake above\****, a void function **can’t** have **return** statement.
* Function name also acts as a **pointer**.
* You **can’t** print function answer with **pointer name**, but can **perform** it with it.
* Replacing **conditional statements** with pointers:-



* It is a function pointer’s example.

**Void Pointer**

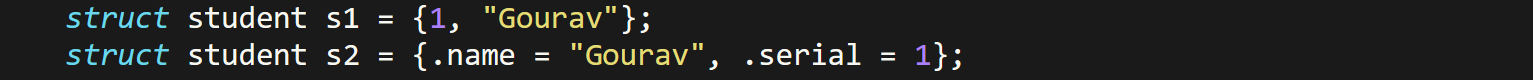
* Type casted as **(int\*)ptr** and referred as **\*((int\*)ptr)**.
* Used the same way as a **normal** pointer.
* Function taking and returning void pointers:-



* Here above, function pointer **(void \*)** used as a parameter is referred as a **callback function**.

**Structure**

* Variables inside struct are called its members.
* Structure ends with a semicolon.
* Structure is also called **composite/aggregate** data type.

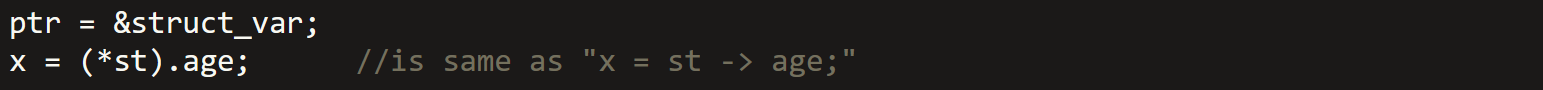


* We access structure members just like class members, by using **dot**.
* Member values transfer:-



* Using **typedef** for struct makes it **unnecessary** to give it name, except for cases where we define pointers to struct itself.

**Working With Structures**



* There can be array of structures.



**Unions**

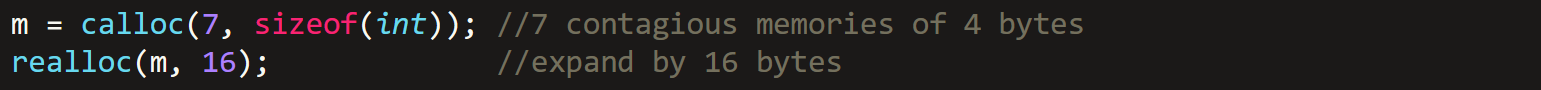
* Its memory location is occupied by only one member at a time.
* Thus accessing another member’s memory location at that time is meaningless.
* Union inside a structure must be defined as:-



**Working With Unions**

* For unions also -> is used to access its members through pointers.
* Union operations are **acting weird**, not responding & returning errors.

**Working With Memory**

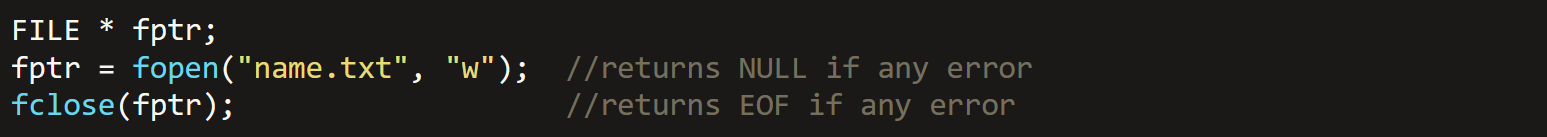


**calloc And realloc**

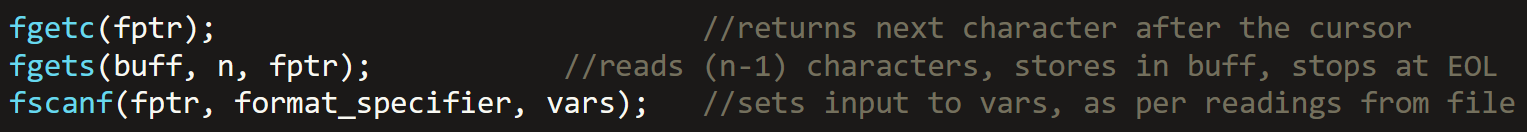
* **calloc()** function allocates memory as per the size of a particular item, and not any size of bytes.

**Working With Files**

* **<stdio.h>** library includes file functions.
* FILE is a default typedef for file pointer.
* Mode options for file: **r, w, a, r+, w+, a+**
* For **w & a**, file **need not exist**.

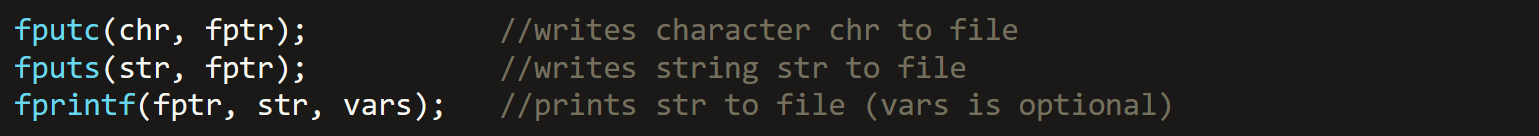


* **Buffer:** A char array used for temporary storage of file data.

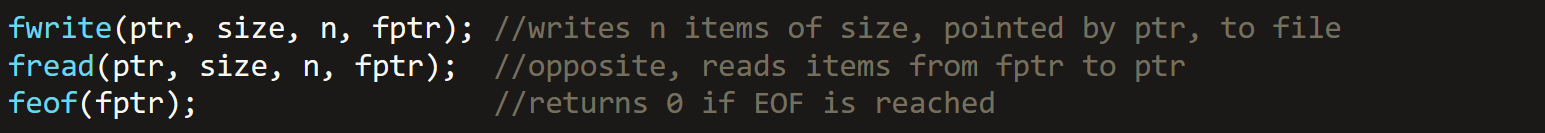


* Functions above **must be** used with **printf()**.
* Default cursor position for **“r”** mode is at starting.
* And for **“a”** its at the end.
* **“w”** overwrites the text files permanently.

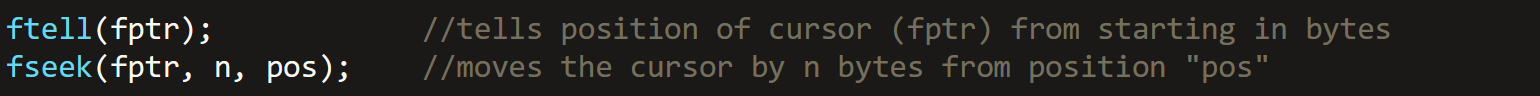
**Binary Files I/O**



* Binary mode options: **rb, wb, ab, wb+, rb+, ab+**
* Usage of **put & get** and **scanf & printf** are swapped in **file handling**.



* **fwrite()** and **fread()** is for array to file or vice versa generally.
* Actually **fread()** returns **integer** which tells **how many bytes** has been read.
* Giving irrelevant size to **fwrite()** or **fread()** results in some **4-digited** numbers being stored in the file.
* File types:-
  + **.txt** = text
  + **.bin** = binary
  + **.csv** = comma separated values
  + **.dat** = data file (can be opened in binary mode)



* position **pos** can be:-
  + **SEEK\_SET** start of file
  + **SEEK\_CUR** current position
  + **SEEK\_END** end of file

**Error Handling**

* **EXIT\_FAILURE** exit because of failure
* **EXIT\_SUCCESS** exit because of success

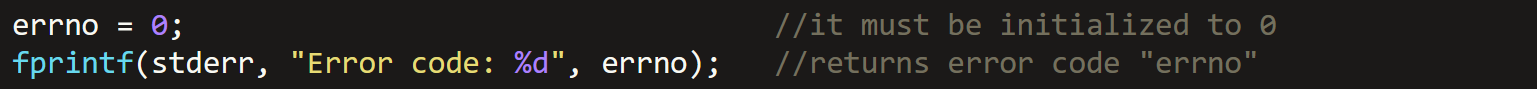


* Exit function is part of ***<stdlib.h>***.

**Using Error Codes**

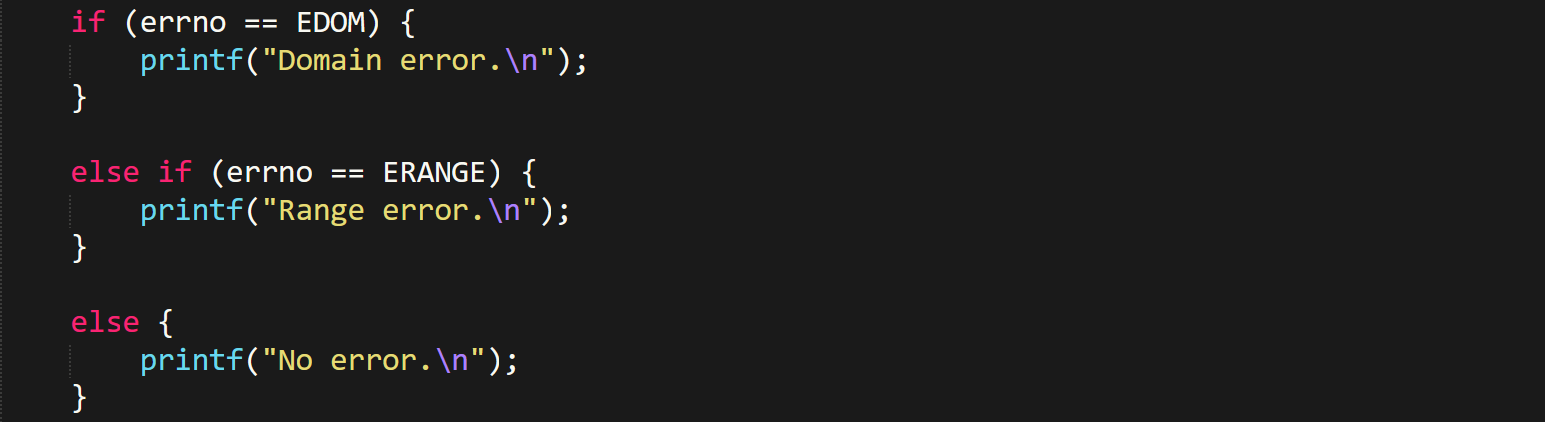
* Are used for files only.



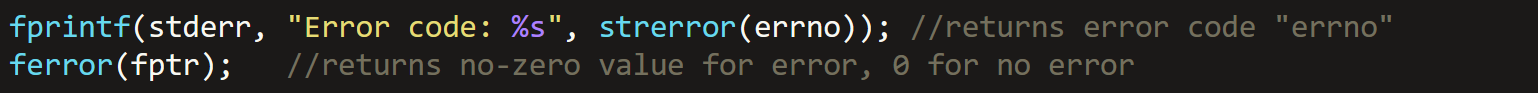




* **perror()** prints the message along with **error cause**.
* Importing **<math.h>** and using error codes:-

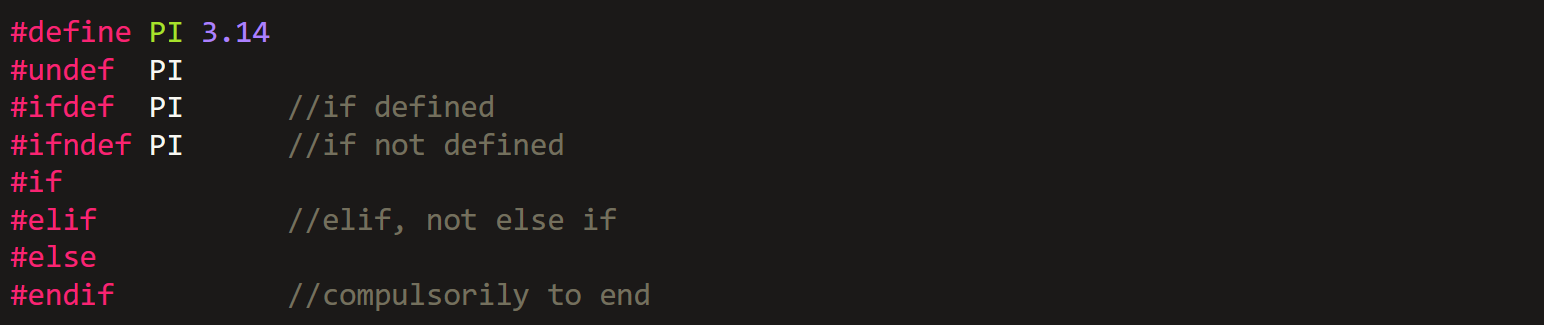


* The **statement to be checked** must be **immediately before** these control statements.
* **EDOM** is raised when function **can’t be performed** like **sqrt(-1.0)**.
* **ERANGE** is raised when function is **undefined** like **log(0.0)**.

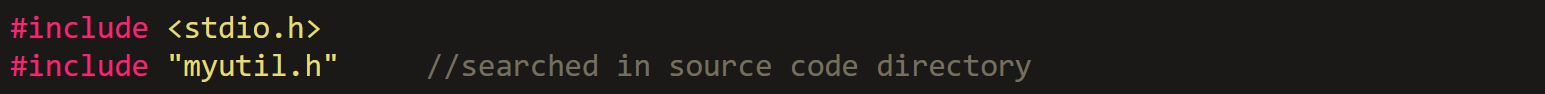


**Preprocessor Directives**

**Macro:** Replaceable piece of value.



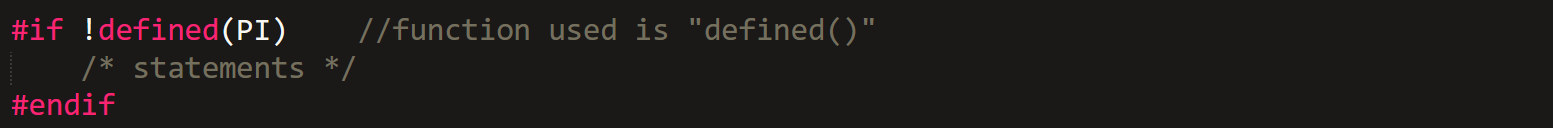
* The **data type** of variable defined via **#define** is decided by compiler.
* **#endif** is like break in switch-cases.





* Predefined directives (can directly write these in code as **string %s**):-
  + ***\_\_DATE\_\_*** current date in **mm dd yyy**
  + ***\_\_TIME\_\_*** current time in **hh:mm:ss**
  + ***\_\_FILE\_\_*** current filename as **string**
  + ***\_\_LINE\_\_*** current line number as **integer**

**Conditional Compilation Directives**



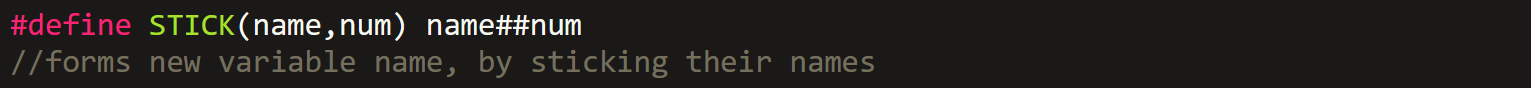
* We can even write statements like **int x = 5;** in macro conditional blocks!

**Preprocessor Operators**

* **Stringification/stringizing** operator converts parameter to string constant.



We just created a function that will **convert parameter to a string**. (TO\_STR name is not necessary).



* Even multiple variable names are **stickable**.

**Multithreading**

* 2 types of multi-tasking: **Process based** & **thread based**.
* **<pthread.h>** for **C/C++**
* **<thread>** for **C++** only.

Steps for creating a thread:-

* **Step 1:** Create a function of **void pointer** type:



* + There must be **atleast one *void\** type** **argument** in this function.
  + The **NULL** in **pthread\_exit()** is its exit status.
* **Step 2:** Create thread ID storage (in **main()** function):



**OR (for array of threads)**



* **Step 3:** Creating thread:



* **Step 4:** Add join function:



* + This function **waits** for thread to finish before main function can end.

Arguments of pthread\_create():-

* **&thread** – Address of the thread ID
* **NULL** – Attributes for thread
* **func** – Function we want to call
* **NULL** – The void pointer argument discussed earlier

Storing thread data in structures:-



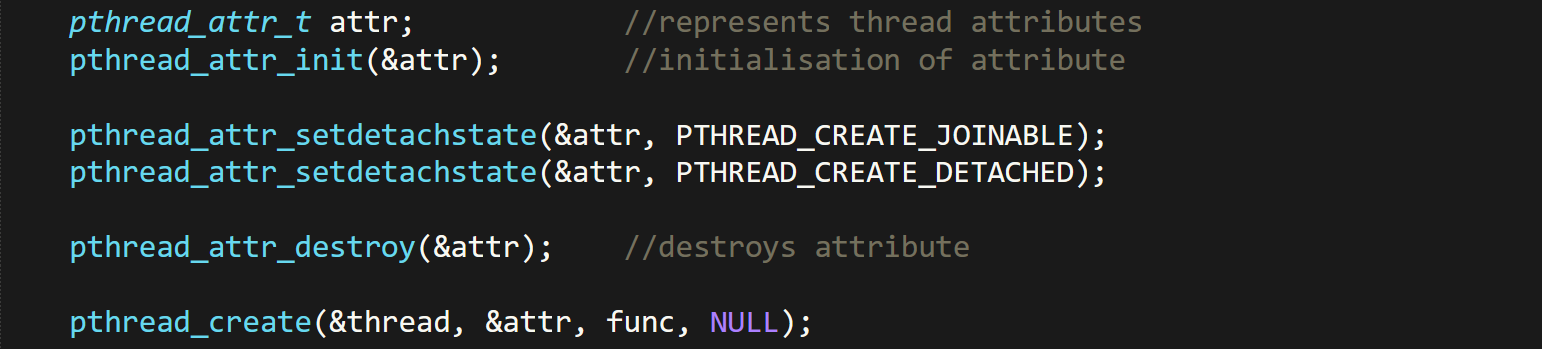
For delaying & sleep:-

* Use **<unistd.h>**.

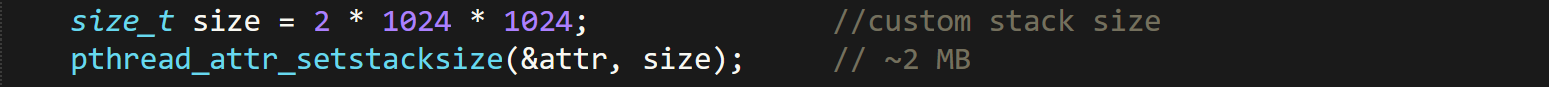


Setting attributes for threads:-

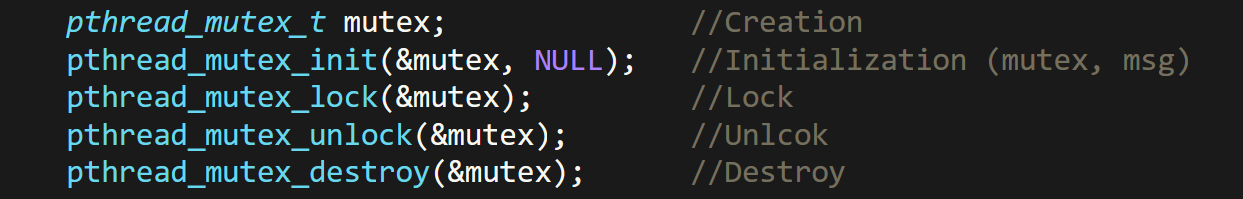
* **Threads can be made joinable or detachable:**



* **For allocating stack memory:**



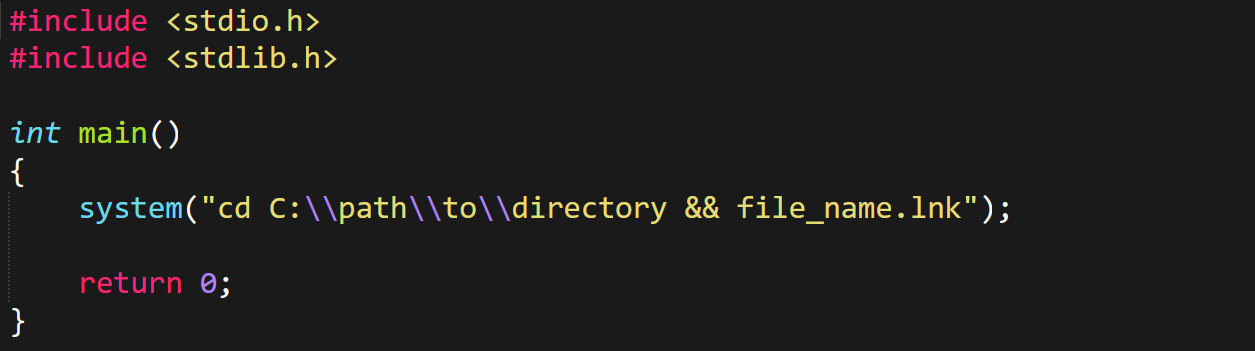
* **Mutex:**



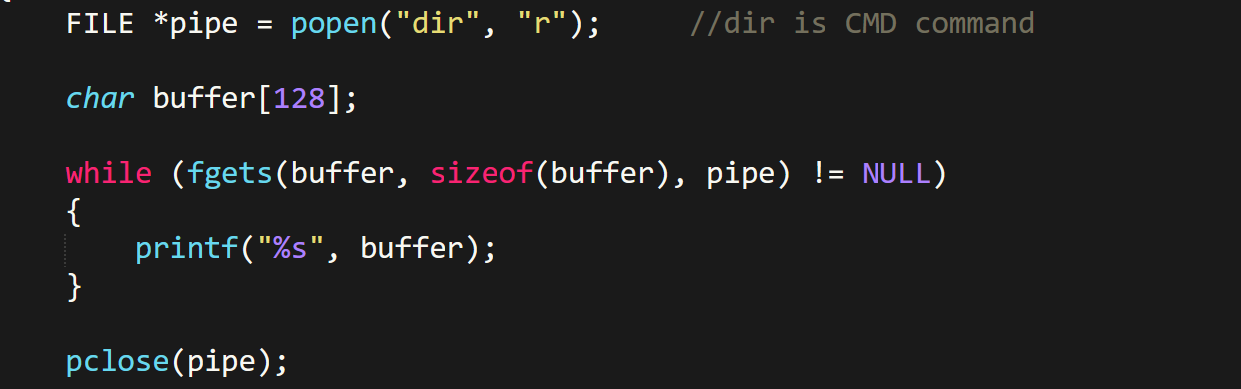
**Miscellaneous Library Functions**

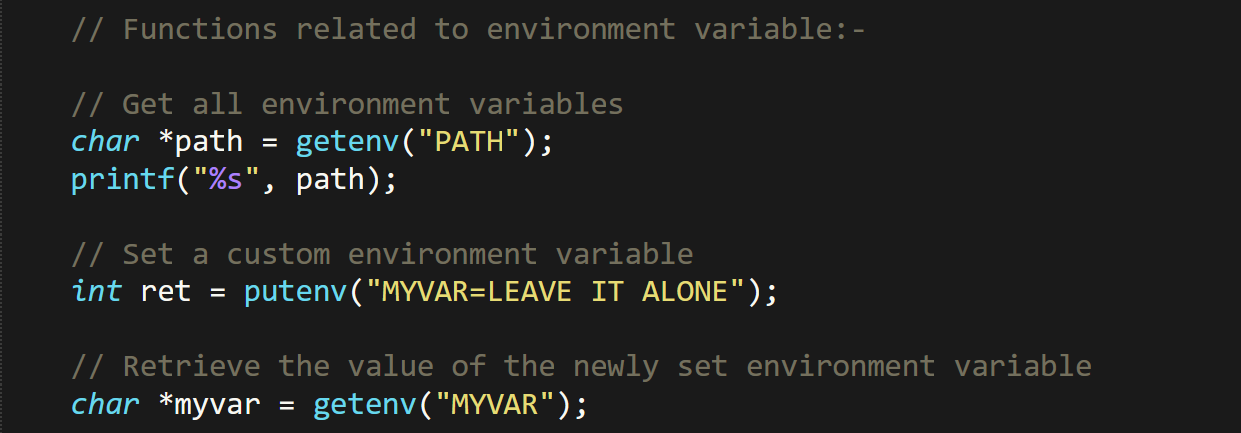
***<stdlib.h>***

* It directly writes command to **OS terminal**.



* Each time ***system()*** is used, they start from **default** environment.
* Notice that ***.lnk*** is used as an extension for the file to be opened.
* It reads the output of **CMD**.
* Uses simple **file management** and similar functions.





* **MYVAR** is an **in-built** mechanism, **not** an ordinary piece of string.
* Also, the **=** used in it.

