C cheatsheet

C quick reference cheat sheet that provides basic syntax and methods.

Getting Started

hello.c

```c
#include <stdio.h>

int main(void) {
  printf("Hello World!\n");

  return 0;
}
```

Compile `hello.c` file with `gcc`

```
$ gcc -Wall -g hello.c -o hello
```

Run the compiled binary `hello`

```
$ ./hello
```

Output => Hello World!

Variables

```c
int myNum = 15;

int myNum2; // do not assign, then assign
myNum2 = 15;

int myNum3 = 15; // myNum3 is 15
myNum3 = 10;     // myNum3 is now 10

float myFloat = 5.99; // floating point num
char myLetter = 'D';  // character

int x = 5;
int y = 6;
int sum = x + y; // add variables to sum

// declare multiple variables
int a = 5, b = 6, c = 50;
```

Constants

```c
const int minutesPerHour = 60;
const float PI = 3.14;
```

Best Practices

```c
const int BIRTHYEAR = 1980;
```

Comment

```c
// this is a comment
printf("Hello World!\n"); // Can comment an

/*Multi-line comment, print Hello World!
to the screen, it's awesome */
```

Print text

```c
printf("I am learning C.\n");
int testInteger = 5;
printf("Number = %d\n", testInteger);

float f = 5.99; // floating point number
printf("Value = %f\n", f);

short a = 0b1010110; // binary number
int b = 02713; // octal number
long c = 0X1DAB83; // hexadecimal number

// output in octal form
printf("a=%ho, b=%o, c=%lo\n", a, b, c);
// output => a=126, b=2713, c=7325603

// Output in decimal form
printf("a=%hd, b=%d, c=%ld\n", a, b, c);
// output => a=86, b=1483, c=1944451

// output in hexadecimal form (letter lower
printf("a=%hx, b=%x, c=%lx\n", a, b, c);
// output => a=56, b=5cb, c=1dab83

// Output in hexadecimal (capital letters)
printf("a=%hX, b=%X, c=%lX\n", a, b, c);
// output => a=56, b=5CB, c=1DAB83
```

Control the number of spaces

```c
int a1 = 20, a2 = 345, a3 = 700;
int b1 = 56720, b2 = 9999, b3 = 20098;
int c1 = 233, c2 = 205, c3 = 1;
int d1 = 34, d2 = 0, d3 = 23;

printf("%-9d %-9d %-9d\n", a1, a2, a3);
printf("%-9d %-9d %-9d\n", b1, b2, b3);
printf("%-9d %-9d %-9d\n", c1, c2, c3);
printf("%-9d %-9d %-9d\n", d1, d2, d3);
```

output result

```
20        345       700
56720     9999      20098
233       205       1
34        0         23
```

In `%-9d`, `d` means to output in `10` base, `9` means to occupy at least `9` characters width, and the width is not enough to fill with spaces, `-` means left alignment

Strings

```c
char greetings[] = "Hello World!";
printf("%s", greetings);
```

Access string

```c
char greetings[] = "Hello World!";
printf("%c", greetings[0]);
```

Modify string

```c
char greetings[] = "Hello World!";
greetings[0] = 'J';

printf("%s", greetings);
// prints "Jello World!"
```

Another way to create a string

```c
char greetings[] = {'H','e','l','l','\0'};

printf("%s", greetings);
// print "Hell!"
```

Creating String using character pointer (String Literals)

```c
char *greetings = "Hello";
printf("%s", greetings);
// print "Hello!"
```

NOTE: String literals might be stored in read-only section of memory. Modifying a string literal invokes undefined behavior. You can't modify it!

`C` **does not** have a String type, use `char` type and create an `array` of characters

Condition

```c
int time = 20;
if (time < 18) {
  printf("Goodbye!\n");
} else {
  printf("Good evening!\n");
}
// Output -> "Good evening!"
int time = 22;
if (time < 10) {
  printf("Good morning!\n");
} else if (time < 20) {
  printf("Goodbye!\n");
} else {
  printf("Good evening!\n");
}
// Output -> "Good evening!"
```

Ternary operator

```c
int age = 20;
(age > 19) ? printf("Adult\n") : printf("Teenager\n");
```

Switch

```c
int day = 4;

switch (day) {
  case 3: printf("Wednesday\n"); break;
  case 4: printf("Thursday\n"); break;
  default:
    printf("Weekend!\n");
}
// output -> "Thursday" (day 4)
```

While Loop

```c
int i = 0;

while (i < 5) {
  printf("%d\n", i);
  i++;
}
```

NOTE: Don't forget to increment the variable used in the condition, otherwise the loop will never end and become an "infinite loop"!

Do/While Loop

```c
int i = 0;

do {
  printf("%d\n", i);
  i++;
} while (i < 5);
```

For Loop

```c
for (int i = 0; i < 5; i++) {
  printf("%d\n", i);
}
```

Break out of the loop Break/Continue

```c
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

Break out of the loop when `i` is equal to 4

While Break Example

```c
int i = 0;

while (i < 10) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);

  i++;
}
```

While continue example

```c
int i = 0;

while (i < 10) {
  i++;

  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

```c
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

Example to skip the value of 4

Arrays

```c
int myNumbers[] = {25, 50, 75, 100};

printf("%d", myNumbers[0]);
// output 25
```

Change array elements

```c
int myNumbers[] = {25, 50, 75, 100};
myNumbers[0] = 33;

printf("%d", myNumbers[0]);
```

Loop through the array

```c
int myNumbers[] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

Set array size

```c
// Declare an array of four integers:
int myNumbers[4];

// add element
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;
```

Enumeration Enum

```c
enum week { Mon = 1, Tues, Wed, Thurs, Fri, Sat, Sun };
```

Define enum variable

```c
enum week a, b, c;
enum week { Mon = 1, Tues, Wed, Thurs, Fri, Sat, Sun } a, b, c;
```

With an enumeration variable, you can assign the value in the list to it

```c
enum week { Mon = 1, Tues, Wed, Thurs, Fri, Sat, Sun };
enum week a = Mon, b = Wed, c = Sat;
// or
enum week{ Mon = 1, Tues, Wed, Thurs, Fri, Sat, Sun } a = Mon, b = Wed, c = Sat;
```

Enumerate sample applications

```c
enum week {Mon = 1, Tues, Wed, Thurs} day;

scanf("%d", &day);

switch(day) {
  case Mon: puts("Monday"); break;
  case Tues: puts("Tuesday"); break;
  case Wed: puts("Wednesday"); break;
  case Thurs: puts("Thursday"); break;
  default: puts("Error!");
}
```

User input

```c
// Create an integer variable to store the
int myNum;

// Ask the user to enter a number
printf("Enter a number: ");

// Get and save the number entered by the u
scanf("%d", &myNum);

// Output the number entered by the user
printf("The number you entered: %d\n", myNu
```

User input string

```c
// create a string
char firstName[30];
// Ask the user to enter some text
printf("Enter your name: ");
// get and save the text
scanf("%s", &firstName);
// output text
printf("Hello %s.\n", firstName);
```

memory address

When a variable is created, it is assigned a memory address

```c
int myAge = 43;

printf("%p", &myAge);
// Output: 0x7ffe5367e044
```

To access it, use the reference operator (&)

create pointer

```c
int myAge = 43; // an int variable
printf("%d\n", myAge); // output the value

// Output the memory address of myAge (0x7f
printf("%p\n", &myAge);
```

pointer variable

```c
int myAge = 43; // an int variable
int*ptr = &myAge; // pointer variable named ptr, used to store the address of myAge

printf("%d\n", myAge); // print the value of myAge (43)

printf("%p\n", &myAge); // output the memory address of myAge (0x7ffe5367e044)
printf("%p\n", ptr); // use the pointer (0x7ffe5367e044) to output the memory address of myAg
```

Dereference

```c
int myAge = 43; // variable declaration
int*ptr = &myAge; // pointer declaration

// Reference: output myAge with a pointer
// memory address (0x7ffe5367e044)
printf("%p\n", ptr);
// dereference: output the value of myAge w
printf("%d\n", *ptr);
```

Operators

Arithmetic Operators

```
int myNum = 100 + 50;
int sum1 = 100 + 50; // 150 (100 + 50)
int sum2 = sum1 + 250; // 400 (150 + 250)
int sum3 = sum2 + sum2; // 800 (400 + 400)
```

+	Add	x + y
-	Subtract	x - y
*	Multiply	x * y
/	Divide	x / y
%	Modulo	x % y
++	Increment	++x
--	Decrement	--x

Assignment operator

x = 5	x = 5
x += 3	x = x + 3
x -= 3	x = x - 3
x *= 3	x = x * 3
x /= 3	x = x / 3
x %= 3	x = x % 3
x &= 3	x = x & 3
x |= 3	x = x | 3
x ^= 3	x = x ^ 3
x >>= 3	x = x >> 3
x <<= 3	x = x << 3

Comparison Operators

```
int x = 5;
int y = 3;

printf("%d", x > y);
// returns 1 (true) because 5 is greater th
```

==	equals	x == y
!=	not equal to	x != y
>	greater than	x > y
<	less than	x < y
>=	greater than or equal to	x >= y
<=	less than or equal to	x <= y

Comparison operators are used to compare two values

Logical Operators

Symbol	Name	Description	Example
&&	and logical	returns true if both statements are true	x < 5 && x < 10
||	or logical	returns true if one of the statements is true	x < 5 || x < 4
!	not logical	Invert result, return false if true	!(x < 5 && x < 10)

Operator Examples

```
unsigned int a = 60; /*60 = 0011 1100 */
unsigned int b = 13; /*13 = 0000 1101 */
int c = 0;

c = a & b; /*12 = 0000 1100 */
printf("Line 1 -the value of c is %d\n", c)

c = a | b; /*61 = 0011 1101 */
printf("Line 2 -the value of c is %d\n", c)
c = a ^ b; /*49 = 0011 0001 */
printf("Line 3 -the value of c is %d\n", c)
c = ~a; /*-61 = 1100 0011 */
printf("Line 4 -The value of c is %d\n", c)
c = a << 2; /*240 = 1111 0000 */
printf("Line 5 -the value of c is %d\n", c)
c = a >> 2; /*15 = 0000 1111 */
printf("Line 6 -The value of c is %d\n", c)
```

Bitwise operators

Operator	Description	Instance
&	Bitwise AND operation, "AND" operation by binary digits	(A & B) will get 12 which is 0000 1100
|	Bitwise OR operator, "or" operation by binary digit	(A | B) will get61 which is 0011 1101
^	XOR operator, perform "XOR" operation by binary digits	(A ^ B) will get 49 which is 0011 0001
~	Inversion operator, perform "inversion" operation by binary bit	(~A) will get -61 which is 1100 0011
<<	binary left shift operator	A << 2 will get 240 which is 1111 0000
>>	binary right shift operator	A >> 2 will get 15 which is 0000 1111

Data Types

Basic data types

Data Type	Size	Range	Description
`char`	1 byte	−128 ~ 127	single character/alphanumeric/ASCII
`signed char`	1 byte	−128 ~ 127	
`unsigned char`	1 byte	0 ~ 255	
`int`	2 to 4 bytes	−32,768 ~ 32,767	store integers
`signed int`	2 bytes	−32,768 ~ 32,767	
`unsigned int`	2 bytes	0 ~ 65,535	
`short int`	2 bytes	−32,768 ~ 32,767	
`signed short int`	2 bytes	−32,768 ~ 32,767	
`unsigned short int`	2 bytes	0 ~ 65,535	
`long int`	4 bytes	-2,147,483,648 ~ 2,147,483,647	
`signed long int`	4 bytes	-2,147,483,648 ~ 2,147,483,647	
`unsigned long int`	4 bytes	0 ~ 4,294,967,295	
`float`	4 bytes	3.4E-38 ~ 3.4E+38	
`double`	8 bytes	1.7E-308 ~ 1.7E+308	
`long double`	10 bytes	3.4E-4932 ~ 1.1E+4932	

Data types

```c
// create variables
int myNum = 5; // integer
float myFloatNum = 5.99; // floating point
char myLetter = 'D'; // string
// High precision floating point data or nu
double myDouble = 3.2325467;
// print output variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
printf("%lf\n", myDouble);
```

`char`	character type
`short`	short integer
`int`	integer type
`long`	long integer
`float`	single-precision floating-point type
`double`	double-precision floating-point type
`void`	no type

Basic format specifiers

Format Specifier	Data Type
`%d` or `%i`	`int` integer
`%f`	`float` single-precision decimal type
`%lf`	`double` high precision floating point data or number
`%c`	`char` character
`%s`	for `strings` strings

Separate base format specifiers

Format	Short	Int	Long
Octal	%ho	%o	%lo
Decimal	%hd	%d	%ld
Hexadecimal	%hx / %hX	%x / %X	%lx / %lX

Data format example

```
int myNum = 5;
float myFloatNum = 5.99; // floating point
char myLetter = 'D';     // string
// print output variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
```

C Preprocessor

Preprocessor Directives

Directive	Description
`#define`	define a macro
`#include`	include a source code file
`#undef`	undefined macro
`#ifdef`	Returns true if the macro is defined
`#ifndef`	Returns true if the macro is not defined
`#if`	Compile the following code if the given condition is true
`#else`	Alternative to `#if`
`#elif`	If the `#if` condition is false, the current condition is `true`
`#endif`	End a `#if...#else` conditional compilation block
`#error`	Print an error message when standard

Predefined macros

Macro	Description
`__DATE__`	The current date, a character constant in the format "MMM DD YYYY"
`__TIME__`	The current time, a character constant in the format "HH:MM:SS"
`__FILE__`	This will contain the current filename, a string constant
`__LINE__`	This will contain the current line number, a decimal constant
`__STDC__`	Defined as `1` when the compiler compiles against the `ANSI` standard

`ANSI C` defines a number of macros that you can use, but you cannot directly modify these predefined macros

Predefined macro example

```
#include <stdio.h>
```

Macro continuation operator (\)

A macro is usually written on a single line.

```
#define message_for(a, b) \
    printf(#a " and " #b ": We love you!\n"
```

If the macro is too long to fit on a single line, use the macro continuation operator `\`

	error is encountered
`#pragma`	Issue special commands to the compiler using the standardized method

```
// replace all MAX_ARRAY_LENGTH with 20
#define MAX_ARRAY_LENGTH 20
// Get stdio.h from the system library
#include <stdio.h>
// Get myheader.h in the local directory
#include "myheader.h"
#undef FILE_SIZE
#define FILE_SIZE 42 // undefine and define
```

```
int main(void) {
  printf("File: %s\n", __FILE__);
  printf("Date: %s\n", __DATE__);
  printf("Time: %s\n", __TIME__);
  printf("Line: %d\n", __LINE__);
  printf("ANSI: %d\n", __STDC__);
}
```

String Constantization Operator (#)

```
#include <stdio.h>

#define message_for(a, b) \
  printf(#a " and " #b ": We love you!\n")

int main(void) {
  message_for(Carole, Debra);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Carole and Debra: We love you!
```

When you need to convert a macro parameter to a string constant, use the string constant operator #

tag paste operator (##)

```c
#include <stdio.h>

#define tokenpaster(n) printf ("Token " #n

int main(void) {
  int token34 = 40;
  tokenpaster(34);

  return 0;
}
```

defined() operator

```c
#include <stdio.h>

#if !defined (MESSAGE)
   #define MESSAGE "You wish!"
#endif

int main(void) {
  printf("Here is the message: %s\n", MESSA

  return 0;
}
```

Parameterized macros

```c
int square(int x) {
  return x * x;
}
```

The macro rewrites the above code as follows:

```c
#define square(x) ( (x) * (x) )
```

No spaces are allowed between the macro name and the opening parenthesis

```c
#include <stdio.h>
#define MAX(x,y) ( (x) > (y) ? (x) : (y) )

int main(void) {
  printf("Max between 20 and 10 is %d\n", M

  return 0;
}
```

C Function

Function declaration and definition

```c
int main(void) {
  printf("Hello World!\n");

  return 0;
}
```

The function consists of two parts

```c
void myFunction() { // declaration declarat
  // function body (code to be executed) (d
}
```

- `Declaration` declares the function name, return type and parameters (if any)
- `Definition` function body (code to execute)

```c
// function declaration
void myFunction();
// main method
int main() {
  myFunction(); // --> call the function

  return 0;
}

void myFunction() {// Function definition
  printf("Good evening!\n");
}
```

Call function

```c
// create function
void myFunction() {
  printf("Good evening!\n");
}

int main() {
  myFunction(); // call the function
  myFunction(); // can be called multiple t

  return 0;
}
// Output -> "Good evening!"
// Output -> "Good evening!"
```

Function parameters

```c
void myFunction(char name[]) {
  printf("Hello %s\n", name);
}

int main() {
  myFunction("Liam");
  myFunction("Jenny");

  return 0;
}
// Hello Liam
// Hello Jenny
```

Multiple parameters

```c
void myFunction(char name[], int age) {
  printf("Hi %s, you are %d years old.\n",n
}
int main() {
  myFunction("Liam", 3);
  myFunction("Jenny", 14);

  return 0;
}
// Hi Liam you are 3 years old.
// Hi Jenny you are 14 years old.
```

Return value

```c
int myFunction(int x) {
  return 5 + x;
}

int main() {
  printf("Result: %d\n", myFunction(3));
  return 0;
}
// output 8 (5 + 3)
```

Two parameters

```c
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  printf("Result: %d\n", myFunction(5, 3));
  // store the result in a variable
  int result = myFunction(5, 3);
  printf("Result = %d\n", result);

  return 0;
```

Recursive example

```c
int sum(int k);

int main() {
  int result = sum(10);
  printf("%d\n", result);

  return 0;
}

int sum(int k) {
  if (k > 0) {
    return k + sum(k -1);
  } else {
    return 0;
  }
}
```

Mathematical functions

```c
#include <math.h>

void main(void) {
  printf("%f\n", sqrt(16)); // square root
  printf("%f\n", ceil(1.4)); // round up (r
  printf("%f\n", floor(1.4)); // round down
  printf("%f\n", pow(4, 3)); // x(4) to the
}
```

- `abs(x)` absolute value
- `acos(x)` arc cosine value
- `asin(x)` arc sine
- `atan(x)` arc tangent
- `cbrt(x)` cube root
- `cos(x)` cosine
- the value of `exp(x)` Ex
- `sin(x)` the sine of `x`
- tangent of `tan(x)` angle

```c
}
// result: 8 (5 + 3)
// result = 8 (5 + 3)
```

C Structures

Create structure

```c
struct MyStructure { // structure declarati
  int myNum; // member (int variable)
  char myLetter; // member (char variable)
}; // end the structure with a semicolon
```

Create a struct variable called `s1`

```c
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  struct myStructure s1;

  return 0;
}
```

Copy structure

```c
struct myStructure s1 = {
  13, 'B', "Some text"
};

struct myStructure s2;
s2 = s1;
```

In the example, the value of `s1` is copied to `s2`

Strings in the structure

```c
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30]; // String
};

int main() {
  struct myStructure s1;
  strcpy(s1. myString, "Some text");
  // print value
  printf("My string: %s\n", s1.myString);

  return 0;
}
```

Assigning values to strings using the `strcpy` function

Modify value

```c
// Create a struct variable and assign it a v
struct myStructure s1 = {
  13, 'B'
};
// modify the value
s1.myNum = 30;
s1.myLetter = 'C';
// print value
printf("%d %c",
    s1.myNum,
    s1.myLetter);
```

Accessing structure members

```c
// create a structure called myStructure
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  // Create a structure variable called myStr
  struct myStructure s1;
  // Assign values to the members of s1
  s1.myNum = 13;
  s1.myLetter = 'B';

  // Create a structure variable of myStructu
  // and assign it a value
  struct myStructure s2 = {13, 'B'};
  // print value
  printf("My number: %d\n", s1.myNum);
  printf("My letter: %c\n", s1.myLetter);
  return 0;
}
```

Create different structure variables

```c
struct myStructure s1;
struct myStructure s2;
// Assign values to different structure var
s1.myNum = 13;
s1.myLetter = 'B';

s2.myNum = 20;
s2.myLetter = 'C';
```

File Processing

File processing function

Function	Description
`fopen()`	open a new or existing file
`fprintf()`	write data to `file`
`fscanf()`	`read` data from a file
`fputc()`	write a character to `file`
`fgetc()`	`read` a character from a file
`fclose()`	`close` the file
`fseek()`	set the file pointer to `the given position`
`fputw()`	Write an integer `to` a file
`fgetw()`	`read` an integer from a file
`ftell()`	returns the current `position`
`rewind()`	set the file pointer to the beginning of the file

There are many functions in the C library to `open`/`read`/`write`/`search` and `close` files

Open mode parameter

Mode	Description
`r`	Open a text file in `read` mode, allowing the file to be read
`w`	Open a text file in `write` mode, allowing writing to the file
`a`	Open a text file in `append` mode If the file does not exist, a new one will be created
`r+`	Open a text file in `read-write` mode, allowing reading and writing of the file
`w+`	Open a text file in `read-write` mode, allowing reading and writing of the file
`a+`	Open a text file in `read-write` mode, allowing reading and writing of the file
`rb`	Open a binary file in `read` mode
`wb`	Open binary file in `write` mode
`ab`	Open a binary file in `append` mode
`rb+`	open binary file in `read-write` mode
`wb+`	Open binary file in `read-write` mode
`ab+`	open binary file in `read-write` mode

Open the file: fopen()

```c
#include <stdio.h>

void main() {
  FILE *fp;
  char ch;

  fp = fopen("file_handle.c", "r");

  while (1) {
    ch = fgetc(fp);
    if (ch == EOF)
      break;
    printf("%c", ch);
  }
  fclose(fp);
}
```

After performing all operations on the file, the file must be closed with `fclose()`

Write to file: fprintf()

```c
#include <stdio.h>

void main() {
  FILE *fp;
  fp = fopen("file.txt", "w"); // open the fi

  // write data to file
  fprintf(fp, "Hello file for fprintf..\n");
  fclose(fp); // close the file
}
```

Read the file: fscanf()

```c
#include <stdio.h>

void main() {
  FILE *fp;

  char buff[255]; // Create a char array to s
  fp = fopen("file.txt", "r");

  while(fscanf(fp, "%s", buff) != EOF) {
    printf("%s ", buff);
  }
  fclose(fp);
}
```

Write to file: fputc()

```c
#include <stdio.h>

void main() {
  FILE *fp;
  fp = fopen("file1.txt", "w"); // open the f
  fputc('a',fp); // write a single character
  fclose(fp); // close the file
}
```

Read the file: fgetc()

```c
#include <stdio.h>
#include <conio.h>

void main() {
  FILE *fp;
  char c;

  clrscr();

  fp = fopen("myfile.txt", "r");

  while( (c = fgetc(fp) ) != EOF) {
    printf("%c", c);
  }
  fclose(fp);

  getch();
}
```

Write to file: fputs()

```c
#include<stdio.h>
#include<conio.h>

void main() {
  FILE *fp;

  clrscr();

  fp = fopen("myfile2.txt","w");
  fputs("hello c programming",fp);
  fclose(fp);

  getch();
}
```

Read files: fgets()

```c
#include<stdio.h>
#include<conio.h>

void main() {
  FILE *fp;
  char text[300];

  clrscr();

  fp = fopen("myfile2.txt", "r");
  printf("%s", fgets(text, 200, fp));
  fclose(fp);

  getch();
}
```

fseek()

```c
#include <stdio.h>

void main(void) {
  FILE *fp;

  fp = fopen("myfile.txt","w+");
  fputs("This is Book", fp);

  // Set file pointer to the given position
  fseek(fp, 7, SEEK_SET);

  fputs("Kenny Wong", fp);
  fclose(fp);
}
```

Set the file pointer to the given position

rewind()

```c
#include <stdio.h>
#include <conio.h>

void main() {
  FILE *fp;
  char c;

  clrscr();

  fp = fopen("file.txt", "r");

  while( (c = fgetc(fp) ) != EOF) {
    printf("%c", c);
  }

  rewind(fp); // move the file pointer to the

  while( (c = fgetc(fp) ) != EOF) {
    printf("%c", c);
  }

  fclose(fp);

  getch();
}
// output
// Hello World! Hello World!
```

ftell()

```c
#include <stdio.h>
#include <conio.h>

void main () {
   FILE *fp;
   int length;

   clrscr();

   fp = fopen("file.txt", "r");

   fseek(fp, 0, SEEK_END);
   length = ftell(fp); // return current posi
   fclose(fp);

   printf("File size: %d bytes", length);

   getch();
}
// output
// file size: 18 bytes
```


Top Cheatsheet

Python Cheatsheet
Quick Reference

Vim Cheatsheet
Quick Reference

Recent Cheatsheet

Cheatsheet
Quick Reference

Unreal Engine Cheatsheet
Quick Reference

JavaScript Cheatsheet
Quick Reference

Bash Cheatsheet
Quick Reference

OCaml Cheatsheet
Quick Reference

Unity Shader Graph Cheats
Quick Reference