When accessing files through C, the first necessity is to have a way to access the files. For C File I/O you need to use a FILE pointer, which will let the program keep track of the file being accessed. For Example:

|  |
| --- |
| FILE \*fp; |

To open a file you need to use the fopen function, which returns a FILE pointer. Once you've opened a file, you can use the FILE pointer to let the compiler perform input and output functions on the file.

|  |
| --- |
| FILE \*fopen(const char \*filename, const char \*mode); |

Here filename is string literal which you will use to name your file and mode can have one of the following values

|  |
| --- |
| w - open for writing (file need not exist)  a - open for appending (file need not exist)  r+ - open for reading and writing, start at beginning  w+ - open for reading and writing (overwrite file)  a+ - open for reading and writing (append if file exists) |

Note that it's possible for fopen to fail even if your program is perfectly correct: you might try to open a file specified by the user, and that file might not exist (or it might be write-protected). In those cases, fopen will return 0, the NULL pointer.

Here's a simple example of using fopen:

|  |
| --- |
| FILE \*fp;  fp=fopen("/home/tutorialspoint/test.txt", "r"); |

This code will open test.txt for reading in text mode. To open a file in a binary mode you must add a b to the end of the mode string; for example, "rb" (for the reading and writing modes, you can add the b either after the plus sign - "r+b" - or before - "rb+")

To close a function you can use the function:

|  |
| --- |
| int fclose(FILE \*a\_file); |

fclose returns zero if the file is closed successfully.

An example of fclose is:

|  |
| --- |
| fclose(fp); |

To work with text input and output, you use fprintf and fscanf, both of which are similar to their friends printf and scanf except that you must pass the FILE pointer as first argument.

Try out following example:

|  |
| --- |
| #include <stdio.h>  main()  {  FILE \*fp;  fp = fopen("/tmp/test.txt", "w");  fprintf(fp, "This is testing...\n");  fclose(fp;);  } |

Thsi will create a file test.txt in /tmp directory and will write This is testing in that file.

Here is an example which will be used to read lines from a file:

|  |
| --- |
| #include <stdio.h>  main()  {  FILE \*fp;  char buffer[20];  fp = fopen("/tmp/test.txt", "r");  fscanf(fp, "%s", buffer);  printf("Read Buffer: %s\n", %buffer );  fclose(fp;);  } |

It is also possible to read (or write) a single character at a time--this can be useful if you wish to perform character-by-character input. The fgetc function, which takes a file pointer, and returns an int, will let you read a single character from a file:

|  |
| --- |
| int fgetc (FILE \*fp); |

The fgetc returns an int. What this actually means is that when it reads a normal character in the file, it will return a value suitable for storing in an unsigned char (basically, a number in the range 0 to 255). On the other hand, when you're at the very end of the file, you can't get a character value--in this case, fgetc will return "EOF", which is a constnat that indicates that you've reached the end of the file.

The fputc function allows you to write a character at a time--you might find this useful if you wanted to copy a file character by character. It looks like this:

|  |
| --- |
| int fputc( int c, FILE \*fp ); |

Note that the first argument should be in the range of an unsigned char so that it is a valid character. The second argument is the file to write to. On success, fputc will return the value c, and on failure, it will return EOF.

Binary I/O

There are following two functions which will be used for binary input and output:

|  |
| --- |
| size\_t fread(void \*ptr, size\_t size\_of\_elements,  size\_t number\_of\_elements, FILE \*a\_file);    size\_t fwrite(const void \*ptr, size\_t size\_of\_elements,  size\_t number\_of\_elements, FILE \*a\_file); |

Both of these functions deal with blocks of memories - usually arrays. Because they accept pointers, you can also use these functions with other data structures; you can even write structs to a file or a read struct into memory.

C provides two sytles of flow control:

Branching

Looping

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

Branching:

Branching is so called because the program chooses to follow one branch or another.

if statement

This is the most simple form of the branching statements.

It takes an expression in parenthesis and an statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

NOTE: Expression will be assumed to be true if its evaulated values is non-zero.

if statements take the following form:

[Show Example](http://www.tutorialspoint.com/ansi_c/if_statement_examples.htm)

|  |
| --- |
| if (expression)  statement;  or  if (expression)  {  Block of statements;  }  or  if (expression)  {  Block of statements;  }  else  {  Block of statements;  }  or  if (expression)  {  Block of statements;  }  else if(expression)  {  Block of statements;  }  else  {  Block of statements;  } |

? : Operator

The ? : operator is just like an if ... else statement except that because it is an operator you can use it within expressions.

? : is a ternary operator in that it takes three values, this is the only ternary operator C has.

? : takes the following form:

[Show Example](http://www.tutorialspoint.com/ansi_c/ternary_operator_examples.htm)

|  |
| --- |
| if condition is true ? then X return value : otherwise Y value; |

switch statement:

The switch statement is much like a nested if .. else statement. Its mostly a matter of preference which you use, switch statement can be slightly more efficient and easier to read.

[Show Example](http://www.tutorialspoint.com/ansi_c/switch_statement_examples.htm)

|  |
| --- |
| switch( expression )  {  case constant-expression1: statements1;  [case constant-expression2: statements2;]  [case constant-expression3: statements3;]  [default : statements4;]  } |

Using break keyword:

If a condition is met in switch case then execution continues on into the next case clause also if it is not explicitly specified that the execution should exit the switch statement. This is achieved by using break keyword.

Try out given example [Show Example](http://www.tutorialspoint.com/ansi_c/switch_statement_examples.htm)

What is default condition:

If none of the listed conditions is met then default condition executed.

Try out given example [Show Example](http://www.tutorialspoint.com/ansi_c/switch_statement_examples.htm)

Looping

Loops provide a way to repeat commands and control how many times they are repeated. C provides a number of looping way.

while loop

The most basic loop in C is the while loop.A while statement is like a repeating if statement. Like an If statement, if the test condition is true: the statments get executed. The difference is that after the statements have been executed, the test condition is checked again. If it is still true the statements get executed again.This cycle repeats until the test condition evaluates to false.

Basic syntax of while loop is as follows:

[Show Example](http://www.tutorialspoint.com/ansi_c/while_loop_example.htm)

|  |
| --- |
| while ( expression )  {  Single statement  or  Block of statements;  } |

for loop

for loop is similar to while, it's just written differently. for statements are often used to proccess lists such a range of numbers:

Basic syntax of for loop is as follows:

[Show Example](http://www.tutorialspoint.com/ansi_c/for_loop_example.htm)

|  |
| --- |
| for( expression1; expression2; expression3)  {  Single statement  or  Block of statements;  } |

In the above syntax:

expression1 - Initialisese variables.

expression2 - Condtional expression, as long as this condition is true, loop will keep executing.

expression3 - expression3 is the modifier which may be simple increment of a variable.

do...while loop

do ... while is just like a while loop except that the test condition is checked at the end of the loop rather than the start. This has the effect that the content of the loop are always executed at least once.

Basic syntax of do...while loop is as follows:

[Show Example](http://www.tutorialspoint.com/ansi_c/do_loop_example.htm)

|  |
| --- |
| do  {  Single statement  or  Block of statements;  }while(expression); |

break and continue statements

C provides two commands to control how we loop:

break -- exit form loop or switch.

continue -- skip 1 iteration of loop.

You already have seen example of using break statement. Here is an example showing usage ofcontinue statement.

|  |
| --- |
| #include  main()  {  int i;  int j = 10;  for( i = 0; i <= j; i ++ )  {  if( i == 5 )  {  continue;  }  printf("Hello %d\n", i );  }  } |

This will produce following output:

C Programming Files I/O

In C programming, file is a place on disk where a group of related data is stored.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands.

There are large numbers of functions to handle file I/O in C language. In this tutorial, you will learn to handle standard I/O(High level file I/O functions) in C.

High level file I/O functions can be categorized as:

Text file

Binary file

File Operations

Creating a new file

Opening an existing file

Reading from and writing information to a file

Closing a file

Working with file

While working with file, you need to declare a pointer of type file. This declaration is needed for communication between file and program.

FILE \*ptr;

Opening a file

Opening a file is performed using library function fopen(). The syntax for opening a file in standard I/O is:

ptr=fopen("fileopen","mode")

For Example:

fopen("E:\\cprogram\program.txt","w");

/\* --------------------------------------------------------- \*/

E:\\cprogram\program.txt is the location to create file.

"w" represents the mode for writing.

/\* --------------------------------------------------------- \*/

Here, the program.txt file is opened for writing mode.

| Opening Modes in Standard I/O | | |
| --- | --- | --- |
| File Mode | Meaning of Mode | During Inexistence of file |
| r | Open for reading. | If the file does not exist, fopen() returns NULL. |
| w | Open for writing. | If  the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a | Open for append. i.e, Data is added to end of file. | If the file does not exists, it will be created. |
| r+ | Open for both reading and writing. | If the file does not exist, fopen() returns NULL. |
| w+ | Open for both reading and writing. | If  the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a+ | Open for both reading and appending. | If the file does not exists, it will be created. |

Closing a File

The file should be closed after reading/writing of a file. Closing a file is performed using library function fclose().

fclose(ptr); //ptr is the file pointer associated with file to be closed.

The Functions fprintf() and fscanf() functions.

The functions fprintf() and fscanf() are the file version of printf() and fscanf(). The only difference while using fprintf() and fscanf() is that, the first argument is a pointer to the structure FILE

Writing to a file

#include <stdio.h>

int main()

{

int n;

FILE \*fptr;

fptr=fopen("C:\\program.txt","w");

if(fptr==NULL){

printf("Error!");

exit(1);

}

printf("Enter n: ");

scanf("%d",&n);

fprintf(fptr,"%d",n);

fclose(fptr);

return 0;

}

This program takes the number from user and stores in file. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open that file, you can see the integer you entered.

Similarly, fscanf() can be used to read data from file.

Reading from file

#include <stdio.h>

int main()

{

int n;

FILE \*fptr;

if ((fptr=fopen("C:\\program.txt","r"))==NULL){

printf("Error! opening file");

exit(1); /\* Program exits if file pointer returns NULL. \*/

}

fscanf(fptr,"%d",&n);

printf("Value of n=%d",n);

fclose(fptr);

return 0;

}

If you have run program above to write in file successfully, you can get the integer back entered in that program using this program.

Other functions like fgetchar(), fputc() etc. can be used in similar way.

Binary Files

Depending upon the way file is opened for processing, a file is classified into text file and binary file.

If a large amount of numerical data it to be stored, text mode will be insufficient. In such case binary file is used.

Working of binary files is similar to text files with few differences in opening modes, reading from file and writing to file.

Opening modes of binary files

Opening modes of binary files are rb, rb+, wb, wb+,ab and ab+. The only difference between opening modes of text and binary files is that, b is appended to indicate that, it is binary file.

Reading and writing of a binary file.

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

Function fwrite() takes four arguments, address of data to be written in disk, size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

fwrite(address\_data,size\_data,numbers\_data,pointer\_to\_file);

Function fread() also take 4 arguments similar to fwrite() function as above.