**C PROGRAMMING**

**OUTPUT FUNCTION OTHER THAN printf()**

**INTRODUCTION**

Output functions are crucial components in computer programming, enabling developers to display information and results to users or save data to various output sources. While the printf function is a widely recognized method for output in many programming languages, there exist a multitude of other output functions with their unique features and use cases. These alternative output functions offer flexibility, customization, and compatibility with different output destinations, making them invaluable tools for programmers seeking to create diverse and interactive applications. In this discussion, we will explore various output functions beyond printf, highlighting their advantages and showcasing their utility in different programming scenarios.

**PUTS()**

puts() is a function in the C programming language that is used to print a string to the console. It prints the passed string with a newline character at the end and returns a non-negative value on success. The function is defined in the header file stdio.h.The puts() function returns an non-negative integer number for successful execution. Otherwise returns EOF for any error.

* **The puts() function appends a newline character at the end while writing a string/line.**
* **The puts() function in C is used to write a line or string to the output(stdout) stream. It prints the passed string with a newline and returns an integer value.**

SYNTAX:

int puts(const char \*str);

## **EXAMPLE 1 PROGRAM**

#include <stdio.h>

int main()

{

char\* str1 = "hello"; OUTPUT

puts(str1);

puts("welcome"); hello

welcome

return 0;

}

**EXAMPLE 2 PROGRAM**

#include <stdio.h>

int main() {

char input[100];

printf("Enter a message: "); **OUTPUT**

fgets(input, sizeof(input), stdin); Enter a message: hello

printf("You entered: "); You entered:hello

puts(input);

return 0;

}

**FPRINTF()**

F- formate

In C, the fprintf function is used for formatted output to a file, whereas printf is used for formatted output to the standard output (typically the console). fprintf allows you to write formatted data to a specified file stream, such as a text file, rather than displaying it on the screen.

**SYNTAX**

int fprintf(FILE \*stream, const char \*format, ...);

* stream: A pointer to the file stream where you want to write the formatted output.
* format: A format string that specifies how the data should be formatted.
* ...: Additional arguments that match the format specifiers in the format string.

**EXAMPLE 1 PROGRAM**

#include<stdio.h>

int main()

{

int i, n=2;

char str[50];

FILE \*fptr = fopen("sample.txt", "w"); //open file sample.txt in write mode

if (fptr == NULL)

{

printf("Could not open file");

return 0;

}

for (i = 0; i < n; i++)

{

puts("Enter a name");

scanf("%[^\n]%\*c", str);

fprintf(fptr,"%d.%s\n", i, str);

}

fclose(fptr);

return 0;

}

**PUTC()**

putc() is a C library function used to write a character to a specified file stream. It is similar to the fputc() function and can be used to write characters to a file or to the standard output.

SYNTAX:

int putc(int character, FILE \*stream);

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

int main() {

FILE \*file = fopen("output.txt", "w"); // Open a file for writing

if (file == NULL) {

printf("Failed to open the file for writing.\n");

return 1;

}

char message[] = "Hello, putc!";

// Write each character of the message to the file using putc

for (int i = 0; message[i] != '\0'; i++) {

int result = putc(message[i], file);

if (result == EOF) {

printf("Error writing to the file.\n");

break;

}

}

// Close the file when you're done

fclose(file);

return 0;

}

In this example, we open a file named "output.txt" for writing and use a for loop to iterate through the characters in the message array, writing each character to the file using putc. We check the result of putc to handle any potential errors during writing. Finally, we close the file when done.

**VPRINTF()**

In C, the vprintf function is used to print formatted output to the standard output (usually the console) using a variable argument list. It is part of the <stdio.h> library and is often used when you want to print formatted output but don't know the number of arguments in advance.

Here is the syntax of the vprintf function:

int vprintf(const char \*format, va\_list arg);

* format: A format string specifying how to format the output.
* arg: A va\_list object that contains the variable arguments to be printed.

The vprintf function is similar to printf, but instead of specifying individual arguments, it accepts a va\_list object containing the variable arguments. This is useful when you want to create functions that take a variable number of arguments and pass them to vprintf for formatted output.

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

#include <stdarg.h>

void print\_numbers(const char \*format, ...) {

va\_list args;

va\_start(args, format);

vprintf(format, args);

va\_end(args);

}

int main() {

print\_numbers("Numbers: %d, %d, %d, %f, %s\n", 42, 123, 987, 3.14, "Hello");

return 0;

}

**EXAMPLE 2 PROGRAM**

#include <stdio.h>

#include <stdarg.h>

void printIntegers(const char \*format, ...) {

va\_list args;

va\_start(args, format);

while (\*format) {

if (\*format == 'd') {

int num = va\_arg(args, int);

printf("%d ", num);

}

format++;

}

va\_end(args); **OUTPUT**

printf("\n");

} 10 20 30

int main() {

int a = 10;

int b = 20;

int c = 30;

printIntegers("The numbers are: %d %d %d", a, b, c);

return 0;

}

**FWRITE()**

The fwrite function in C is used to write binary data to a file. It's commonly used to save data structures, such as arrays or structs, directly to a file.

SYNTAX:

size\_t fwrite(const void \*ptr, size\_t size, size\_t count, FILE \*stream);

* ptr: A pointer to the data that you want to write to the file.
* size: The size in bytes of each element that you want to write.
* count: The number of elements you want to write.
* stream: A pointer to a FILE object that represents the file you want to write to.

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

int main() {

int numbers[] = {1, 2, 3, 4, 5};

FILE \*file = fopen("numbers.bin", "wb"); // Open the file in binary write mode

if (file == NULL) {

perror("Error opening the file");

return 1;

}

size\_t num\_elements = sizeof(numbers) / sizeof(numbers[0]);

size\_t element\_size = sizeof(numbers[0]);

size\_t elements\_written = fwrite(numbers, element\_size, num\_elements, file);

if (elements\_written != num\_elements) {

perror("Error writing to the file");

return 1;

}

fclose(file); // Close the file

return 0;

}

**EXAMPLE 2 PROGRAM**

#include <stdio.h>

#include <string.h>

// Define a structure to represent a student record

struct Student {

char name[50];

int roll\_number;

float marks;

};

int main() {

// Create an array of student records

struct Student students[] = {

{"Alice", 101, 85.5},

{"Bob", 102, 78.0},

{"Charlie", 103, 92.5},

};

FILE \*file = fopen("student\_data.bin", "wb"); // Open the file in binary write mode

if (file == NULL) {

perror("Error opening the file");

return 1;

}

size\_t num\_students = sizeof(students) / sizeof(students[0]);

// Write the student records to the binary file

size\_t records\_written = fwrite(students, sizeof(struct Student), num\_students, file);

if (records\_written != num\_students) {

perror("Error writing to the file");

return 1;

}

fclose(file); // Close the file

printf("Student records have been successfully written to the file.\n");

return 0;

}

**PUTWC()**

putwc() is a function in C that is used to write a wide character to a file. It is part of the wide character (wchar\_t) I/O functions and is typically used for working with multibyte character encodings, such as UTF-16 or UTF-32.

SYNTAX:

wint\_t putwc(wchar\_t widechar, FILE \*stream);

* widechar: The wide character to be written to the file.
* stream: A pointer to the FILE structure that represents the file where you want to write the wide character

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

#include <wchar.h>

int main() {

FILE \*file = fopen("widechar.txt", "w"); // Open a text file for writing

if (file == NULL) {

perror("Error opening the file");

return 1;

}

wchar\_t widechar = L'€'; // A Euro symbol as a wide character

// Write the wide character to the file using putwc

if (putwc(widechar, file) == WEOF) {

perror("Error writing the wide character to the file");

fclose(file);

return 1;

}

fclose(file); // Close the file

printf("Wide character has been successfully written to the file.\n");

return 0;

}

**PUTCHAR()**

The putchar() function in C is used to write a single character to the standard output, typically the console or terminal. It is part of the standard I/O library in C.

SYNTAX:

int putchar(int character);

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

int main() {

int ch = 'A'; // ASCII code for 'A'

putchar(ch);

return 0;

}

OUTPUT

A

**EXAMPLE 2 PROGRAM**

#include <stdio.h>

int main() {

printf("ASCII Table:\n");

for (int i = 32; i < 127; i++) {

putchar(i);

if ((i - 31) % 16 == 0) {

putchar('\n');

} else {

putchar('\t');

}

}

return 0;

}

OUTPUT

ASCII Table:

! " # $ % & ' ( ) \* + , - . /

0 1 2 3 4 5 6 7 8 9 : ; < = > ?

@ A B C D E F G H I J K L M N O

P Q R S T U V W X Y Z [ \ ]^ \_

` a b c d e f g h i j k l m n o

p q r s t u v w x y z { | } ~

**PRINTF()**

printf() is a commonly used function in C programming for formatted output. It allows you to display text and data on the console or write to a file in a specified format. The name "printf" stands for "print formatted."

SYNTAX

int printf(const char \*format, ...);

**EXAMPLE 1 PROGRAM**

#include <stdio.h>

// Function to calculate the nth Fibonacci number using recursion

int fibonacci(int n) {

if (n <= 1)

return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

int n, i;

printf("Enter the number of terms in the Fibonacci sequence: ");

scanf("%d", &n);

printf("\nFibonacci Sequence:\n");

for (i = 0; i < n; i++) {

int fib = fibonacci(i);

// Demonstrating various formatting options with printf

printf("Term %2d: %6d\n", i + 1, fib); // Right-aligned, width 6

}

return 0;

}