

# File Operation and Stings

Course: Introduction to Programming and Data Structures

**Laltu Sardar**

Institute for Advancing Intelligence (IAI),  
TCG Centres for Research and Education in Science and Technology (TCG Crest)

**tcg crest**

Inventing Harmonious Future

August 22, 2024

# Basics of File Handling in C

## fscanf and fprintf

- **fscanf** and **fprintf** works almost same as **scanf** and **printf**

```

1 // Program to learn basic file operation
2 #include <stdio.h>
3
4 float average(float a, float b){
5     return ((a+b)/2.0);
6 }
7
8 int main(){
9     float a, b, avg;
10
11     FILE * inp_file_ptr, * out_file_ptr; //File type pointer must be declared
12
13     inp_file_ptr = fopen("input_file.txt","r"); // Opening input file for
        reading
14     fscanf(inp_file_ptr, "%f %f", &a, &b); // taking input from file
15     fclose(inp_file_ptr); // closing the input file
16
17     avg = average(a, b); //Computing average
18
19     out_file_ptr = fopen("output_file.txt","w");
20     fprintf(out_file_ptr, "%f", avg); //writing on output file
21     fclose(out_file_ptr); //closing the output file
22
23     return 0;
24 }

```

# File opening modes

- When you open a file, you need to specify the mode in which you want to open it. The following are the different file modes:

Mode	Meaning of Mode	During Inexistence of File
r	Reading.	If the file does not exist, <code>fopen()</code> returns NULL.
w	Writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Append.	Data is added to the end of the file. If the file does not exist, it will be created.
r+	Reading and Writing.	If the file does not exist, <code>fopen()</code> returns NULL.
w+	Reading and Writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Reading and Appending.	If the file does not exist, it will be created.

**Table:** File opening modes in C

# Reading from a file

Function	Description
<code>fscanf()</code>	Use formatted string and variable arguments list to take input from a file. <code>int fscanf(FILE *ptr, const char *format, ...)</code>
<code>fgets()</code>	Input the whole line from the file. <code>char *fgets(char *str, int n, FILE *stream)</code>
<code>fgetc()</code>	Reads a single character from the file. <code>int fgetc(FILE *pointer)</code>
<code>fread()</code>	Reads the specified bytes of data from a binary file. <code>size_t fread(void *ptr, size_t size, size_t nmem, FILE *stream)</code>

Table: Some functions to Read from a file

# Writing to a file

Function	Description
<code>fprintf()</code>	Similar to <code>printf()</code> , this function print output to the file. <code>int fprintf(FILE *fptr, const char *str, ...);</code>
<code>fputs()</code>	Prints the whole line in the file and a newline at the end. <code>int fputs(const char *str, FILE *stream)</code>
<code>fputc()</code>	Prints a single character into the file. <code>int fputc(int char, FILE *pointer)</code>
<code>fwrite()</code>	This function writes the specified amount of bytes to the binary file. <code>size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)</code>

Table: Some functions to Write from a file

# Closing a file

- 1 The `fclose()` function is used to close the file
- 2 After successful file operations, you must always close a file **to remove it from the memory**.
- 3 Syntax of `fclose()`  
`fclose(file_pointer);`

# Dynamic Memory Allocation



# Dynamic Memory Allocation

- We were defining array as  
`int a[N]`
- Problem: what if failed?
- What if more memory required?
- Available Function `malloc`
- Library required `stdlib.h`

# Dynamic Memory Allocation

- We were defining array as  
`int a[N]`
- Problem: what if failed?
- What if more memory required?
- Available Function `malloc`
- Library required `stdlib.h`

```
1  int *A ;  
2  scanf("%d", &N);  
3  
4  A = (int *) malloc(N);
```

# Memory Allocation: `malloc`

- `malloc` allocates memory in bytes.
- Input: a positive number  $N$
- Output: A contiguous memory of size  $N$ -bytes from RAM.
- Is Typecast required?

# Memory Allocation: malloc

- malloc allocates memory in bytes.
- Input: a positive number  $N$
- Output: A contiguous memory of size  $N$ -bytes from RAM.
- Is Typecast required?

Try your own

```
A = (int *) malloc(5);
```

# Contiguous Allocation: calloc

```
A = (int *) calloc(N, sizeof(int));
```

- malloc just allocates memory
- calloc allocates memory and initialized with 0
- malloc is faster.

# Re-allocation: realloc

```
new_ptr = (int *)realloc(old_ptr, new_size);
```

- realloc just re-allocates memory
- In general when we need to increase memory? (check what will happen if decreased)

## Freeing the allocated memory

- Why? it does not automatically makes them free
- syntax:  
`free(ptr);`

# Swapping values of two variables

Write a function that swaps value of two integer variables.

- Take input from command line two integers a and b as  
`scanf("%d %d",&a,&b);`
- output the values after swapping as  
`printf("%d %d",a,b);`
- name the function as `swap_int()`

# Strings



# Introduction

- Strings are a fundamental concept in C programming.
- In C, strings are represented as arrays of characters.
- Strings can be accessed using pointers. A pointer to a string is a variable that stores the address of the first character in the string.
- C-style strings are null-terminated, meaning they are terminated by a null character (`\0`).

# String Declaration and Initialization

- Strings can be declared and initialized in various ways:
  - `char str[] = "Hello";`
  - `char str[10] = "Hello";`
  - `char *str = "Hello";`
- The size of the array should accommodate the string length plus one for the null character.

# Some common Operations on Strings

There are many operations that can be performed on strings in C. Some of the most common operations include:

- Concatenating two strings: This operation combines two strings into a single string.
- Determining the length of a string: This operation returns the number of characters in a string.
- Searching for a substring in a string: This operation returns the index of the first occurrence of a substring in a string.
- Replacing a substring in a string: This operation replaces all occurrences of a substring in a string with another substring.
- Sorting the characters in a string: This operation sorts the characters in a string in alphabetical order.
- Copying: Copying one string to another.

# String Functions

- C provides a set of functions in the `<string.h>` library for string manipulation:
  - `strlen()`
  - `strcpy()` and `strncpy()`
  - `strcat()` and `strncat()`
  - `strcmp()` and `strncmp()`
  - `strstr()` and `strchr()`
  - `sprintf()` and `sscanf()`

# Array of Strings

# Declaration and Initialization

- Declaring an array of strings:
  - `char names[5][20];`
  - `char cities[3][15];`
- Initializing the array of strings:
  - `char fruits[][10] = {"apple", "banana", "cherry"};`

# Accessing and Modifying Elements

- Accessing individual strings: `names[2]`
- Modifying strings: `strcpy(names[1], "John");`
- Using loops for batch operations:
  - `for (int i = 0; i < 3; i++) { strcpy(cities[i], "Unknown"); }`

# Multidimensional Arrays vs. Array of Strings

- Multidimensional arrays: Elements are of the same data type (e.g., `int`).
- Array of strings: Elements are arrays themselves (`char` arrays).
- Array of strings allows flexibility in handling variable-length text.