# File Operation, Dynamic Memory Allocation and Structures in C

## 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

```c
// Program to learn basic file operation
#include<stdio.h>

float average(float a, float b){
    return ((a+b)/2.0);
}

int main(){
    float a, b, avg;

    FILE * inp_file_ptr, * out_file_ptr; //File type pointer must be declared

    inp_file_ptr = fopen("input_file.txt","r"); // Opening input file for
        reading
    fscanf(inp_file_ptr, "%f %f", &a, &b);  // taking input from file
    fclose(inp_file_ptr);   // closing the input file

    avg = average(a, b);    //Compauting avarage

    out_file_ptr = fopen("output_file.txt","w");
    fprintf(out_file_ptr, "%f",avg); //writing on output file
    fclose(out_file_ptr); //closing the output file

    return 0;
}
```

# 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, `fopen()` returns `NULL`. |
| `w` | Writing. | If the file exists, its contents are overwritten.<br>If the file does not exist, it will be created. |
| `a` | Append. | Data is added to the end of the file.<br>If the file does not exist, it will be created. |
| `r+` | Reading and Writing. | If the file does not exist, `fopen()` returns `NULL`. |
| `w+` | Reading and Writing. | If the file exists, its contents are overwritten.<br>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

tcg crest
Inventing Harmonious Future

# Reading from a file

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

Table: Some functions to Read from a file

tcg crest
Inventing Harmonious Future

# Writing to a file

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

Table: Some functions to Write from a file

tcg crest
Inventing Harmonious Future

# 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);
```

tcg crest
Inventing Harmonious Future

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

tcg crest
Inventing Harmonious Future

# 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);
```

tcg crest
Inventing Harmonious Future

# Contiguous Allocation: `calloc`

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

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

tcg crest
Inventing Harmonious Future

# 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);
  ```

tcg crest
Inventing Harmonious Future

# 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()`

# Structures

# Introduction to Structures

- Structures in C allow the combination of different data types.
- They are used to represent a record.
- A structure is defined using the ‘struct‘ keyword.

# Defining a Structure

- A structure is defined as follows:

```
struct Person {
char name[50];
int age;
float salary;
};
```

# Accessing Structure Members

- Structure members are accessed using the dot operator.
- Example:

```
struct Person p1;
p1.age = 30;
```

# Initializing Structures

- Structures can be initialized at the time of declaration.

```
struct Person p1 = {"John", 30, 50000.0};
```

# Structure as Function Argument

- Structures can be passed to functions by value or by reference.
- Passing by reference is more efficient.

# Example: Function with Structure Argument

```
void printPerson(struct Person p) {
printf("%s is %d years old and earns %.2f",
p.name, p.age, p.salary);
}
```

# Structures and Pointers

- Pointers can be used to access structure members.
- The arrow operator (->) is used to access members via a pointer.

# Example: Pointer to Structure

```
struct Person *ptr;
ptr = &p1;
printf("%s", ptr->name);
```

# Nested Structures

- Structures can be nested, meaning one structure can contain another.
- This is useful for representing complex data.

# Example: Nested Structures

```
struct Address {
char city[30];
int zip;
};

struct Person {
char name[50];
struct Address addr;
};
```

# Summary

- Structures are a powerful feature in C for grouping related data.
- They can be used with functions and pointers effectively.
- Nested structures allow for more complex data models.

tcg crest
Inventing Harmonious Future