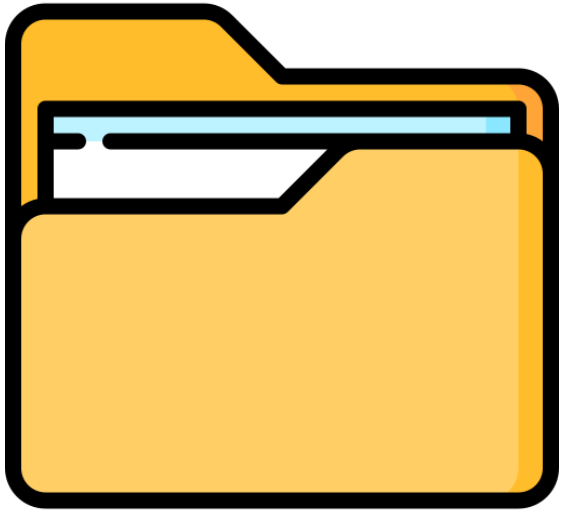
Program in C

Previous Classes

- Basic of C
- Variables & Data Types
- Input / Output Statements
- Types of Operator
- Decision making Statements
- Switch Statements
- Looping Statements
- Array & Dimensional array
- Strings & Methods
- Functions & Arguments
- Pointers
- Structures

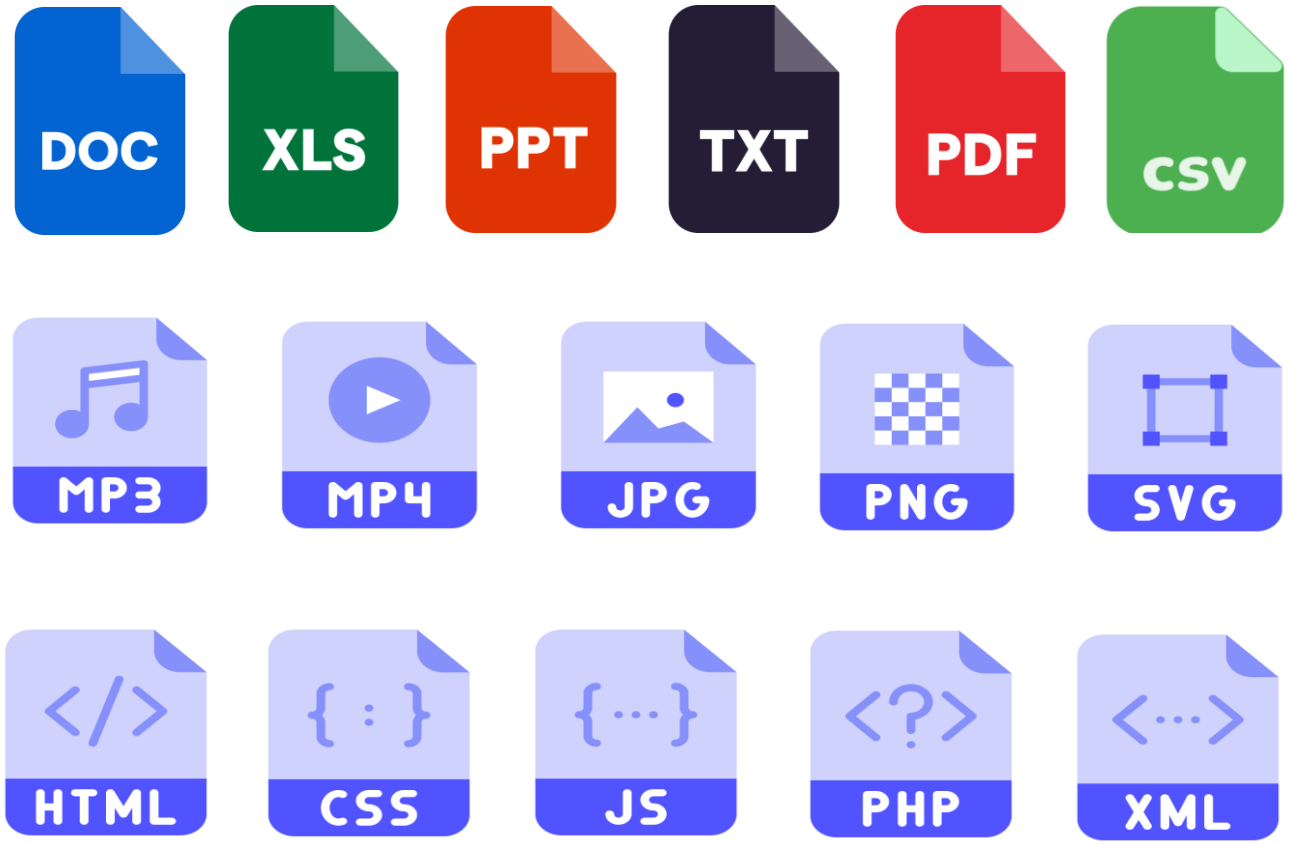
File Management

FILES



A file a container in a computer system that stores data, information, settings, or commands, which are used with a computer program

Some File Formats



Today's Content

- Open / close files
- Write data to files
- Reading data to files
- Error handling files
- Command line arguments

Special

Start-up Mni Project

Types of Files

TEXT File

A text file contains data in the form of ASCII characters and is generally used to store a stream of characters.

- ✓ Each line in a text file ends with a new line character ('\n').
- ✓ It can be read or written by any text editor.
- ✓ They are generally stored with .txt file extension.
- ✓ Text files can also be used to store the source code.

Binary File

A binary file contains data in binary form (i.e. 0's and 1's) instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- ✓ The binary files can be created only from within a program and their contents can only be read by a program.
- ✓ More secure as they are not easily readable.
- ✓ They are generally stored with .bin file extension.

Open / close Files

Declare Pointer Variable (syntax)

```
FILE *file_pointer_name;
```

Open File (syntax)

```
file_pointer_name = fopen(const char *file_name, const char *mode);
```

Close File (syntax)

```
fclose(FILE *file_pointer_name);
```

Example

```
#include <stdio.h>
int main(){
    char name[10];
    FILE *fp;
    fp = fopen("file.txt", "r");

    fclose(fp);
}
```

Open / close Files

Files Mode

Mode	Sort Description
r	Read text file
w	Write text file
a	Append text file
rb	Read binary file
wb	Write binary file
ab	Append binary file

r+	r + b / rb+
a+	a + b / ab+
w+	w + b / wb+

Example

```
#include <stdio.h>
int main(){
    char name[10];
    FILE *fp;
    fp = fopen("file.txt", "r");

    fclose(fp);
}
```

Write Data for Files

```
fprintf() – syntax

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

```
fputc() - syntax

int fputc(int c, FILE *stream);
```

```
fputs() - syntax

int fputs(const char *str, FILE
*stream);
```

```
fwrite() – syntax

int fwrite(const void *str,
size_t size, size_t count, FILE
*stream);
```

Write Data for Files

fprintf() - Example

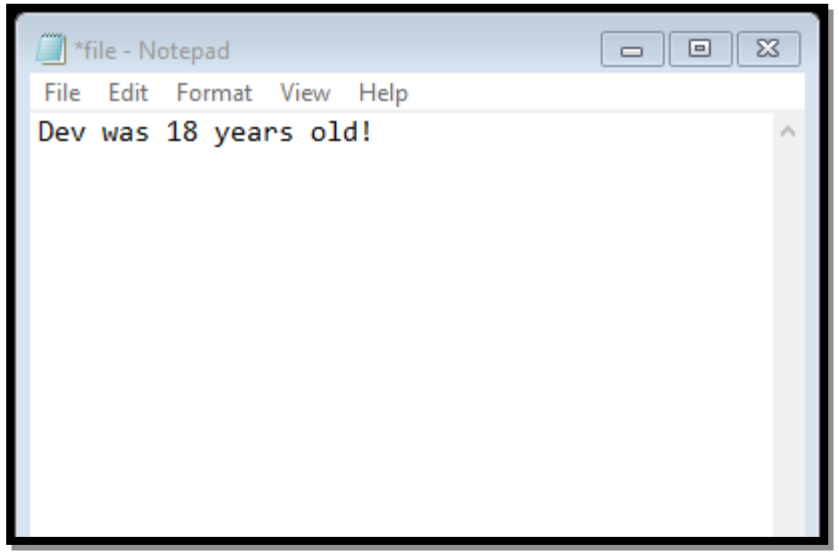
```c
#include <stdio.h>
int main(){
    char name[30];
    int age;
    FILE *fp;

    printf("Enter Name: ");
    gets(name);
    printf("Enter age: ");
    scanf("%d", &age);

    fp = fopen("file.txt", "w");
    fprintf(fp, "%s was %d years old.",
name, age);
    fclose(fp);

    printf("Data stored in file!");
}
```

Syntax

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

output

```
Enter Name: Dev
Enter age: 18
Data stored in file!
```



Write Data for Files

fputs() - Example

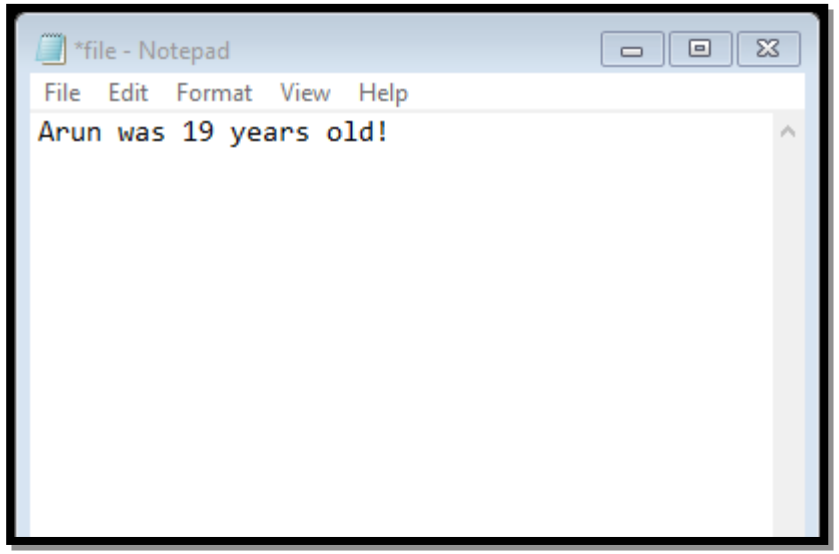
```c
#include <stdio.h>
int main(){
    char name[30];
    char age[12];
    FILE *fp;
    printf("Enter Name: ");
    gets(name);
    printf("Enter age: ");
    gets(age);
    fp = fopen("file.txt", "w");
    fputs(name, fp);
    fputs(" was ", fp);
    fputs(age, fp);
    fputs(" years old! ", fp);
    fclose(fp);
    printf("Data stored in file!");
}
```

syntax

```c
int fputs(const char *str, FILE *stream);
```

output

```
Enter Name: Arun
Enter age: 19
Data stored in file!
```



Write Data for Files

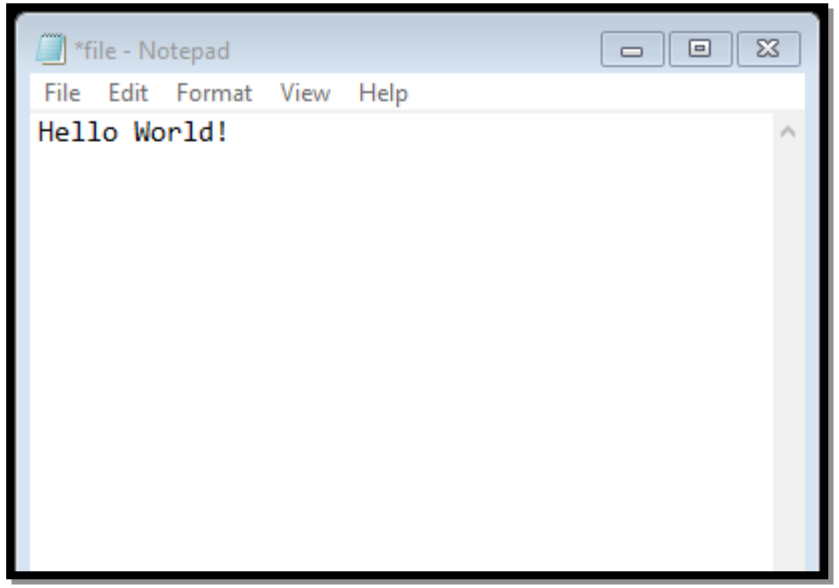
fputc() - Example

```c
#include <stdio.h>
#include <string.h>
int main(){
    char text[30];
    FILE *fp;

    printf("Enter some character: ");
    gets(text);

    fp = fopen("file.txt", "w");
    for (int i = 0; i<strlen(text); i++){
        fputc(text[i], fp);
    }
    fclose(fp);

    printf("Data stored in file!");
}
```

```c
syntax
int fputc(int c, FILE *stream);
```

output

```
Enter some character: Hello World!
Data stored in file!
```



Write Data for Files

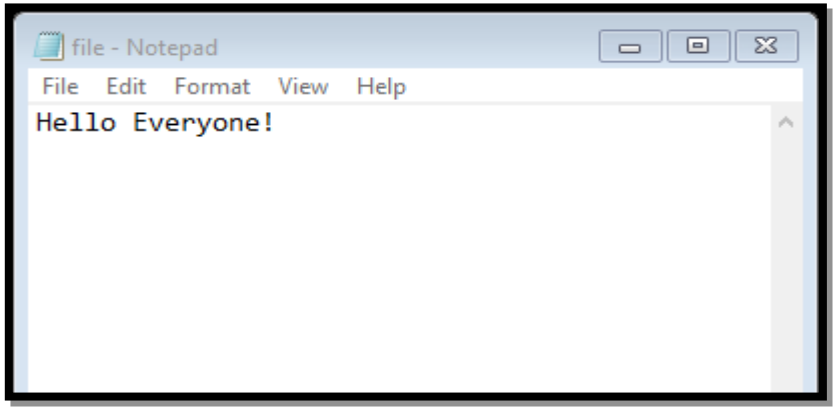
fwrite() - Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(){
    size_t count;
    char text[] = "Hello Everyone!";
    FILE *fp;

    fp = fopen("file.txt", "w");
    count = fwrite(text, 1, strlen(text),
fp);
    fclose(fp);

    printf("%d bytes stored in file!",
count);
}
```

```
syntax
fwrite(const void *str, size_t size,
size_t count, FILE *stream);
```

output

```
15 bytes stored in file!
```



Read Data for Files

```
fscanf() – syntax

int fscanf(FILE *stream, const
char *format, ...);
```

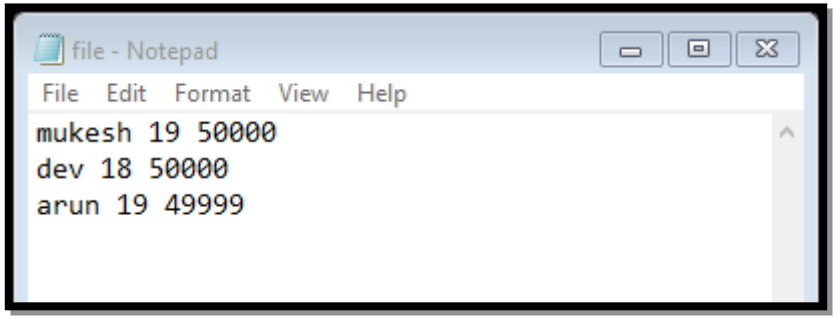
```
fgetc() - syntax

int fgetc(FILE *stream);
```

```
fgets() - syntax

char *fgets(char *str, int size,
FILE *stream);
```

```
fread() – syntax

int fread(void *str, size_t
size, size_t num, FILE *stream);
```

Read Data for Files

fscanf() - Example



Syntax

```
int fscanf(FILE *stream, const char *format, ...);
```

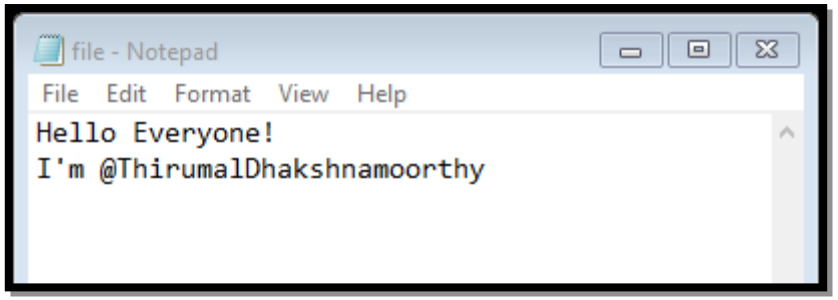
output

```
Name: mukesh    Age: 19    Salary: 50000
Name: dev       Age: 18    Salary: 50000
Name: arun      Age: 19    Salary: 49999
```

```
#include <stdio.h>
#include <string.h>
int main(){
    char name[30];
    int age, salary, i=1;
    FILE *fp;
    fp = fopen("file.txt", "r");
    while(i<4){
        fscanf(fp, "%s %d %d", name, &age, &salary);
        printf("Name: %s\tAge: %d\tSalary: %d\n",name, age, salary);
        i++;
    }
    fclose(fp);
}
```

Read Data for Files

fgets() - Example



Syntax

```
int fgets(char *str, int size, FILE *stream);
```

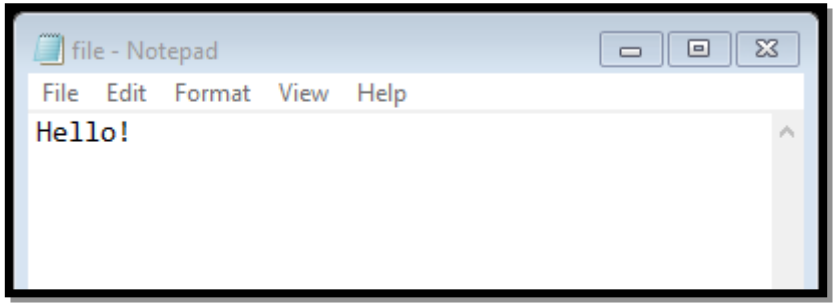
output

```
Hello Everyone!
I'm @ThirumalDhakshnamoorthy
```

```
#include <stdio.h>
#include <string.h>
int main(){
    char text[150];
    FILE *fp;
    fp = fopen("file.txt", "r");
    while(!feof(fp)){
        fgets(text, 150, fp);
        printf("%s", text);
    }
    fclose(fp);
}
```

Read Data for Files

fgetc() - Example




```
Syntax
int fgetc(FILE *stream);
```

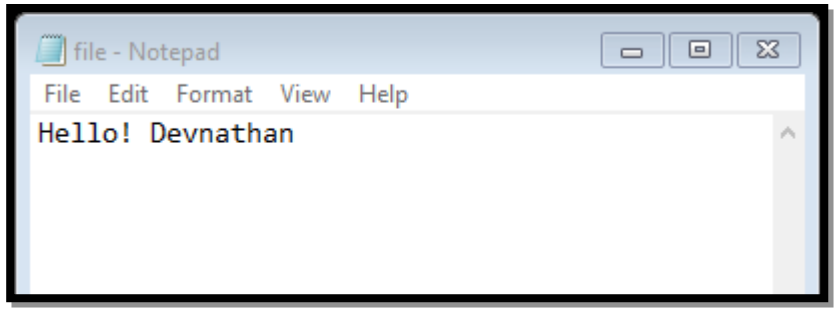
output

```
Hello!
```

```c
#include <stdio.h>
#include <string.h>
int main(){
    char ch;
    FILE *fp;
    fp = fopen("file.txt", "r");
    while(!feof(fp)){
        ch = fgetc(fp);
        printf("%c", ch);
    }
    fclose(fp);
}
```

Read Data for Files

fread() - Example




```
syntax
int fread(void *str, size_t size,
size_t num, FILE *stream);
```

output

```
Hello! Devnathan
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
    char text[100];
    size_t count;
    FILE *fp;
    fp = fopen("file.txt", "r");
    while(!feof(fp)){
        count = fread(text, 1, 20, fp);
        printf("%d : %s\n", count,
text);
    }
    fclose(fp);
}
```

Error Handling

In C programming, error handling is essential to handle unexpected situations or failures that may occur during program execution.

```
ferror()
clearerr()
perror()
```

```
strerr()
feof()
```

Error Handling

perror()
show the error description

syntax

```
void perror(char *msg);
```

Output

```
Error: No such file or directory
```

```
#include <stdio.h>
int main()
{
    FILE* file = fopen("needed_file.txt", "r");
    if (file == NULL) {
        perror("Error");
        return 1; }
    fclose(file);
    return 0;
}
```

Error Handling

ferror()
check whether an error occurred during a file operation.

```
syntax
int ferror(char *msg);
```

Output

```
File is opened in writing mode! You cannot read data from it!
```

```
#include <stdio.h>
int main(){

    FILE *fp;
    fp = fopen("test.txt","w");
    char ch = fgetc(fp);
    if(ferror(fp)){
        printf("File is opened in writing
mode! You cannot read data from it!");
    }
    fclose(fp);
    return(0);
}
```

Error Handling

clearerr()
Used to clear the end-of-file and error indicators for the stream.

```
syntax
void clearerr(FILE *fp);
```

Output

```
File is opened in writing mode! You cannot read data from it!
```

```c
#include <stdio.h>
#include <errno.h>
void main(){
    FILE *fp;
    char text[100];
    fp = fopen("new_file.txt", "r");
    if(fp == NULL){
        clearerr(fp);
    }
    printf("Enter some Text: ");
    gets(text);

    fprintf(fp, "%s", text);

    fclose(fp);
}
```

Command line argument

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

```
syntax
int main(int argc, char *argv[])
```

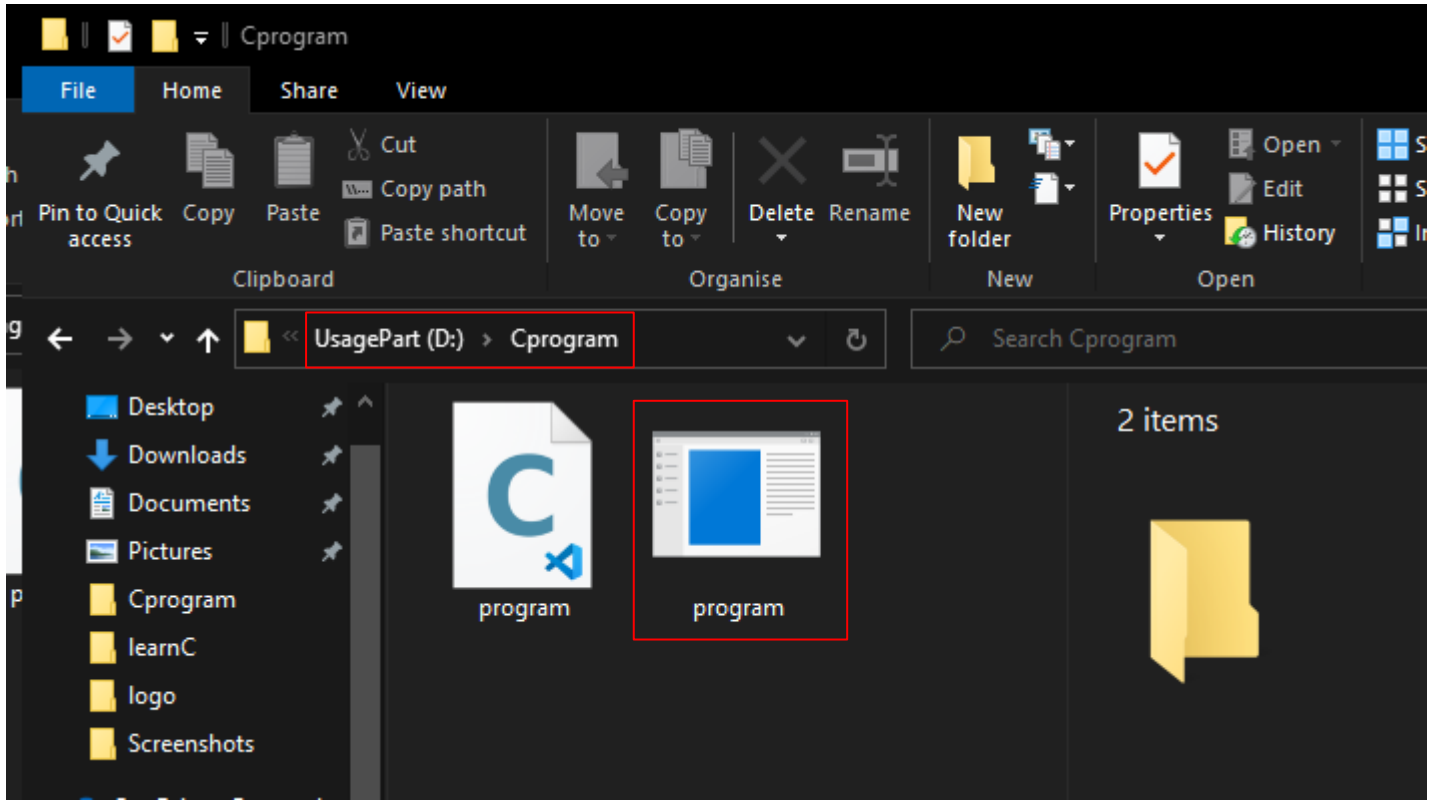
argc - counts the number of arguments. It counts the file name as the first argument.

argv[] - contains the total number of arguments. The first argument is the file name always.

Command line argument

```c
#include <stdio.h>
int main(int argc, char
*argv[]){
    // printf("Hello");
    printf("\n%d", argc);
    for (int i = 0; i<argc;
i++){
        printf("\n%s",
argv[i]);
    }
    return 0;
}
```

Write program and once run.



Command line argument

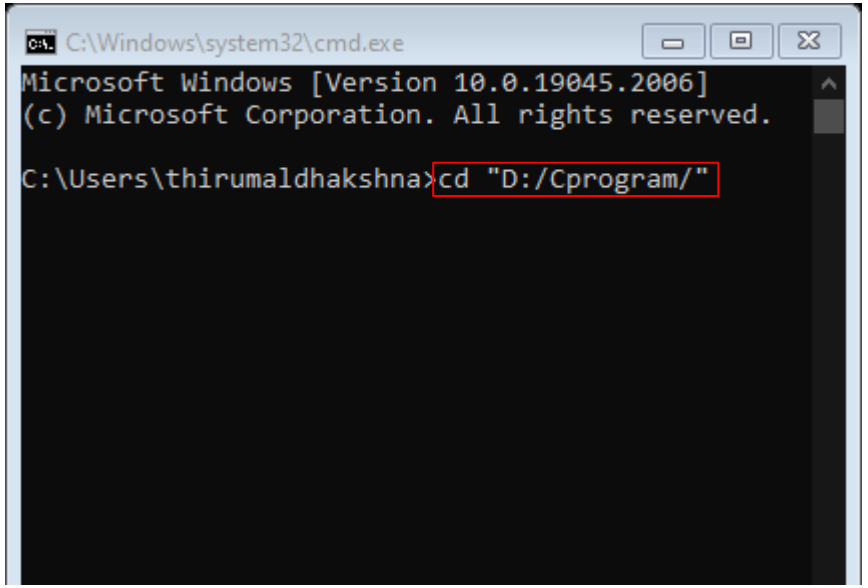
Run ‘program.exe’ via terminal or command prompt

```
#include <stdio.h>
int main(int argc, char
*argv[]){
    // printf("Hello");
    printf("\n%d", argc);
    for (int i = 0; i<argc;
i++){
        printf("\n%s",
argv[i]);
    }
    return 0;
}
```

Command line argument

```c
#include <stdio.h>
int main(int argc, char
*argv[]){
    // printf("Hello");
    printf("\n%d", argc);
    for (int i = 0; i<argc;
i++){
        printf("\n%s",
argv[i]);
    }
    return 0;
}
```

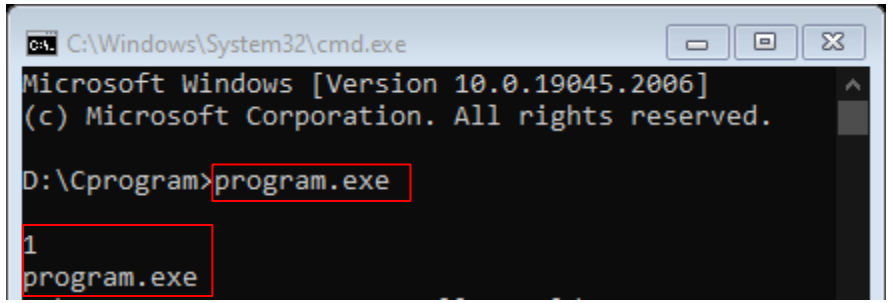
Run ‘program.exe’ via terminal or command prompt



Command line argument

```c
#include <stdio.h>
int main(int argc, char
*argv[]){
    // printf("Hello");
    printf("\n%d", argc);
    for (int i = 0; i<argc;
i++){
        printf("\n%s",
argv[i]);
    }
    return 0;
}
```

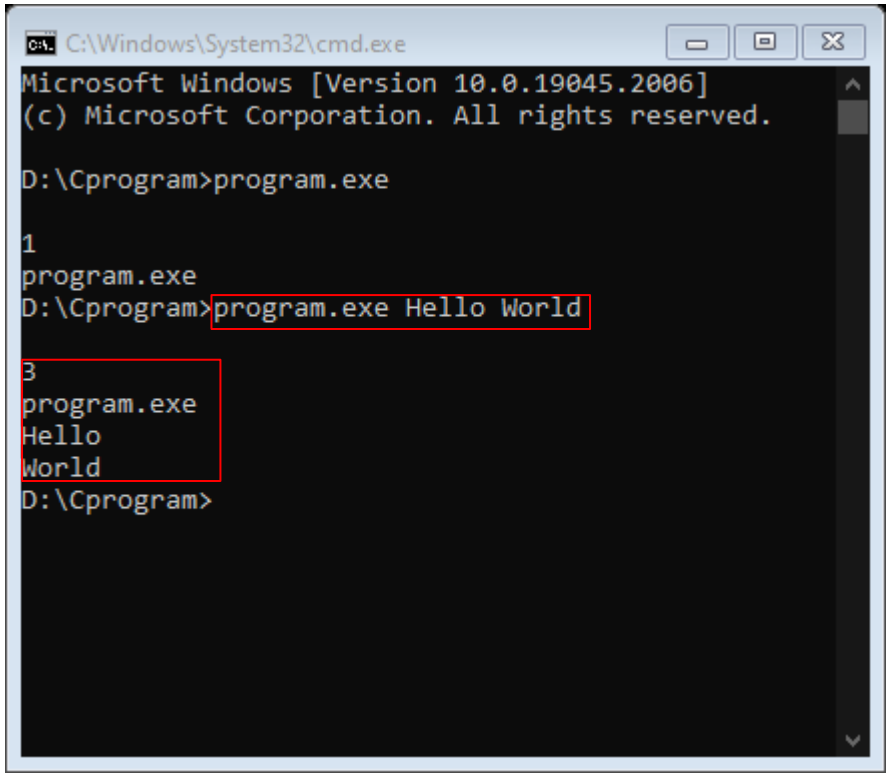
Run ‘program.exe’ via terminal or command prompt



Command line argument

```c
#include <stdio.h>
int main(int argc, char
*argv[]){
    // printf("Hello");
    printf("\n%d", argc);
    for (int i = 0; i<argc;
i++){
        printf("\n%s",
argv[i]);
    }
    return 0;
}
```

Run ‘program.exe’ via terminal or command prompt



Reference:

Reema Thareja

Assistant Professor
Dept of Computer Science
Shyama Prasad Mukherji College for Woman
University of Delhi

Geeks for Geeks

Online Platform of Technical teach site

Special

Start-up Mni Project