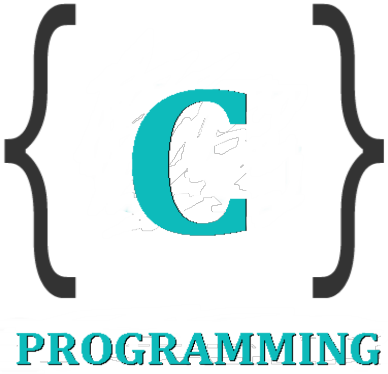
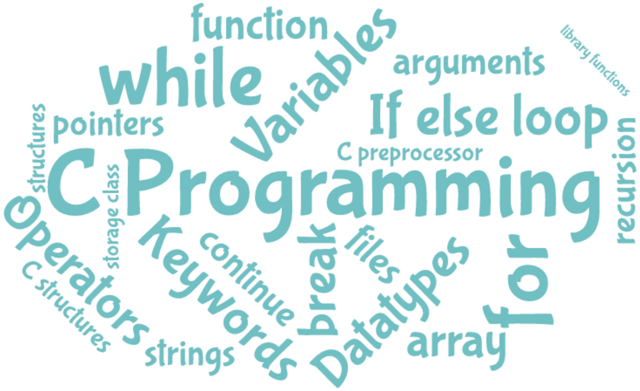


IT 112: Introduction to Programming



Dr. Manish Khare
Dr. Bakul Gohel

DATA TYPE REVISITED

There are four storage classes in C

- Automatic storage class
- Static storage Class
- External storage class
- Register storage class

Automatic storage class

- Scope: local to the block in which the variable is defined
- Default value is garbage value if variable is not initialized
- Life: till the control remains within the block in which the variable is defined
- Local variable stored at stack segment in memory layout
- It is a default storage class for all local variable

Static storage Class

- Default value is Zero
- Stored at data segment in memory layout
- Scope: Local to the block in which the variable is defined
- Life: value of the variable persists between the different function calls

```
#include<stdio.h>

int fun_0();
int fun_1();

int main()
{
 for(int i=0;i<5;i++)
 {
  printf("fun_0:%d , fun_1:%d\n", fun_0() , fun_1());
 }
 return 0;
}
```

```
int fun_0()
{
  int count = 0;  // auto class
  count++;
  return count;

}

int fun_1()
{
  static int count = 0;  //static class
  count++;
  return count;
}
```

OUTPUT:

```
fun_0:1 , fun_1:1
fun_0:1 , fun_1:2
fun_0:1 , fun_1:3
fun_0:1 , fun_1:4
fun_0:1 , fun_1:5
```

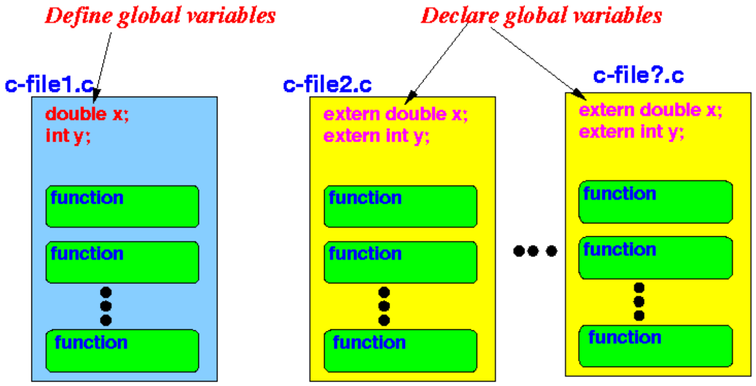
Extern Storage class

- Default value is Zero
- Scope: global
- Life: as long as program's execution does not come to an end
- Stored at data segment in memory layout

- extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.

Extern Storage class

- The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.



Program_file.c

```
#Include <functon_file.h>

int global_var = 3;
int main()
{
 fun(); // WORKING: it shows '4'
}
```

funciton_file.h

```
void fun(){
    extern int global_var;//Note: 'int global_var' without 'extern' would
                              // simply create a separate different variable
    ++global_var;         // and '++global_var' wouldn't work since it'll
                             // complain that the variable was not initiazed.
    printf(“%d”, global_var);
}
```

Register Class

- Storage at CPU register (if availalbe)
- Default value: Garbage
- Scope: local to block in which the variable is defined
- Life : Till control remain to block
- CPU registers are very limited in size
- Value stored in the CPU register can be accessed faster tehan the one that is stored in memory

```
register int I;
for(i=1;i<=100000;i++)
    // do some computation
```

PRE-PROCESSOR

PROCESS BEFORE COMPILE

MACRO EXAPNASION

#define NICK_NAME ACTUAL_ONE

- Constant
- Faster and more compact processing
- Variable may inadvertently get altered

```
#include <stdio.h>
#define PI 3.1415

int main()
{
    float radius, area;
    printf("Enter the radius: ");
    scanf("%f", &radius);

    // Notice, the use of PI
    area = PI*radius*radius;

    printf("Area=%.2f",area);
    return 0;
}
```

```
#define AND &&
#define  OR   ||
```

FUNCTION like MACRO

- #define circleArea(r) (PI*r*r)
- #define ISDIGIT(y) (y>=48 && y<=57)

- *Do not give space between the macro template and its arguments*

- *Be careful*
  - #define SQUARE(n) n*n
  - Value = 100/SQUARE(4)

    → Value = 100/4*4 not 100/64

- *Make macro function simple and sweet*

```
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)

int main() {
    float radius, area;

    printf("Enter the radius: ");
    scanf("%f", &radius);
    area = circleArea(radius);
    printf("Area = %.2f", area);

    return 0;
}
```

#include

one file to be included in another file

- #include “filename” // search in current directory and specified path
- #include <filename> // search in specified path only

```
// C program to illustrate file inclusion
// <> used to import system header file
#include <stdio.h>

// " " used to import user-defined file
#include "process.h"

// main function
int main()
{
    // add function defined in process.h
    add(10, 20);

    // mult function defined in process.h
    mult(10, 20);

    // printf defined in stdio.h
    printf("Process completed");
    return 0;
}
```

“process.h”

```
// It is not recommended to put function definitions
// in a header file. Ideally there should be only
// function declarations. Purpose of this code is
// to only demonstrate working of header files.

void add(int a, int b)
{
    printf("Added value=%d\n", a + b);
}

void multiply(int a, int b)
{
    printf("Multiplied value=%d\n", a * b);
}
```

#ifdef / #ifndef / #undef

```
#include <stdio.h>

#define UNIX 1.0

int main()
{
   #ifdef UNIX
         printf("UNIX specific function calls go here.\n");
        // code specific for UNIX system
    #else
         // code specific for INTEL  system

   #endif

   // common code is here

   return 0;
}
```

```
#include <stdio.h>

#define UNIX 1.0
#ifndef UNIX
    #define UNIX 0.5
#endif

int main()
{
   // some code is here using UNIX
   return 0;
}
```

```
#include <stdio.h>

#ifndef  __myfile_h
    #define __myfile_h
    #include “myfile.h”
#endif

int main()
{
   // some code is here
   return 0;
}
```

#pragma

- Pragma is special purpose directive that you can turn on and turn of certain feature of your programme
- **Functionality highly vary from compiler to compiler**

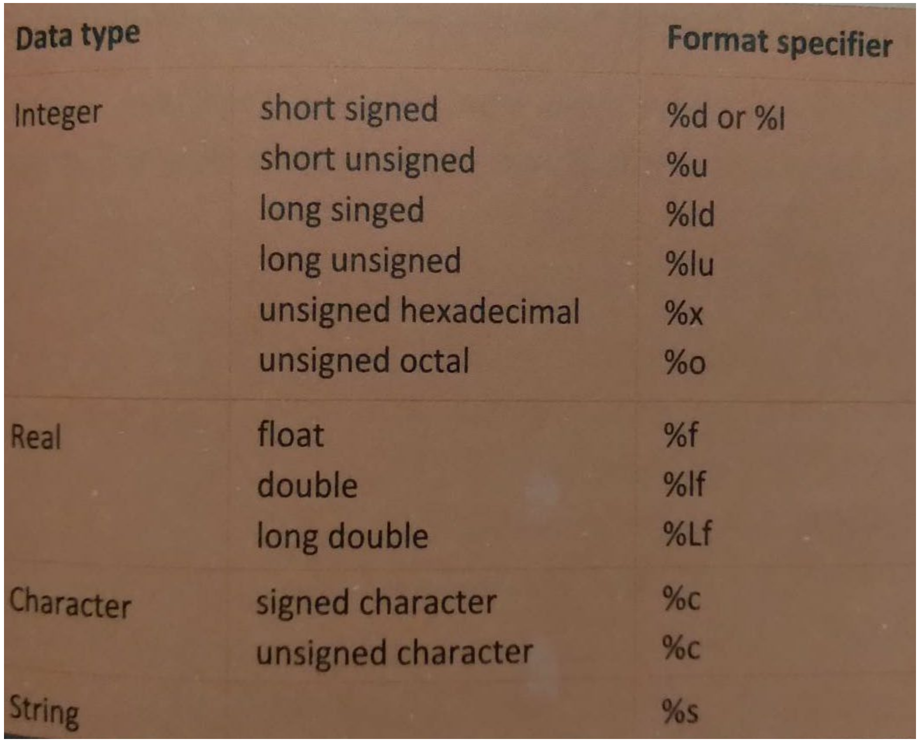
```
include<stdio.h>

void func1();
void func2();

#pragma startup func1
#pragma exit func2

void func1()
{
    printf("Inside func1()\n");
}

void func2()
{
    printf("Inside func2()\n");
}

int main()
{
    printf("Inside main()\n");

    return 0;
}
```

Output:

```
        Inside func1()
        Inside main()
        Inside func2()
```

CONSOLE I/O

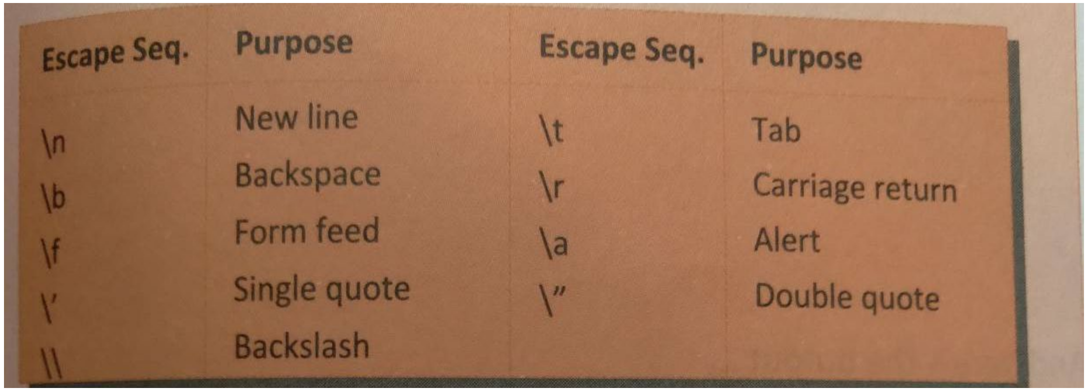
Formatted I/O



Data type		Format specifier
Integer	short signed	%d or %I
	short unsigned	%u
	long singed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	float	%f
	double	%lf
	long double	%Lf
Character	signed character	%c
	unsigned character	%c
String		%s

```
85       /* decimal */
0213      /* octal */
0x4b      /* hexadecimal */
30       /* int */
30u       /* unsigned int */
30l      /* long */
30ul      /* unsigned long */
```

FORMATED I/O



Escape Seq.	Purpose	Escape Seq.	Purpose
\n	New line	\t	Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\”	Double quote
\\	Backslash		

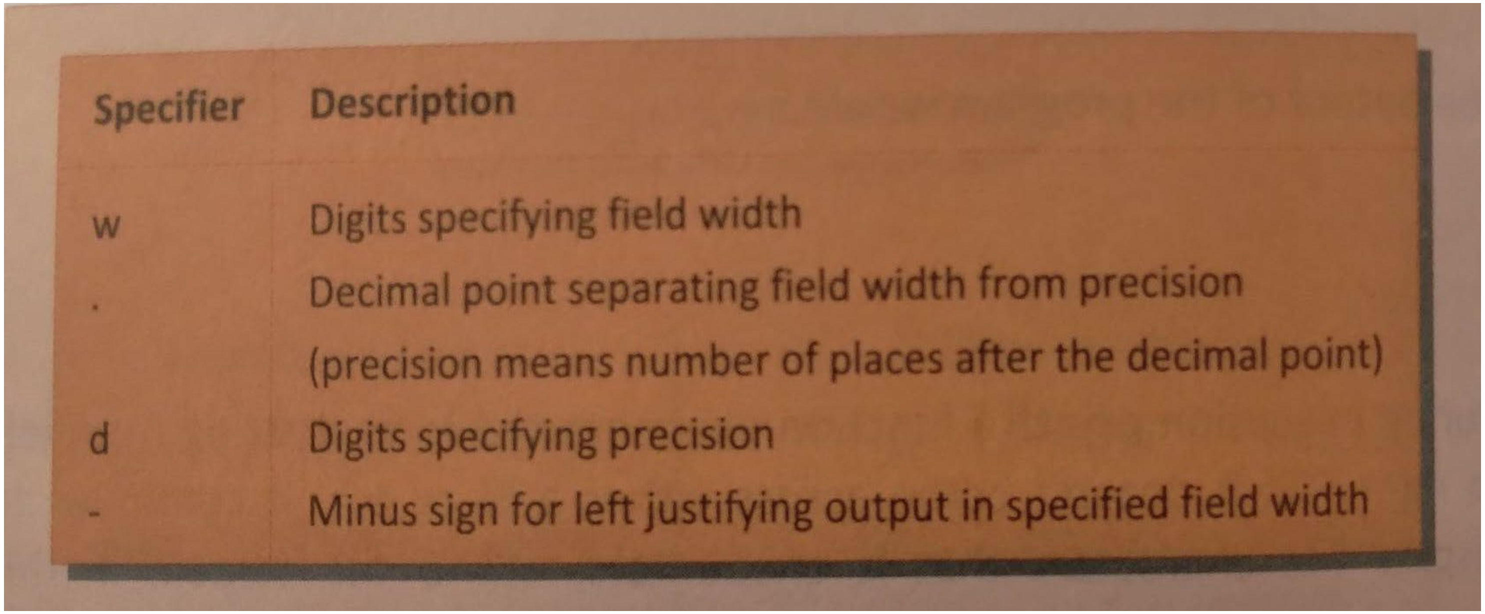
```
#include <stdio.h>

int main()
{
   printf("\n\n-----\\n---\n");
   printf("Hello World\nBakul");

   printf("\n\n-----\\t---\n");
   printf("\tHello World\n\tBakul");

   printf("\n\n-----\\r---\n");
   printf("Hello World\rBakul");

   printf("\n\n-----\\f---\n");
   printf("Hello World\fBakul");

   return 0;
}
```

OUTPUT:

```
-----\n---
Hello World
Bakul

-----\t---
        Hello World
        Bakul

-----\r---
Bakul World

-----\f---
Hello World
           Bakul
```

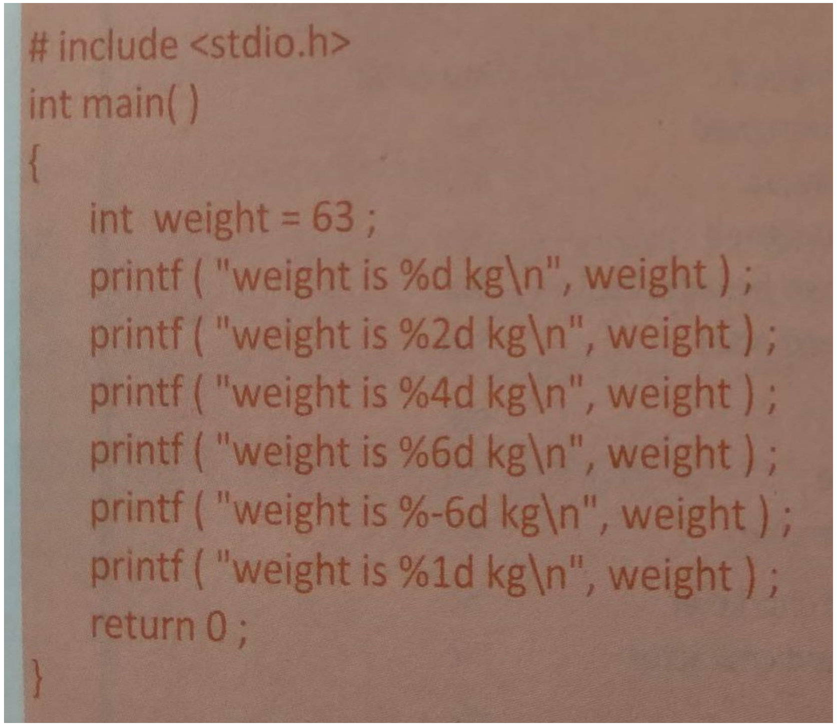
Formatted I/O

```
printf("%f",var)
printf("%-w.df",var)
```

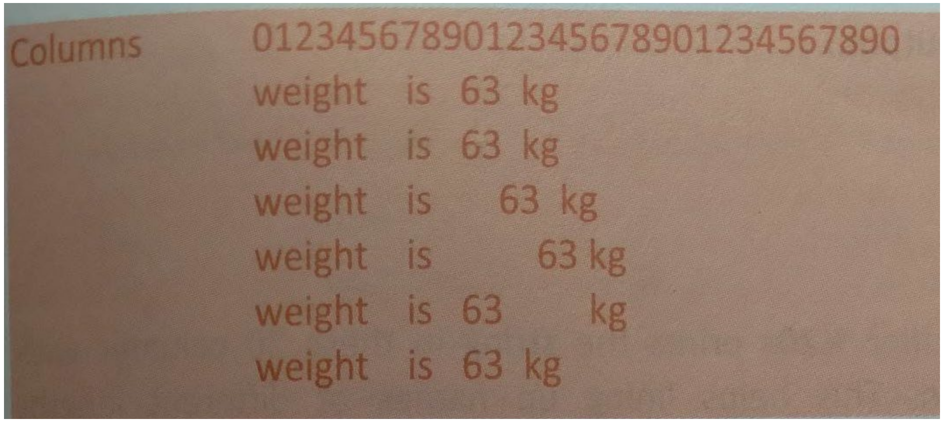


Specifier	Description
w	Digits specifying field width
.	Decimal point separating field width from precision (precision means number of places after the decimal point)
d	Digits specifying precision
-	Minus sign for left justifying output in specified field width

Formatted I/O



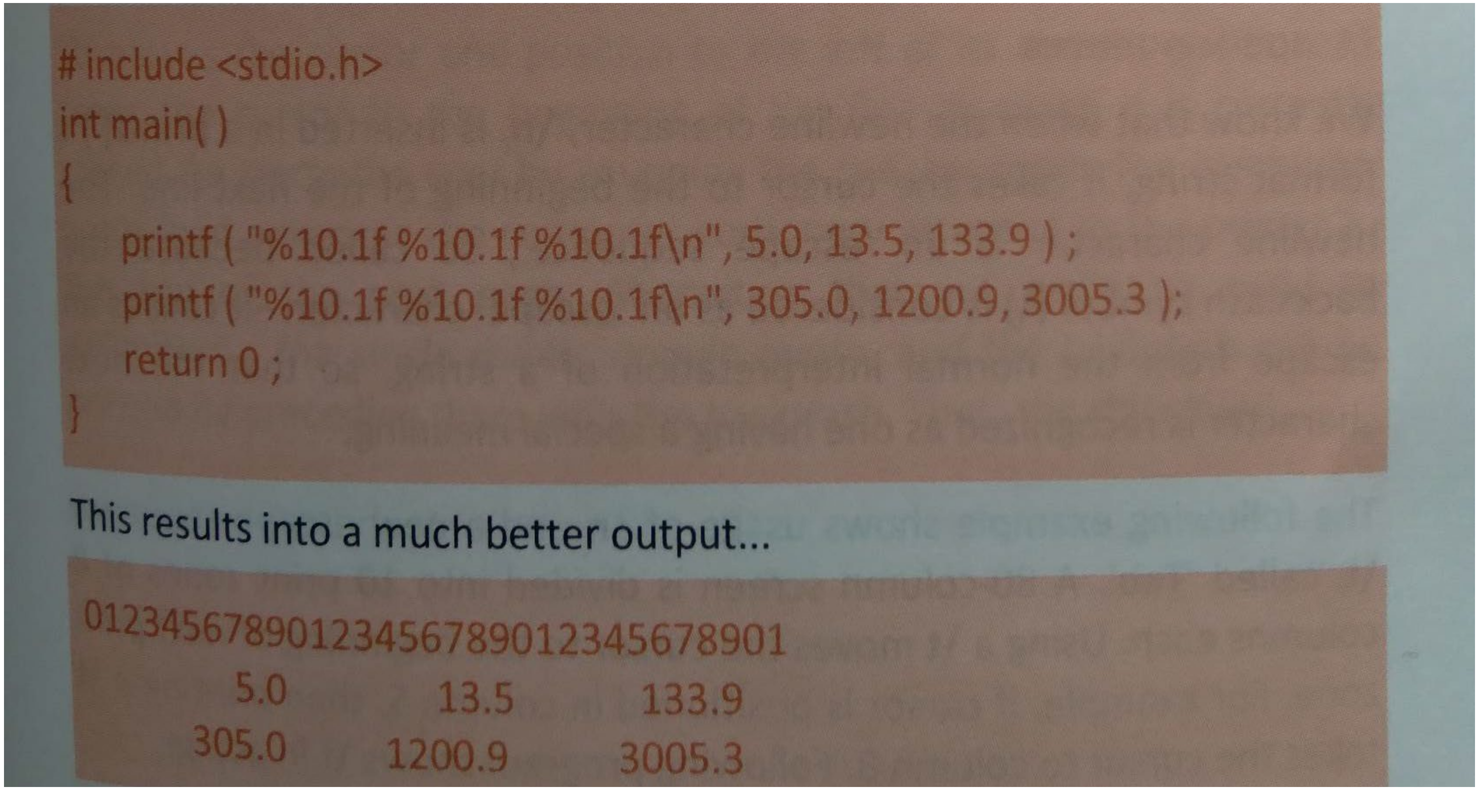
```
# include <stdio.h>
int main( )
{
    int  weight = 63 ;
    printf ( "weight is %d kg\n", weight ) ;
    printf ( "weight is %2d kg\n", weight ) ;
    printf ( "weight is %4d kg\n", weight ) ;
    printf ( "weight is %6d kg\n", weight ) ;
    printf ( "weight is %-6d kg\n", weight ) ;
    printf ( "weight is %1d kg\n", weight ) ;
    return 0 ;
}
```



```
Columns    0123456789012345678901234567890
           weight is 63 kg
           weight is 63 kg
           weight is   63 kg
           weight is     63 kg
           weight is 63     kg
           weight is 63 kg
```

printf("%04d",var) : padding with zero instead of space

Formatted I/O



```
# include <stdio.h>
int main( )
{
    printf ( "%10.1f %10.1f %10.1f\n", 5.0, 13.5, 133.9 ) ;
    printf ( "%10.1f %10.1f %10.1f\n", 305.0, 1200.9, 3005.3 );
    return 0 ;
}
```

This results into a much better output...

```
01234567890123456789012345678901
       5.0      13.5     133.9
     305.0    1200.9    3005.3
```

printf("%05.2f",3.14) : 03.14

Formated I/O

- sscanf() Read from string
- sprintf() write to string

```
#include <stdio.h>
int main()
{
    int i=10;
    char ch='A';
    float a=3.14;
    char str[20];

    printf("%d %c %f\n",i,ch,a);
    sprintf(str, "%d %c %f\n",i,ch,a);
    printf("%s\n",str);

    return 0;
}
```

```
10 A 3.140000
10 A 3.140000
```

```
#include <stdio.h>
#include <string.h>


int main () {
   int day, year;
   char weekday[20], month[20], dtm[100];

   strcpy( dtm, "Saturday March 25 1989" );
   sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );

   return(0);
}
```

```
March 25, 1989 = Saturday
```

UNFORMATED I/O

```
#include <stdio.h>
int main()
{
    char ch;
    printf("\nType any alphabet: ");
    ch=getchar(); //must be followed by enter key
    printf("you typed: ");
    putchar(ch);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char footballer[40];
    puts("Enter name:");
    gets(footballer);    // now deprecated
    puts("Congratess...");
    puts(footballer);
    return 0;
}
```

FILE I/O

FILE I/O

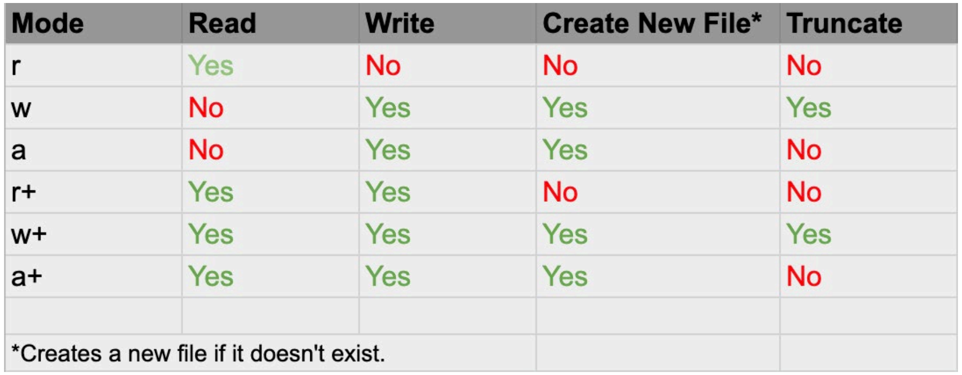
```
FILE *fp ;
```

file pointer

```
fp = fopen(file_name, Mode) ;
```

file name in string

Operation mode

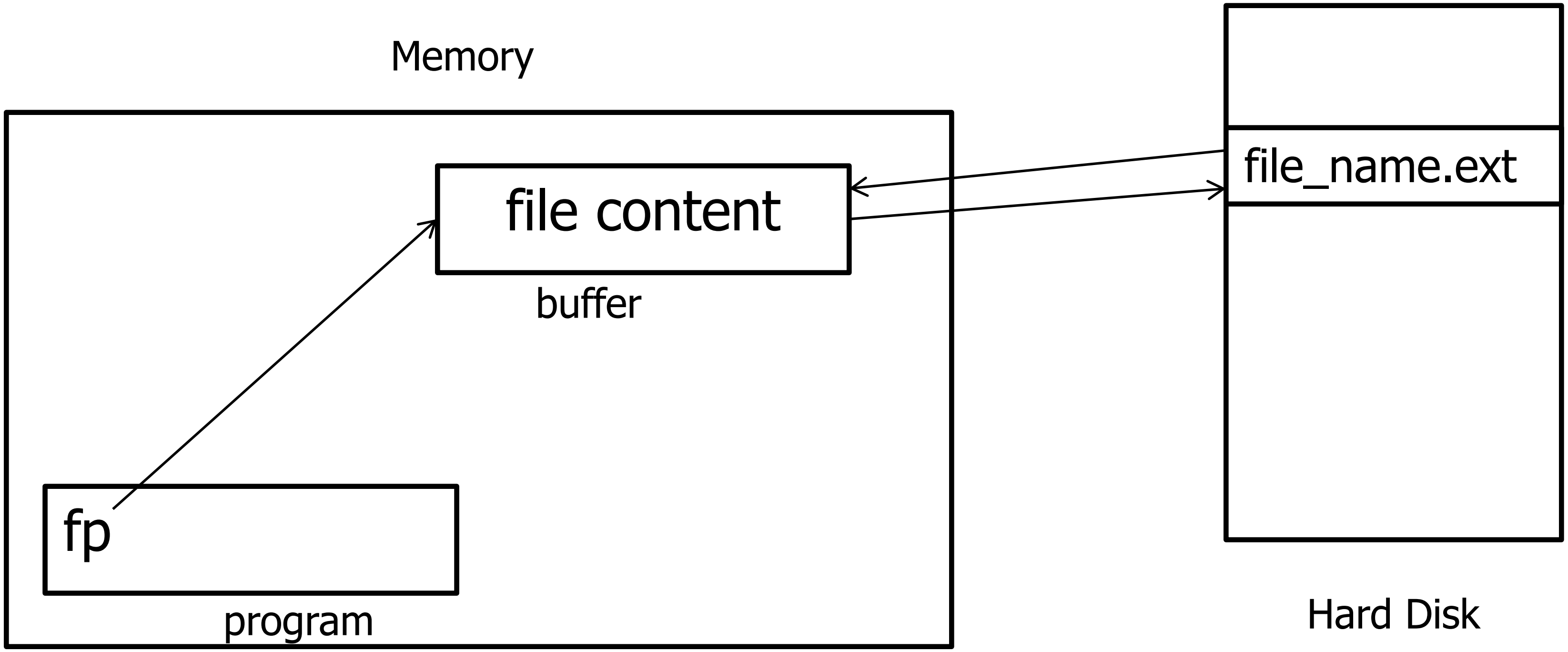


Mode	Read	Write	Create New File*	Truncate
r	Yes	No	No	No
w	No	Yes	Yes	Yes
a	No	Yes	Yes	No
r+	Yes	Yes	No	No
w+	Yes	Yes	Yes	Yes
a+	Yes	Yes	Yes	No

*Creates a new file if it doesn't exist.

```
FILE  *fp;

fp = fopen("newfile.txt", "r");
                              OR
fp = fopen("C:\\myfiles\\newfile.txt", "r");
```

FILE I/O


Memory
file content
buffer
fp
program
file_name.ext
Hard Disk

Read and write single char

```
int fgetc( FILE * fp );
```

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

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

    /* Pointer for both the file*/
    FILE *fpr, *fpw;
    /* Opening file FILE1.C in “r” mode for reading */
    fpr = fopen("C:\\file1.txt", "r");

    /* Ensure FILE1.C opened successfully*/
    if (fpr == NULL)
    {
         puts("Input file cannot be opened");
    }

 /* Opening file FILE2.C in “w” mode for writing*/
    fpw= fopen("C:\\file2.txt", "w");

    /* Ensure FILE2.C opened successfully*/
    if (fpw == NULL)
    {
       puts("Output file cannot be opened");
    }
```

```
    /*Read & Write Logic*/
    while(1)
    {
        ch = fgetc(fpr);
        if (ch==EOF)
            break;
        else
            fputc(ch, fpw);
    }

    /* Closing both the files */
    fclose(fpr);
    fclose(fpw);

    return 0;
}
```

Read and write string

```
char *fgets( char *buf, int n, FILE *fp )
```

```
int fputs( const char *s, FILE *fp )
```

If fgets function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters then it returns only the characters read up to that point including the new line character.

```
#include <stdio.h>

main() {

   FILE *fp;
   char buff[255];

   fp = fopen("/tmp/test.txt", "r");

   fgets(buff, 255, fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

```
#include <stdio.h>

main() {
   FILE *fp;

   fp = fopen("/tmp/test.txt", "w+");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

Read and write string

int fscanf(FILE *stream, const char *format [, argument, ...])

int fprintf(FILE *stream, const char *format [, argument, ...])

Similar to sscanf and sprintf as seen above
Only different is first argument that is file pointer instead of string

```
#include <stdio.h>

int main ()
{
  char str [80];
  float f;
  FILE * pFile;

  pFile = fopen ("myfile.txt","w+");
  fprintf (pFile, "%f %s", 3.1416, "PI");
  rewind (pFile);
  fscanf (pFile, "%f", &f);
  fscanf (pFile, "%s", str);
  fclose (pFile);
  printf ("I have read: %f and %s \n",f,str);
  return 0;
}
```

FILE I/O

Command line argument

```
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

programe.c

```
#include <stdio.h>
#include <conio.h>

int main(int argc, char *argv[])
{
   int i;
   if( argc >= 2 )
   {
      printf("The arguments supplied are:\n");
      for(i = 0; i < argc; i++)
      {
         printf(“%d : %s\n",i, argv[i]);
      }
   }
   else
   {
      printf("argument list is empty.\n");
   }
   return 0;
}
```

- Compiling programme's file give executable file programe.exe
- Run executable file on command prompt

```
>>./programe.exe   file.txt  100
```

```
The arguments supplied are:
0 : programe.exe
1 : file.txt
2 : 100
```

TYPEDEF

typedef

```
typedef unsigned char BYTE;

 BYTE  b1, b2;

  b1=255
  b2=1;
```

typedef and structure

```
#include<stdio.h>

struct Point{
  int x;
  int y;
};
typedef struct Point Point;

int main() {
    Point p1;
    p1.x = 1;
    p1.y = 3;
    printf("%d \n", p1.x);
    printf("%d \n", p1.y);
    return 0;
}
```

```
#include<stdio.h>

typedef struct Point{
  int x;
  int y;
} Point;
int main() {
    Point p1;
    p1.x = 1;
    p1.y = 3;
    printf("%d \n", p1.x);
    printf("%d \n", p1.y);
    return 0;
}
```