CSE 1320

Week of 04/03/2023

Instructor : Donna French

```
Enter the radius of the circle 1
The area of your circle is 3.141593


Enter the length of one side 2
The area of your square is 4.000000


Enter the length of side 1 4
Enter the length of side 2 5
The area of your rectangle is 20.000000


Enter the length of the base 2
Enter the length of the height 5
The area of your triangle is 5.000000
```

```
Find the area of a shape

1. Circle
2. Square
3. Rectangle
4. Triangle

Enter choice
```

typedef3Demo.c

Typedefs

Structures are a good use of typedefs

```
typedef struct
{
   float radius;
}
CIRCLE;
```

```
typedef struct
{
   float side;
}
SQUARE;
```

```
typedef struct
{
   float side1;
   float side2;
}
RECTANGLE;

typedef struct
{
   float base;
   float height;
}
TRIANGLE;
```

```
union shape
{
   CIRCLE circle;
   SQUARE square;
   RECTANGLE rectangle;
   TRIANGLE triangle;
};

enum shapes
{
   circle=1, square, rectangle, triangle
};
```

Note about using the `math` library

- Using `abs()`
  - Do not need `math.h`
  - `abs()` is part of `stdlib.h`

- Using the math library requires an extra compiler command
- Compiling a program containing this line…

```
result = pow(base, exponent);
```

```
[frenchdm@omega ~]$ gcc mathDemo.c
/tmp/cc2NMDDA.o: In function `main':
mathDemo.c:(.text+0x76): undefined reference to `pow'
collect2: ld returned 1 exit status
```

`#include <math.h>` is in the program

```
[frenchdm@omega ~]$ gcc mathDemo.c -lm
[frenchdm@omega ~]$
```

`-lm` tells the compiler to link the math library

```
printf("The area of your circle is %f\n",
       M_PI * pow(EnteredShape.circle.radius, 2));
```

```
[frenchdm@omega ~]$ gcc typedef3Demo.c
[frenchdm@omega ~]$
```

The compiler optimized the call to `radius * radius` and did not use `pow()`

```
printf("The area of your circle is %f\n",
       M_PI * pow(EnteredShape.circle.radius, 3));
```

```
[frenchdm@omega ~]$ gcc typedef3Demo.c
/tmp/ccAIzQQ5.o: In function `main':
typedef3Demo.c:(.text+0xc6): undefined reference to `pow'
typedef3Demo.c:(.text+0x13b): undefined reference to `pow'
collect2: ld returned 1 exit status

[frenchdm@omega ~]$ gcc typedef3Demo.c -lm
[frenchdm@omega ~]$
```

```
int MyShape;
union shape EnteredShape;

printf("Find the area of a shape\n\n");

printf("1. Circle\n"
       "2. Square\n"
       "3. Rectangle\n"
       "4. Triangle\n\n"
       "Enter choice ");

scanf("%d", &MyShape);
```

```
Find the area of a shape

1. Circle
2. Square
3. Rectangle
4. Triangle

Enter choice
```

```
union shape EnteredShape;

(gdb) ptype EnteredShape
type = union shape {
    CIRCLE circle;
    SQUARE square;
    RECTANGLE rectangle;
    TRIANGLE triangle;
}



(gdb) p
sizeof(EnteredShape)
$1 = 8
```

```
(gdb) ptype EnteredShape.circle
type = struct {
    float radius;
}
(gdb) ptype EnteredShape.square
type = struct {
    float side;
}
(gdb) ptype EnteredShape.rectangle
type = struct {
    float side1;
    float side2;
}
(gdb) ptype EnteredShape.triangle
type = struct {
    float base;
    float height;
}
```

```
switch(MyShape)
{
  case circle :
    printf("Enter the radius of the circle ");
    scanf("%f", &EnteredShape.circle.radius);
    printf("The area of your circle is %f\n",
            M_PI * pow(EnteredShape.circle.radius, 2));
    break;
```

M_PI is defined in math.h

```
# define M_PI           3.14159265358979323846  /* pi */
```

```
  case square :
    printf("Enter the length of one side ");
    scanf("%f", &EnteredShape.square.side);
    printf("The area of your square is %f\n",
            pow(EnteredShape.square.side, 2));
    break;
```

```
   case rectangle :
      printf("Enter the length of side 1 ");
      scanf("%f", &EnteredShape.rectangle.side1);
      printf("Enter the length of side 2 ");
      scanf("%f", &EnteredShape.rectangle.side2);
      printf("The area of your rectangle is %f\n",
            EnteredShape.rectangle.side1 * EnteredShape.rectangle.side2);
      break;

   case triangle :
      printf("Enter the length of the base ");
      scanf("%f", &EnteredShape.triangle.base);
      printf("Enter the length of the height ");
      scanf("%f", &EnteredShape.triangle.height);
      printf("The area of your triangle is %f\n",
            (EnteredShape.triangle.base * EnteredShape.triangle.height) / 2);

      break;

   default :
      printf("You are out of shape\n");
}
```

```
Enter the radius of the circle 1
The area of your circle is 3.141593


Enter the length of one side 2
The area of your square is 4.000000


Enter the length of side 1 4
Enter the length of side 2 5
The area of your rectangle is 20.000000


Enter the length of the base 2
Enter the length of the height 5
The area of your triangle is 5.000000
```

typedef3Demo.c

Issue : When to use array notation `[]` and when to use pointer notation `->`.

```
typedef struct
{
    char name[20];
    char sandwich[20];
    char fry_size;
    char drink_size;
    char drink_type[20];
}
Combos;

Combos LunchOrders[1000] = {};
```

How to call a function named `PrintLunchOrders` and pass the entire array `LunchOrders` and the number of items in the array?

```
PrintLunchOrders(LunchOrders, numberOfOrders);
```

What is the first line of the function definition?

```
void PrintLunchOrders(Combos LunchOrders[], int numberOfOrders)
```

```
void PrintLunchOrders(Combos LunchOrders[], int numberOfOrders)
{
      int i;

      printf("\t%35s\t%s\t%s\n", "Fry", "Drink", "Drink");
      printf("\t%-15s%-15s\t%s\t%s\t%s\n",
            "Name", "Sandwich", "Size", "Size", "Type");

      for (i = 0; i < numberOfOrders; i++)
      {
            printf("%d.\t%-15s%-15s\t%c\t%c\t%s\n",
                  i+1, LunchOrders[i].name,
                  LunchOrders[i].sandwich,
                  LunchOrders[i].fry_size,
                  LunchOrders[i].drink_size,
                  LunchOrders[i].drink_type);
      }
}
```

Pass a single order/element from the array `LunchOrders` to a function called `SuperSizeIt()`.

```
SuperSizeIt(&LunchOrders[WhichCombo-1]);
```

So how do we call this function?

```
void SuperSizeIt(Combos *Order)
{
    Order->fry_size = 'L';
    Order->drink_size = 'L';
}
```

Issue : When to use array notation [] and when to use pointer notation ->.

```
LunchOrders[x]->name
```

This is using both array notation [] and pointer notation ->

If you pass in an array

```
PrintLunchOrders(LunchOrders, numberOfOrdersInFile);

void PrintLunchOrders(Combos LunchOrders[], int numberOfOrdersInFile)
```

then use array notation [] to access the structure members

```
printf("%d.\t%-15s%-15s\t%c\t%c\t%s\n",
        i+1, LunchOrders[i].name, LunchOrders[i].sandwich,
             LunchOrders[i].fry_size,
        LunchOrders[i].drink_size, LunchOrders[i].drink_type);
```

Issue : When to use array notation `[]` and when to use pointer notation `->`.

```
LunchOrders[x]->name
```

This is using both array notation `[]` and pointer notation `->`

If you pass in a pointer to an array element

```
SuperSizeIt(&LunchOrders[WhichCombo-1]);
```

```
void SuperSizeIt(Combos *Order)
```

then use pointer notation to access the structure members

```
Order->fry_size = 'L';
```

Issue : Using pointers when you should be using arrays

```
typedef struct
{
    char *lettuce;
    char *tomato;
    char *cheese;
}
TacoPtrStruct;
```

```
typedef struct
{
    char lettuce[20];
    char tomato[20];
    char cheese[20];
}
TacoArrayStruct;
```

```
int main(void)
{
  TacoArrayStruct TacoArray[10] = {};
  TacoPtrStruct TacoPtr[10] = {};

  strcpy(TacoArray[0].lettuce, "Iceberg");
  strcpy(TacoPtr[0].lettuce, "Iceberg");

  return 0;
}
```

```
(gdb) p TacoArray[0].lettuce
$3 = '\000' <repeats 19 times>

(gdb) p TacoPtr[0].lettuce
$4 = 0x0
```

```
typedef struct
{
   char *lettuce;
   char *tomato;
   char *cheese;
}
TacoPtrStruct;
```

```
typedef struct
{
   char lettuce[20];
   char tomato[20];
   char cheese[20];
}
TacoArrayStruct;
```

```
int main(void)
{
   TacoArrayStruct TacoArray[10] = {};
   TacoPtrStruct TacoPtr[10] = {};

   strcpy(TacoArray[0].lettuce, "Iceberg");
   strcpy(TacoPtr[0].lettuce, "Iceberg");

   return 0;
}
```

```
(gdb) p sizeof(TacoArray)
$6 = 600
(gdb) p sizeof(TacoPtr)
$7 = 240
(gdb) p sizeof(TacoArray[0])
$8 = 60
(gdb) p sizeof(TacoPtr[0])
$9 = 24
```

```
typedef struct
{
   char *lettuce;
   char *tomato;
   char *cheese;
}
TacoPtrStruct;
```

```
typedef struct
{
   char lettuce[20];
   char tomato[20];
   char cheese[20];
}
TacoArrayStruct;
```

```
int main(void)
{
   TacoArrayStruct TacoArray[10] = {};
   TacoPtrStruct TacoPtr[10] = {};

   strcpy(TacoArray[0].lettuce, "Iceberg");
   strcpy(TacoPtr[0].lettuce, "Iceberg");

   return 0;
}
```

```
23              strcpy(TacoArray[0].lettuce, "Iceberg");
(gdb) p TacoArray[0].lettuce
$5 = "Iceberg", '\000' <repeats 12 times>

24              strcpy(TacoPtr[0].lettuce, "Iceberg");
(gdb) step

Program received signal SIGSEGV, Segmentation fault.
0x00000000004004ea in main () at pvsaDemo.c:24
```

Command Line Parameters

Command line parameters are a sequence of strings used to pass information to a C program at execution time.

They appear as strings on the command line after the name of the program when it is executed.

These strings are separated by blanks, tabs, or other whitespace.

Command line parameters are available throughout the time that the program is executing.

Command Line Parameters

Command line parameters are accessed via arguments to `main()`

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

`argc` and `argv` are traditional names but can be anything

`argc` contains the count of parameters on the command line. The name of the program is the first command line parameter and it is part of the count so `argc` is always at least one.

Command Line Parameters

`argv` is an array of pointers to `chars`

the pointers point to the strings that appear on the command line

the array is indexed by `0` to `argc - 1` and terminated with a `NULL` pointer

Command Line Parameters

Running a program with command line parameters

```
a.out clp1 clp2 clp3
```

Running a program in debug with command line parameters

```
gdb --args a.out clp1 clp2 clp3
```

argcargv1Demo.c

```
int main(int argc, char *argv[])
{
   int i;

   printf("argc = %d\n", argc);

   for (i = 0; i < argc; i++)
   {
      printf("argv %d - %s\n", i, argv[i]);
   }

   return 0;
}
```

```
a.out clp1 clp2

argc = 3
argv 0 - a.out
argv 1 - clp1
argv 2 - clp2
```

```
a.out Monday,Tuesday
argc = 2
argv 0 - a.out
argv 1 - Monday,Tuesday
```

```
a.out
argc = 1
argv 0 - a.out
```

```
a.out frog TOAD elePhant candy
argc = 5
argv 0 - a.out
argv 1 - frog
argv 2 - TOAD
argv 3 - elePhant
argv 4 - candy
```

```
a.out What day is today?
argc = 5
argv 0 - a.out
argv 1 - What
argv 2 - day
argv 3 - is
argv 4 - today?
```

```
a.out trick-or-treat
argc = 2
argv 0 - a.out
argv 1 - trick-or-treat
```

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

int main(int argc, char *argv[])
{
   char filename1[20] = {};
   char filename2[20] = {};

   if (argc == 3)
   {
      strcpy(filename1, argv[1]);
      strcpy(filename2, argv[2]);

      printf("filename1 is %s and filename2 is %s\n",
      filename1, filename2);
   }
   else
   {
      printf("Need 2 command line parameters\n");
   }

   return 0;
}
```

```
[frenchdm@omega ~]$ argcargv4Demo.e
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger
filename1 is lion and filename2 is tiger
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger bear
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger-bear
filename1 is lion and filename2 is tiger-bear
[frenchdm@omega ~]$ argcargv4Demo.e lion, tiger-bear
filename1 is lion, and filename2 is tiger-bear
[frenchdm@omega ~]$ argcargv4Demo.e lion,tiger-bear
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion,tiger bear
filename1 is lion,tiger and filename2 is bear
```

```
gdb --args argcargv4Demo.e inputfile outputfile
…
Reading symbols from /home/f/fr/frenchdm/argcargv4Demo.e...done.
(gdb) break main
Breakpoint 1 at 0x400537: file argcargv4Demo.c, line 8.
(gdb) run
Starting program: /home/f/fr/frenchdm/argcargv4Demo.e inputfile outputfile
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x2aaaaaaab000
Breakpoint 1, main (argc=3, argv=0x7fffffffe7e8) at argcargv4Demo.c:8
8               char filename1[20] = {};
(gdb) p argv
$1 = (char **) 0x7fffffffe7e8
(gdb) p *argv@argc
$2 = {0x7fffffffea2a "/home/f/fr/frenchdm/argcargv4Demo.e",
  0x7fffffffea4e "inputfile", 0x7fffffffea58 "outputfile"}
(gdb) p argc
$3 = 3
(gdb) p argv[0]
$4 = 0x7fffffffea2a "/home/f/fr/frenchdm/argcargv4Demo.e"
(gdb) p argv[1]
$5 = 0x7fffffffea4e "inputfile"
(gdb) p argv[2]
$6 = 0x7fffffffea58 "outputfile"
```

Advanced Command Line Parameter Handling

What if a program has 10 command line parameters?

What if it has 20?

As the number of command line parameters increases, the harder and harder it gets to keep them in the right order on the command line.

It would be more convenient to not have to list the command line arguments in the correct order.

Advanced Command Line Parameter Handling

We can list the parameters in any order we want if we name them.

This is one method of doing that…

```
Code7_1000074079.e INFILE=input.txt OUTFILE=output.txt MODE=r
```

Each command line parameter is named and that allows them to be listed in any order and our program will work the same.

```
Code7_1000074079.e MODE=r OUTFILE=output.txt INFILE=input.txt
```

Advanced Command Line Parameter Handling

```
Code7_1000074079.e P1=input.txt P2=output.txt P3=r P4=cat P5=dog

Code7_1000074079.e P5=dog P1=input.txt P3=r P2=output.txt P4=cat
```

We can code our program such that the order of the parameters does not matter.

```
a.out Date=04202020 DOW=Monday Time=13:23 Department=CSE Course=1320
a.out DOW=Monday Date=04202020 Course=1320 Time=13:23 Department=CSE
a.out Time=13:23 Course=1320 Department=CSE Date=04202020 DOW=Monday
a.out Course=1320 Date=04202020 DOW=Monday Department=CSE Time=13:23
```

```
int main(int argc, char *argv[])
{
   char filename[100] = {};
   char mode[3] = {};

   if (argc < 3)
   {
      printf("\n\nNeed 2 command line parameters");
      exit(0);
   }

   strcpy(filename, argv[1]);

   strcpy(mode, argv[2]);

   printf("You want to use file %s with mode %s\n", filename, mode);

   return 0;
}
```

```
a.out input.txt r

a.out FILENAME=input.txt MODE=r
a.out MODE=r FILENAME=input.txt
```

```
int main(int argc, char *argv[])
{
   char filename[100] = {};
   char mode[3] = {};

   if (argc < 3)
   {
      printf("\n\nParameters : FILENAME=xxxx MODE=yy\n\nExiting....\n\n");
      exit(0);
   }

   get_command_line_parameter(argv, "FILENAME=", filename);

   get_command_line_parameter(argv, "MODE=", mode);

   printf("You want to use file %s with mode %s\n", filename, mode);

   return 0;
}
```

```
a.out Date=04202020 DOW=Monday Time=13:23 Department=CSE Course=1320
a.out DOW=Monday Date=04202020 Course=1320 Time=13:23 Department=CSE
a.out Time=13:23 Course=1320 Department=CSE Date=04202020 DOW=Monday
a.out Course=1320 Date=04202020 DOW=Monday Department=CSE Time=13:23
```

```
char date[9];
char dayofweek[10];
char time[6];
char dept[4];
char coursename[5];
```

```
get_command_line_parameter(argv, "Date=", date);
get_command_line_parameter(argv, "DOW=", dayofweek);
get_command_line_parameter(argv, "Time=", classtime);
get_command_line_parameter(argv, "Department=", dept);
get_command_line_parameter(argv, "Course=", coursename);
```

```
a.out FILENAME=input.txt MODE=r
argv[0] a.out
argv[1] FILENAME=input.txt
argv[2] MODE=r
```

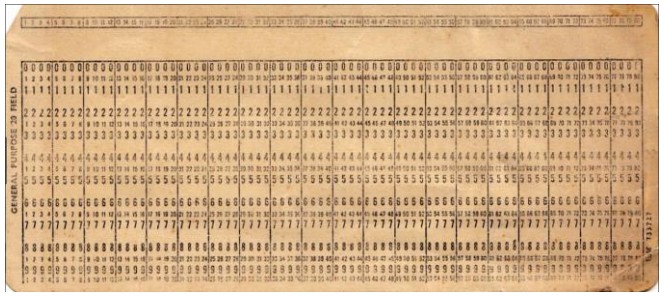
```
a.out MODE=r FILENAME=input.txt
argv[0] a.out
argv[1] MODE=r
argv[2] FILENAME=input.txt
```

```
void get_command_line_parameter(char *argv[], char PName[], char PValue[])
{
   int i = 1;

   while (argv[i] != NULL)
   {
      if (!strncmp(argv[i], PName, strlen(PName)))
      {
         strcpy(PValue, strchr(argv[i], '=') + 1);
      }
      i++;
   }
}
```

```
get_command_line_parameter(argv, "FILENAME=", filename);

get_command_line_parameter(argv, "MODE=", mode);
```



File Handling in C

Storage of data in memory is temporary.

Files are used for data persistence – permanent retention of data.

Computers store files on secondary storage devices such as hard disks, CDs, DVDs, flash drives and tapes.

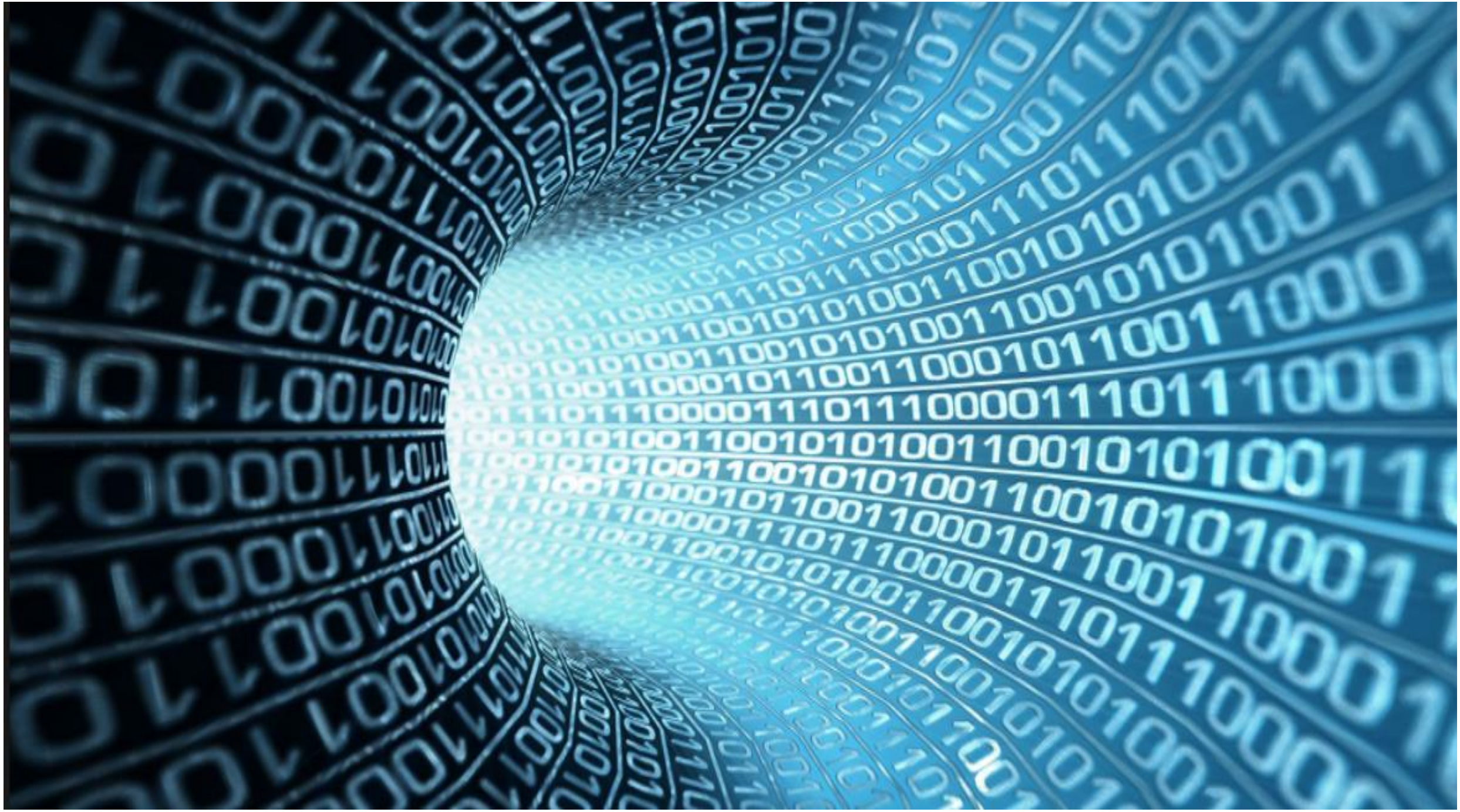
File Handling in C

C does not impose structure on a file. A file is just a sequence of data.

The concept of a record in a file does not exist.

The application using the file imposes the structure/record on the file.

For example, a Word document is just a sequence of data that Word knows how to interpret and display and manipulate.



File Handling in C

C views each file as simply a sequence of bytes.

Each file ends either with an end-of-file marker (EOF) or at a specific byte number recorded in an operating-system-maintained administrative database.

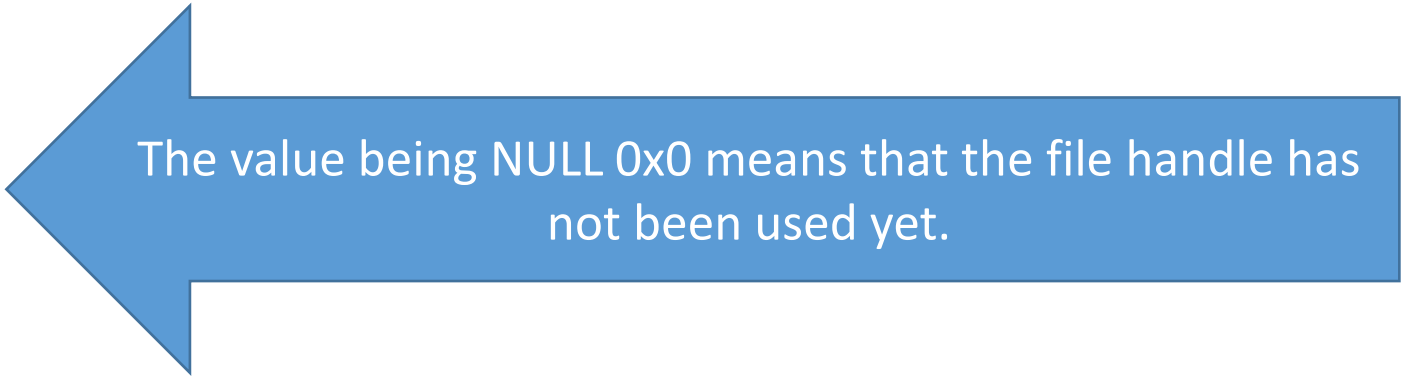
When a file is opened, a file handle is associated with that file and is used by the C program to refer to the file.

File Handling in C

When a file is to be opened for use in a program, the programmer must declare a new variable of type `FILE *`.

```
FILE *myfile, *yourfile;
```

```
(gdb) p myFile
$1 = (FILE *) 0x0
```



Definition of `FILE *`

```
(gdb) ptype myFile
type = struct _IO_FILE {
    int _flags;
    char *_IO_read_ptr;
    char *_IO_read_end;
    char *_IO_read_base;
    char *_IO_write_base;
    char *_IO_write_ptr;
    char *_IO_write_end;
    char *_IO_buf_base;
    char *_IO_buf_end;
    char *_IO_save_base;
    char *_IO_backup_base;
    char *_IO_save_end;
    struct _IO_marker *_markers;
    struct _IO_FILE *_chain;
    int _fileno;
    int _flags2;
    __off_t _old_offset;
    short unsigned int _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];
    _IO_lock_t *_lock;
    __off64_t _offset;
    void *__pad1;
    void *__pad2;
    void *__pad3;
    void *__pad4;
    size_t __pad5;
    int _mode;
    char _unused2[20];
} *
```

File Handling in C

C library function, `fopen()`, is then used to connect these declarations with the files on disk. Using `fopen()` makes a file available for use by the program.

```
myfile = fopen("filename", "mode");
```

```
myFile = fopen("it.txt", "r");
```

We will refer to `myfile` as the file's handle or file handle.

```
(gdb) p myFile
$2 = (FILE *) 0x601010
```

File handle has been used.

After opening a file, it should be closed as soon as possible to prevent conflicts with other processes.

```
fclose(FileHandle);
```

```
fclose(myfile);
```

```
fclose(myFile);
```

```
10                  myFile = fopen("it.txt", "r");
(gdb) shell ls -l /proc/7661/fd

total 0
lrwx------ 1 frenchdm staff 64 Nov  3 16:00 0 -> /dev/pts/31
lrwx------ 1 frenchdm staff 64 Nov  3 16:00 1 -> /dev/pts/31
l-wx------ 1 frenchdm staff 64 Nov  3 16:00 6 -> pipe:[124469164]

(gdb) shell ls -l /proc/7661/fd
total 0
lrwx------ 1 frenchdm staff 64 Nov  3 16:00 0 -> /dev/pts/31
lrwx------ 1 frenchdm staff 64 Nov  3 16:00 1 -> /dev/pts/31
l-wx------ 1 frenchdm staff 64 Nov  3 16:00 6 -> pipe:[124469164]
lr-x------ 1 frenchdm staff 64 Nov  3 16:00 7 -> /home/f/fr/frenchdm/it.txt
```

69 fclose(MyFile);

```
(gdb) shell ls -l /proc/26669/fd
total 0
lrwx------ 1 frenchdm staff 64 Nov  3 19:39 0 -> /dev/pts/32
lrwx------ 1 frenchdm staff 64 Nov  3 19:39 1 -> /dev/pts/32
l-wx------ 1 frenchdm staff 64 Nov  3 19:39 6 -> pipe:[124773435]
lrwx------ 1 frenchdm staff 64 Nov  3 19:39 7 -> /home/f/fr/frenchdm/it.txt
```

```
(gdb) shell ls -l /proc/26669/fd

total 0
lrwx------ 1 frenchdm staff 64 Nov  3 19:39 0 -> /dev/pts/32
lrwx------ 1 frenchdm staff 64 Nov  3 19:39 1 -> /dev/pts/32
l-wx------ 1 frenchdm staff 64 Nov  3 19:39 6 -> pipe:[124773435]
```

File Handling in C

```
myfilehandle = fopen("filename", "mode");
```

mode of a file determines whether the file is opened for reading, writing or a combination of the two

`"r"`	open an existing file for reading only
`"w"`	open new file for writing only
`"a"`	open a file for appending (writing at the end of the file)
`"r+"`	open an existing file for update (reading and writing)
`"w+"`	create a new file for update (reading and writing)
`"a+"`	open a (new or existing) file for reading and appending

```
int main(int argc, char *argv[])
{
  char readfilename[100] = {};
  char writefilename[100] = {};
  char rwfilename[100] = {};
  char buffer[100] = {};
  FILE *ReadFH, *WriteFH, *ReadWriteFH;

  if (argc > 3)
  {
    strcpy(readfilename, argv[1]);
    strcpy(writefilename, argv[2]);
    strcpy(rwfilename, argv[3]);
  }

  ReadFH = fopen(readfilename, "r");
  WriteFH = fopen(writefilename, "w");
  ReadWriteFH = fopen(rwfilename, "w+");
```

File Handles

get command line parameters

r – open file for read
w – open file for write
w+ - open file for update

```
ReadFH = fopen(readfilename, "r");
WriteFH = fopen(writefilename, "w");
ReadWriteFH = fopen(rwfilename, "w+");

if (ReadFH == NULL || WriteFH == NULL || ReadWriteFH == NULL)
{
     printf("A file did not properly open...exiting\n");
     exit(0);
}
else
     printf("All files opened\n");
```

```
fclose(ReadFH);
fclose(WriteFH);
fclose(ReadWriteFH);
```

If a file handle's value is `NULL` after using `fopen()`, then something is wrong and the file did not open. In this case, the program does not continue to run and `exit()` is used to leave program immediately. Depending on the platform where the program is being run, the value passed with `exit()` may trigger an event.

Formatted Input and Output

`printf()` and `scanf()`

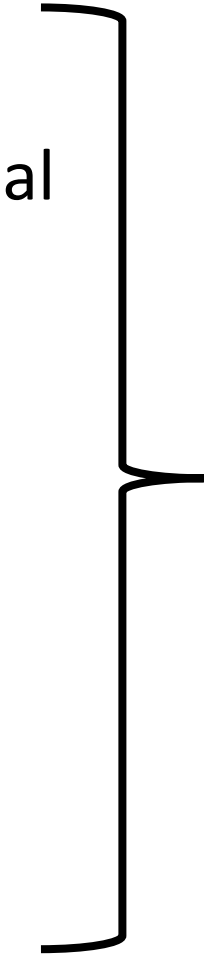
do formatted input and output to and from the terminal

`fprintf()` and `fscanf()`

do formatted input and output to and from a file

`sprintf()` and `sscanf()`

write and read to and from a string



`printf()` and `scanf()` families

Error Detection with the `printf()` and `scanf()` Families

The `printf()` family will return an `int` value indicating the total number of characters written by each particular call.

The `scanf()` family will return an `int` indicating the number of conversions that were made which should match the conversion specifications in its control string.

Depending on the criticality of your application, adding this level of error checking may not be worth the added complexity.

Formatted Input and Output

`fscanf()` and `fprintf()`

```
fscanf(fp, control_string, args, …)

fprintf(fp, control_string, args, …)
```

`fp`	file handle (`FILE *`) - associated with an open file
`control_string`	conversion specifier
`args`	argument to conversion specifier

```
[frenchdm@omega ~]$ ls WFILE
ls: WFILE: No such file or directory

[frenchdm@omega ~]$ ls RWFILE
ls: RWFILE: No such file or directory

[frenchdm@omega ~]$ more RFILE
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu
```

```
[frenchdm@omega ~]$ fprint1Demo.e RFILE WFILE RWFILE
All files opened
Enter a string Hello there
[frenchdm@omega ~]$ more WFILE
Hello there
[frenchdm@omega ~]$ more RFILE
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu
[frenchdm@omega ~]$ more RWFILE
Hello there
```

Files opened with "w" and "w+" did not exist but were created when opened.

File opened with "r" was written to but was not updated.

```
printf("Enter a string ");

fgets(buffer, sizeof(buffer)-1, stdin);

fprintf(WriteFH, "%s", buffer);

fprintf(ReadFH, "%s", buffer);

fprintf(ReadWriteFH, "%s", buffer);
```

fprintf1Demo.c

```
[frenchdm@omega ~]$ fprint1Demo.e RFILE WFILE RWFILE
All files opened
Enter a string Good bye
[frenchdm@omega ~]$ more WFILE
Good bye
[frenchdm@omega ~]$ more RFILE
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu
[frenchdm@omega ~]$ more RWFILE
Good bye
```

Files opened with "w" and "w+" discarded the contents of the existing files and then wrote to them.

```
printf("Enter a string ");

fgets(buffer, sizeof(buffer)-1, stdin);

fprintf(WriteFH, "%s", buffer);

fprintf(ReadFH, "%s", buffer);

fprintf(ReadWriteFH, "%s", buffer);
```

fprintf1Demo.c

```
[frenchdm@omega ~]$ more RFILE
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu

[frenchdm@omega ~]$ fprintf1Demo.e RFILE WFILE RWFILE
All file
alpha
[frenche
----alpha____

[frenchdm@omega ~]$ more RWFILE
----alpha____
```

```
fscanf(ReadFH, "%s", buffer);
printf("%s", buffer);
fprintf(WriteFH, "----%s____", buffer);
fprintf(ReadWriteFH, "----%s____", buffer);
```

`fscanf()` stops reading at the whitespace
just like cousin `scanf()`

fprintfDemo.c

To read from the physical file associated with a given file handle

```
fscanf(ReadFH, "%s", buffer);
```

To write to the physical file associated with a given file handle

```
fprintf(WriteFH, "%s", buffer);
```

So what does this do?

```
fscanf(stdin, "%s", buffer);

fprintf(stdout, "%s", buffer);
```

`stdin` is the file handle for standard input (your keyboard) and `stdout` is the file handle for standard out (your screen)

```
frenchdm@omega:~
[frenchdm@omega ~]$ fprintf1Demo.e RFILE WFILE RWFILE
```

fprintfDemo.c

`fgets()` and reading files

We have been using `fgets()` like this

```
char Line[20] = {};
fgets(Line, sizeof(Line)-1, stdin);
```

We gave `stdin` as the third parameter because we wanted to read from `stdin` (our keyboard).

We used `fgets()` to read input instead of `scanf()` when that input contained whitespace since `scanf()` stops at whitespace.

`fgets()` and reading files

So what if we used `fgets()` instead of `fscanf()` when we need to read from a file where the lines in that file contain whitespace?

`fscanf()` only read `alpha` from our file because it stopped reading at the first whitespace it encountered - just like cousin `scanf()`.

```
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu
```

`fgets()` and reading files

```
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu

fgets(buffer, sizeof(buffer)-1, ReadFH);
printf("%s", buffer);

alpha bravo charlie delta
```

fgets() and reading files

```
alpha bravo charlie delta
echo foxtrot golf hotel
india juliett kilo lima
mike november oscar papa
quebec romeo sierra tango
uniform victor whiskey xray
yankee zulu
```

So why did `fgets()` only read the first line of the file and not the whole file?

When reading from `stdin` (keyboard), when does `fgets()` know to stop reading?

When you press <ENTER> which translates to `\n`

What is at the end of each line of this file? How was the file created?

```
fgets(buffer, sizeof(buffer)-1, ReadFH);
printf("%s", buffer);
```

```
alpha bravo charlie delta
```

File Handling in C

Part of the structure defined in the `typedef FILE` is a value that tracks the current position in the file.

We will refer to that as the ***file pointer***.

The file pointer moves as reads and writes are done.

File Handling in C
Two Types of Access

Sequential Access

When a file is opened, reading (or writing) starts at the beginning of the file and proceeds through the file in a sequential manner.

Whenever a read is done, the file pointer moves to point to the next element in the file to be read.

Random Access

Allows the reading of the records in any order.

Random Access in Files

Two library functions in the standard C library help with random access of files

```
fseek(fp, offset, start);
```

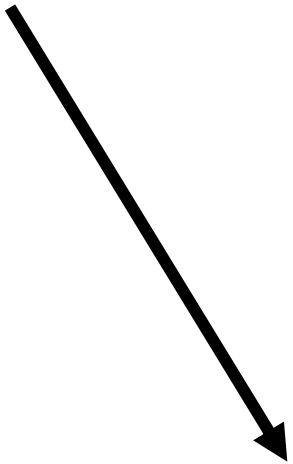
`fp`	file handle (`FILE *`) - associated with an open file
`offset`	variable of type long that represents the byte offset or number of bytes that the pointer is to be moved
`start`	indicates the beginning position for the file pointer must be 0, 1 or 2

```
ftell(fp);
```

`fp`	file handle (`FILE *`) – associated with an open file returns the current byte offset from the beginning of the file

```
for (i = 0; i < 5; i++)
{
   printf("Enter string %d ", i);
   fgets(buffer, sizeof(buffer), stdin);
   fprintf(MyFile, "%s", buffer);
}
```

```
Enter string 0 Hello
Enter string 1 there.
Enter string 2 How
Enter string 3 are
Enter string 4 you?
```



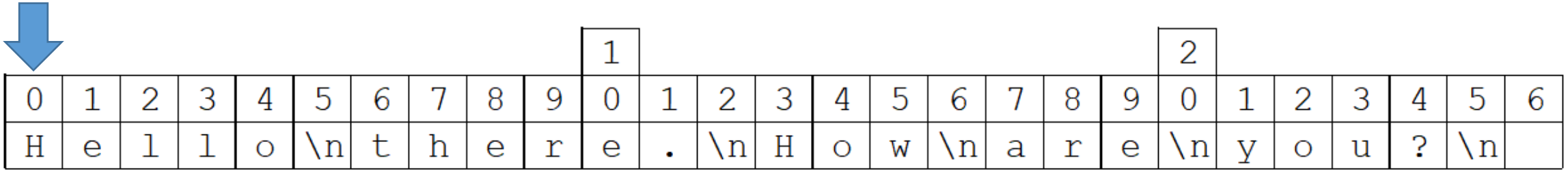
										1										2					
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
H	e	l	l	o	\n	t	h	e	r	e	.	\n	H	o	w	\n	a	r	e	\n	y	o	u	?	\n

										1										2						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
H	e	l	l	o	\n	t	h	e	r	e	.	\n	H	o	w	\n	a	r	e	\n	y	o	u	?	\n	

The file pointer is sitting at position 26. `ftell()` can return the file pointer's location.

```
Breakpoint 4, main (argc=4, argv=0x7fffffffe868) at fseekDemo.c:62
62                  printf("ftell() = %d\n\n", ftell(MyFile));
(gdb) step

ftell() = 26
```



										1										2						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
H	e	l	l	o	\n	t	h	e	r	e	.	\n	H	o	w	\n	a	r	e	\n	y	o	u	?	\n	

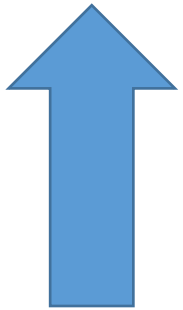
```
fseek(MyFile, 0, 0);

for (i = 0; i < 5; i++)
{
   printf("ftell() = %d\t", ftell(MyFile));
   fscanf(MyFile, "%s", &buffer);
   printf("Printing string %d from file : %s\t", i, buffer);
   printf("ftell() = %d\n", ftell(MyFile));
}
```

fscanf() stops at \n

```
ftell() = 0     Printing string 0 from file : Hello     ftell() = 5
ftell() = 5     Printing string 1 from file : there.    ftell() = 12
ftell() = 12    Printing string 2 from file : How       ftell() = 16
ftell() = 16    Printing string 3 from file : are       ftell() = 20
ftell() = 20    Printing string 4 from file : you?      ftell() = 25
```

										1										2						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
H	e	l	l	o	\n	t	h	e	r	e	.	\n	H	o	w	\n	a	r	e	\n	y	o	u	?	\n	



```
75                  printf("Enter an offset of fseek() ");
(gdb)
76                  scanf("%ld", &offset);
(gdb)
```

Enter an offset of fseek() 21


```
80                          fseek(MyFile, offset, SEEK_SET);
(gdb) step
81                          fscanf(MyFile, "%s", &buffer);
(gdb)
82                          printf("Printing string from file : %s\n\n", buffer);
```

Printing string from file : you?

										1										2						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
H	e	l	l	o	\n	t	h	e	r	e	.	\n	H	o	w	\n	a	r	e	\n	y	o	u	?	\n	

```
84                          printf("Enter an offset of fseek() ");
(gdb) step
85                          scanf("%ld", &offset);
```

Enter an offset of fseek() 14


```
80                          fseek(MyFile, offset, SEEK_SET);
(gdb)
81                          fscanf(MyFile, "%s", &buffer);
(gdb)
82                          printf("Printing string from file : %s\n\n", buffer);
(gdb)
```

Printing string from file : ow

Random Access in Files

Defines from `stdio.h` that can be used with file access

```
#define SEEK_SET        0       /* Seek from beginning of file.  */

#define SEEK_CUR        1       /* Seek from current position.  */

#define SEEK_END        2       /* Seek from end of file.  */
```

Formatted Input and Output

`sscanf()` and `sprintf()`

```
sscanf(buffer, control_string, args, …)

sprintf(buffer, control_string, args, …)
```

`buffer`	buffer in memory
`control_string`	conversion specifier
`args`	argument to conversion specifier

Formatted Input and Output

`sscanf()` **and** `sprintf()`

```
sprintf(buffer, "%s %s has student id %s ",
        first_name, last_name, id);

sscanf(buffer, "%s %s %*s %*s %*s %s", a, b, c);
```

```
char buffer[100] = {};
char first_name[50] = {};
char last_name[50] = {};
char id[10] = {};
char a[50] = {};
char b[50] = {};
char c[10] = {};

printf("Enter first name ");
scanf("%s", &first_name);
printf("\nEnter last name ");
scanf("%s", &last_name);
printf("\nEnter id ");
scanf("%s", &id);
```

```
Enter first name Fred

Enter last name Flintstone

Enter id 1000000001
```

sprintfDemo.c

```
Breakpoint 2, main () at sprintfDemo.c:23
23                sprintf(buffer, "%s %s has student id %s ",
24                        first_name, last_name, id);
(gdb) p first_name
$1 = "Fred", '\000' <repeats 36 times>"\377, \265\360\000\000\000\000\000",
<incomplete sequence \302>
(gdb) p last_name
$2 =
"Flintstone\000\000\000\000\000\000\000\347\377\377\001\000\000\000\340\366\252\252\25
2*\000\000`藪\252*\000\000\020\350\377\377\377\177\000\000\000", <incomplete sequence
\340>
(gdb) p id
$3 = "1000000001"
(gdb) step


(gdb) p buffer
$4 = "Fred Flintstone has student id 1000000001
\000\000\000\000\000\000\001\000\000\000\000\000\303\000\307\006@", '\000' <repeats 13
times>"\300, \313!\311>\000\000\000\220\006@", '\000' <repeats 13 times>"\220,
\350\377\377"
```

sprintfDemo.c

`%*s` tells `sscanf()` to skip the characters between whitespaces.

```
Fred Flintstone has student id 1000000001

27              sscanf(buffer, "%s %s %*s %*s %*s %s", a, b, c);
(gdb) p a
$5 = "\000\000\000\000\000\000\000\000\002\223\000\311>", '\000' <repeats 11
times>"\340, \366\252\252\252*\000\000\001", '\000' <repeats 15 times>, "\001"
(gdb) p b
$6 = "\000\000\000\000\001\000\000\000\227\a\000\000\001", '\000' <repeats 11 times>,
"`藪\252*\000\000\340\347\377\377\377\177\000\000\220\347\377\377\377\177\000\000.N"
(gdb) p c
$7 = "@\374@\311>\000\000\000\250\002"
(gdb) step
28              printf("First name = %s\nLast Name = %s\nID = %s\n\n", a, b, c);
(gdb) p a
$8 = "Fred\000\000\000\000\002\223\000\311>", '\000' <repeats 11 times>"\340,
\366\252\252\252*\000\000\001", '\000' <repeats 15 times>, "\001"
(gdb) p b
$9 = "Flintstone\000\000\001", '\000' <repeats 11 times>, "`藪
\252*\000\000\340\347\377\377\377\177\000\000\220\347\377\377\377\177\000\000.N"
(gdb) p c
$10 = "1000000001"
```

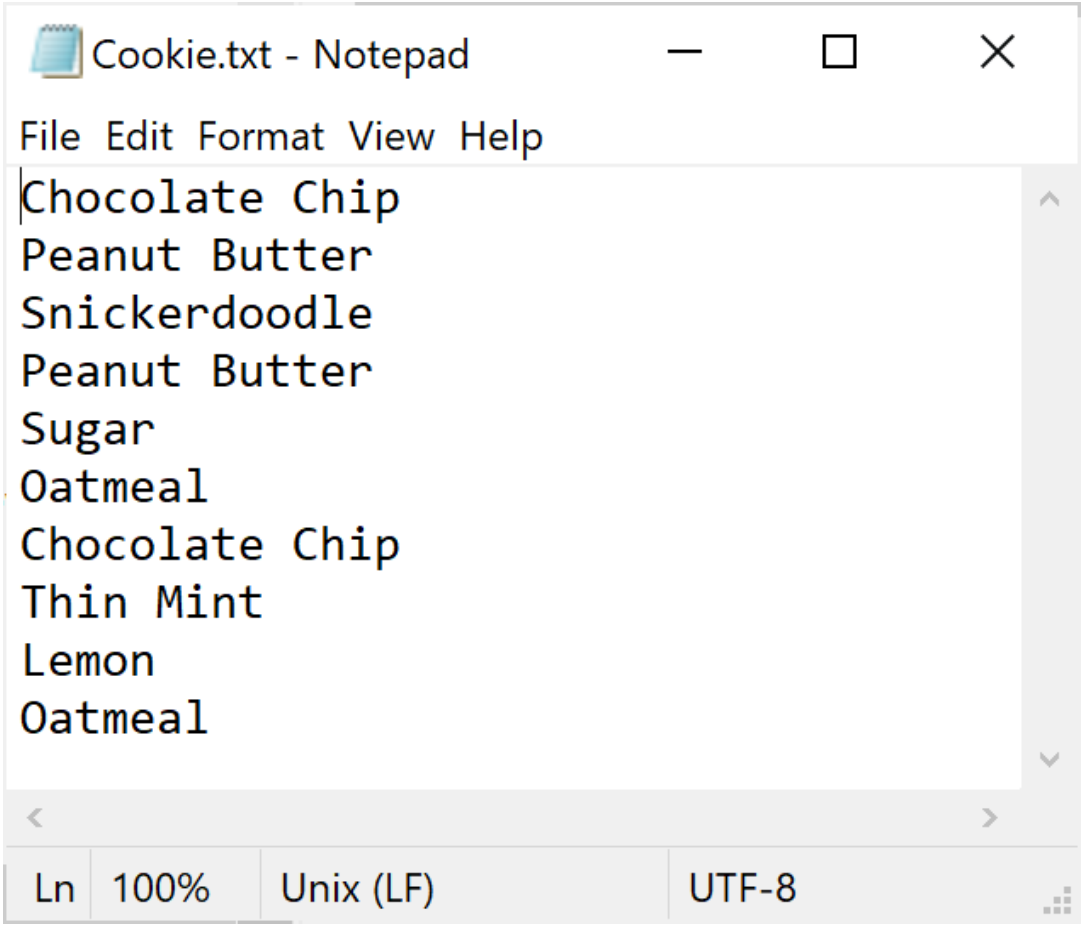
sprintfDemo.c

In Class Exercise

Create a complete C program that prompts for a file name and opens that file for appending.

Read the file and print each line in the file to the screen.

After reading all lines, print “File Processed” to the end of file.



```
FILE *FH;
File *FH;
FILE FH;

char FileLine[] = {};
char FileLine[100] = {};

char FileName[] = {};
char FileName[100] = {};

printf("Enter your filename ")
printf("Enter your filename ");

scanf("%s", &FileName);
scanf("%s", FileName);

FH = fopen("a+", FileName);
FH = fopen(FileName, "a+");
FH = fopen(FileName, 'a+');

if (FH = NULL)
if (FH == NULL)
```

```
printf("File did not open...bye!")
printf("File did not open...bye!");

exit(0);
exit;

while (fgets(FH, FileLine, sizeof(FileLine)-1))
while (fgets(FileLine, sizeof(FileLine)-1, FH))
while (fgets(FileLine, sizeof(FileLine)-1, FileName))

fprintf("%s", FileLine);
printf("%s", FileLine);

fprintf("\n%s", "File Processed", FH);
fprintf(FH, "\n%s", "File Processed");
fprintf(FileName, "\n%s", "File Processed");
printf(FH, "\n%s", "File Processed");

close(FH);
fclose(FH);
fclose(FileName);
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   FILE *FH;

   char FileLine[100] = {};
   char FileName[100] = {};

   printf("Enter your filename ");
   scanf("%s", FileName);

   FH = fopen(FileName, "a+");

   if (FH == NULL)
   {
      printf("File did not open...bye!");
      exit(0);
   }
   while (fgets(FileLine, sizeof(FileLine)-1, FH))
   {
      printf("%s", FileLine);
   }

   fprintf(FH, "\n%s", "File Processed");

   fclose(FH);
}
```