1. Input and Output Streams
   1. **Streams**: sequence of characters (0’s and 1’s) organized into lines.
      1. Streams can be redirected to other devices.
      2. 3 standard streams
         1. **Input stream**
            1. Standard: keyboard.
         2. **Output stream**
            1. Standard: screen.
         3. **Error stream**
   2. **Lines**: must be 254 characters, including the end-of-line character.
2. Output Functions in C
   1. Advanced features in printf()
      1. Generalized way of printing a number
         1. %[flag][width][.precision]specifier
            1. Everything in the brackets is optional
      2. Parameters
         1. **Specifier**
            1. Integers

%d = int

%i = signed decimal int

%u = unsigned decimal int

%o = unsigned octal int

%x or %X = unsigned hexadecimal (a-f or A-F respectively)

%h = “short”

%l = “long”

* + - * 1. Floats

%f = float

%lf = double-precision float

%e or %E = float with scientific notation

%g or %G = can do floats with or without scientific notation, depending on the size of the number

* + - * 1. Strings

%c = char

%s = strings

* + - 1. **Field width**: specifies the minimum number of characters to be printed.
         1. Does not truncate the result, even if the number is longer than the width.

Increases the size of the field

* + - * 1. If the number does not take up the full width, it will pad with blank spaces to right-justify it.
      1. **Precision**
         1. int: the number of digits to be printed.

Uses 0s if printed value is smaller than the precision.

* + - * 1. float: number of digits after the decimal point that should be shown.
        2. string: number of characters to be printed.
      1. **Flags**
         1. -: left-justify the number instead of right-justifying it.
         2. +: forces the sign of the number to be shown.
         3. 0: pads with zeroes instead of spaces.
         4. space: puts a space before a positive value.
         5. #: prefixes an O to a printed number if octal.
    1. Examples
       1. printf(“%.2lf”, 43.21);
          1. 43.21 instead of 43.21000
       2. printf(“%+05d”, 47);
          1. +0047 instead of 47

1. Input Functions in C
   1. scanf()
      1. Variable name of datum read by its address
         1. scanf(“%d%d%d”,&a,&b,&c);
      2. Formatting
         1. scanf(format-control-string, other arguments);
            1. format-control-string: defines format of inputs
            2. other arguments: indicate where to store them

Address-of-operator & used

Exception: strings

Pointers already

* + 1. Type specifiers
       1. %d = signed decimal int
       2. %i = signed decimal, octal, or hexadecimal int
       3. %o = octal int
       4. %u = unsigned decimal int
       5. %x or %X = hexadecimal int prefixed with x
    2. Assignment Suppression
       1. Example: reading a date in the format DD/MM/YYYY, but extract day, month, and year.
       2. **Assignment suppression character \***
       3. Example of use
          1. int month1, day1, year1;
          2. printf(“Enter date in the mm-dd-yyyy format: ”)
          3. scanf(“%d%\*c%d%\*c%d”, &month1,&day1,&year1);

1. String and Character Literals
   1. Escape sequences: begin with \
      1. \n = new line
      2. \t = tab
      3. \” = double quote
      4. \’ = single quote
      5. \\ = backslash
      6. \0 = null terminator (implicit at the end of every string)
      7. %% = percent sign
   2. Printing Long String Literals on Multiple Lines
      1. printf(“This is a really, really, “
      2. “really long string.\n”);
   3. String Output Functions
      1. puts(): prints a string to the standard output, followed by a new line.
         1. Returns int
      2. putchar(): prints out character argument.
         1. Must be placed within a loop for strings.
   4. String Input Functions
      1. gets(): reads characters from standard input stream and stores them in argument array.
         1. Stops with newline or eof (end-of-file).
         2. Null character automatically appended to end of string.
      2. getchar(): inputs next character from standard input and returns an int.
         1. Does not take an argument.
         2. Must be put in a loop for reading a string.
         3. Null character not automatically appended.
2. File Input/Output
   1. External File Storage
      1. Storing data in data structures is volatile
      2. Permanent storage requires data in external files
   2. File I/O Basics
      1. Working with files is similar to standard I/O
         1. fprintf() ~ printf()
         2. fscanf() ~ scanf()
      2. Steps
         1. Declare pointer of type FILE before file manipulation.
            1. Found in stdio.h
            2. Example

FILE \* fin, \* fout;

\* fin = “file in” to read

\* fout = “file out” to write

If more, \*fin1, etc.

* + - 1. Create a new file or open an existing file.
         1. fopen()

Parameters

File name (“string literal”)

Must be in same folder

Otherwise, need to specify

What you want to do with file (string)

“r” = read file

“w” = write file

Using “w” instead of “r” for an existing file will erase the contents of the file.

“a” = append file

“r+” = opens file for update (read/write)

“w+” = creates file for updating (read/write)

“a+” = append – open/create a file for updating

Returns…

An address (file pointer) associated with the file.

Assigned to \* fin

NULL if it fails to open.

**Must continually check for NULL**.

if ((fin=fopen(“results.dat”,”r”)) == NULL)

printf(“File could not be opened.”);

else

<whatever>

Example

FILE \* fin, \* fout;

fin = fopen(“test.txt”, “r”);

Opens test.txt

fout = fopen(“sample.txt”, “w”);

Writes to sample.txt

* + - 1. Read from a file
         1. fscanf(): reads the file with the file pointer as the first parameter.

Rest of parameters work like normal scanf()

Example

fscanf(fin,”%d”, &x);

* + - 1. Close the file
         1. fclose(): closes the file.

Allows other programs to use the file.

Most OS’s will close it if left open after program exits, but fclose() should be used anyway.

Prevents file corruption.

Takes one parameter

File pointer

Example

fclose(fout);

1. Sequential Data Files
   1. Updating records require rewriting the entire file.
   2. Fields in the file area of different size, as defined by what one writes to it.
      1. Example: if a string in an external file says “This” is replaced by “This is not good”, it will spill over to the next data item and overwrite part of it.
         1. Corrupts data file
2. **Randomly-Accessed Files**: permits easy updates.
   1. All fields of the same size
      1. Replacing one field with other information does not matter
   2. Involves “search” to find element to be replaced
3. End of File
   1. Important to know when you have reached the end.
      1. Not much of an issue if you know the correct data size.
      2. Generally-speaking, should not be trusted.
   2. feof(): file end-of-file function.
      1. Returns “true” value if you have reached the end of the file you passed in.
      2. Returns “false” otherwise