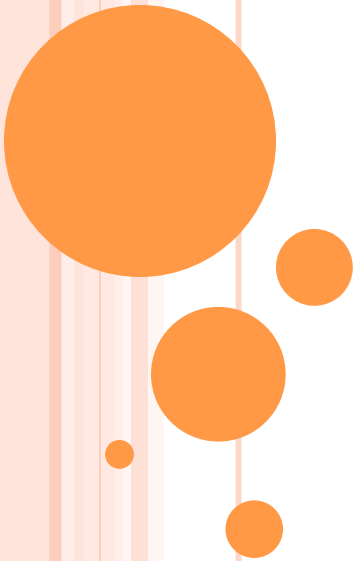


COMP 26120: Algorithms and Imperative Programming

Lecture 2:
Introduction to C programming
Input/Output

Lecture overview

- Introduction to C programming language;
- Study resources;
- Comparison between C and Java (example code);
- Input and output in C;

Introduction to C programming language

- COMP26120 contents:
  - Algorithms
  - C programming
- Course-unit approach:
  - Active learning
  - Reading

Introduction to C programming language

- New Programming Language
- Paradigm
- Concepts
- Useful for certain applications:
  - Uncluttered view of hardware ("high-level assembly-code");
  - Performance;
- A little history:
  - Originally from late 60's - published 1972
  - 1989/1990 ISO standard (C89, C90)
  - 1999/2000 ISO standard (C99)
  - 201? (C1X)

Study resources

- **Starting points**:
  - **Syllabus**: http://studentnet.cs.manchester.ac.uk/ugt/2016/COMP26120/syllabus/
- **Course-unit**:
  - http://studentnet.cs.manchester.ac.uk/ugt/2016/COMP26120/
- man pages etc.
- C programming forum on moodle
  - https://moodle.cs.man.ac.uk/course/view.php?id=28
- **Further reading**:
  - Online C course in Moodle
  - Online "Standard C" book (Plauger & Brodie)
  - The C Programming Language (Kernighan & Ritchie)
  - C: A Reference Manual (Harbison & Steele)
  - Expert C Programming (van der Linden)

Study resources
Online C course in MOODLE

- **URL:**

https://moodle.cs.man.ac.uk/file.php/28/coursedata/c_cbt/frontpage.html

- **Navigation:** begin / return / map / index
- **Themes:**
  - Java to C (briefly covered today through a code example)
  - Information Representation (Lecture 3/4)
  - Control Flow (Lecture 3)
  - Program Structuring (advanced reading – self study)
  - Input & Output (covered in some detail today)
- Colour-coding: green, yellow, orange, red (from easy to difficult);
- Dependencies/Routes (arrows);
- Use as a reference material for the labs;

Comparison between C and Java

- You are given two code listings. The program in Java you met last year (anyone remember what it does?);
- Have a look at the equivalent C program.
- Try to identify the main structures (adopt a top-down approach), i.e. input/output/processing. Work in pairs/groups for 5 minutes.
- What are the conclusions?

Input and Output

- All input and output in C is done in streams;
- A library package has been evolved which is known as known as the “Standard I/O Library” which is used in any C program by using stdio.h header.
- A stream is a sequence of bytes of data. A sequence of bytes flowing into a program is an input stream; a sequence of bytes flowing out of a program is an output stream.
- The major advantage of streams is that input/ output programming is device independent. The program sees input/output as a continuous stream of bytes no matter where the input is coming from or going to.

Input and Output

- There are two groups of C streams: text and binary.
- Text streams consists of characters and are organised into lines (up to 255 characters long, terminated by the EOL special character). Some characters have special functions.
- Binary streams can handle any data, including characters. Data bytes are not interpreted in any way, but transferred as they are.

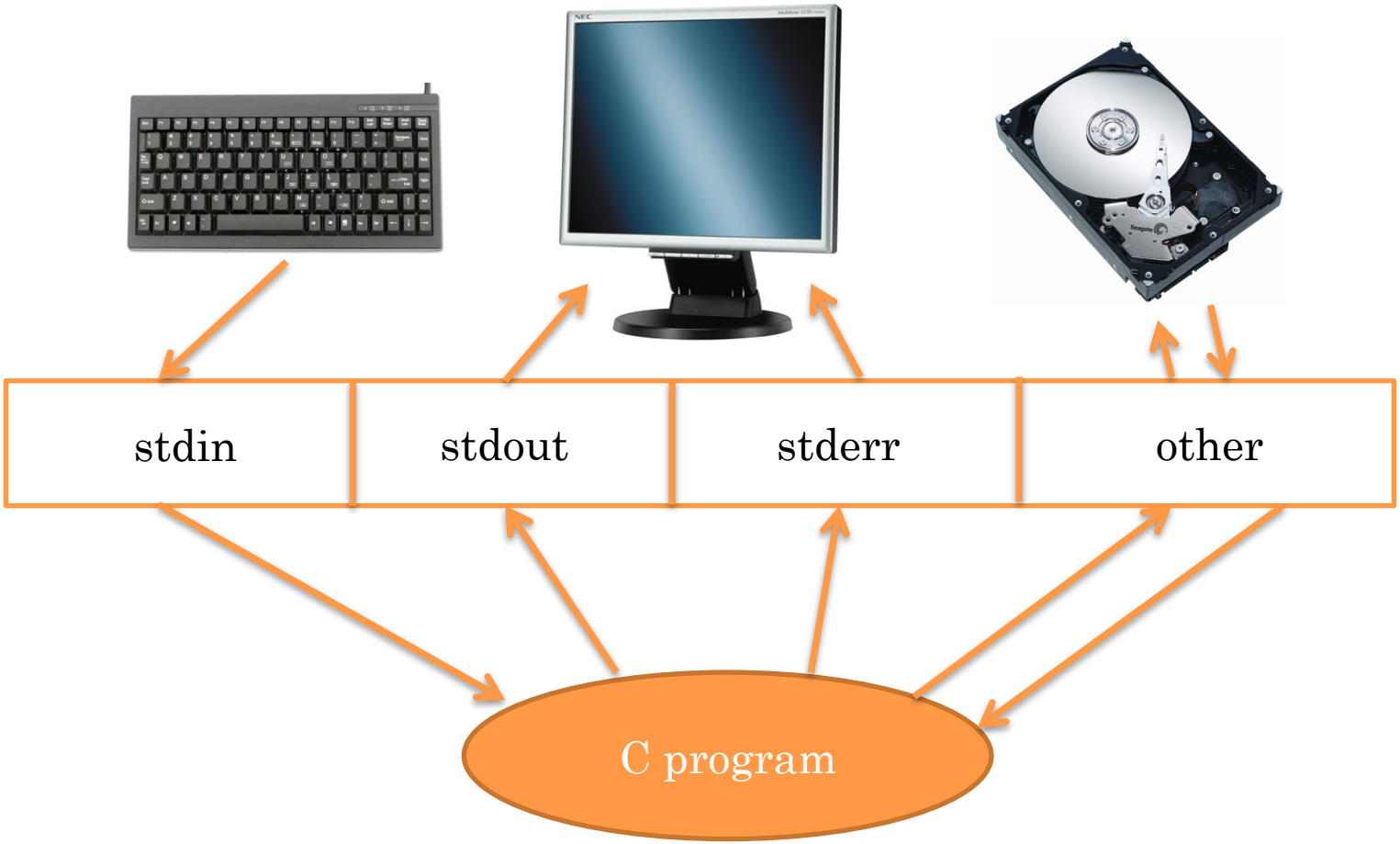
Input and Output
Standard Streams

- ANSI C has three predefined (standard) streams:

NAME	STREAM	DEVICE
stdin	Standard input	keyboard
stdout	Standard output	screen
stderr	Standard error	screen

- Standard streams are opened automatically, other streams (e.g. for manipulating data in a file) must be opened explicitly.

Input and Output Standard Streams



Input and Output Stream functions

- The standard library stream input/output functions:

Uses standard stream	Requires a stream name	Description
printf()	fprintf()	formatted output
vprintf()	vfprintf()	formatted output with variable argument list
puts()	fputs()	string output
putchar()	putc(), fputc()	character output
scanf()	fscanf()	formatted input
gets()	fgets()	string input
getchar()	getc(), fgetc()	character input
perror()		string output to stderr only

Input and Output
Input functions

- Example:

```
#include  <stdio.h>
main()
  {
  char buffer[256];
  puts(gets(buffer));
  return 0;
  }
```

- Input functions are divided into three hierarchy levels:
  - Character input;
  - Line input;
  - Formatted input;

Input and Output
Character input

- The character input functions read one character at a time from an input stream (recall ARM);
- Each of these functions returns the next character in the stream, or EOF (when the end of the file or an error has occurred). EOF is a symbolic constant defined in stdio.h as -1.
- Character input functions differ in terms of buffering and echoing:
  1. Some character input functions are buffered. The operating system holds all characters in a temporary storage space until you press Enter, and then the system sends the characters to the stdin stream. Others are unbuffered, meaning that each character is sent to stdin as soon as the key is pressed.
  2. Some input functions automatically echo each character to stdout as it is received. Others don't echo, i.e. the character is sent to stdin and not stdout (remember that stdout is assigned to the screen).

Input and Output
The getchar() function

- The function `getchar()` obtains the next character from the stream stdin. It provides buffered character input with echo.
- Prototype: `int getchar(void);`
- Example:

```
#include <stdio.h>
main()
{
   int ch;
   while ((ch = getchar()) != '\n')
   putchar(ch);   // not yet explained, but you can guess its purpose
   return 0;
}
```

- Suppose that we type "Hello world!\n". What is the effect of the above program?
- Homework: Write a small program that inputs the entire line of text using `getchar()` function. Take into account that the maximal line length is 80 characters.

Input and Output
Line input

- The line input functions read a line from an input stream.
- They read all characters until the newline character '\n' is found.
- The standard library has two line input functions, `gets()` and `fgets()`.

Input and Output
The gets() function

- The purpose of this function is to read a line from stdin and store it in a string.
- Prototype: `char *gets(char *str);`
- The argument of `gets()` is a pointer to type `char` and returns a pointer to type `char`.
- It reads characters from stdin until a newline (\n) or end-of-file is encountered. The newline is replaced with a null character, and the string is stored at the location indicated by `str`.
- The return value is a pointer to the string (the same as `str`). If an error is encountered, or end-of-file reached before any characters are input, a null pointer is returned.
- Before calling `gets()`, we must allocate sufficient memory space to store the string. If the space hasn't been allocated, the string might overwrite other data and cause program errors.

Input and Output
The gets() function

- Example:

```
#include <STDIO.H>
/* Allocate a character array to hold input. */
char input[81];
main()
{
   puts("Enter some text, then press Enter: ");
   gets(input);
   return 0;
}
```

Input and Output
The fgets() function

- The `fgets()` reads a line of text from an input stream, but allows to specify the specific input stream and the maximum number of characters.
- The `fgets()` function is used to input text from disk files.
- Prototype: `char *fgets(char *str, int n, FILE *fp);`
- The last parameter, `FILE *fp`, is used to specify the input stream. To use it for input from `stdin`, we specify it as the input stream.
- The pointer `*str` indicates where the input string is stored.
- The argument `n` specifies the maximum number of characters.
- The `fgets()` function reads characters from the input stream until a newline or end-of-line is encountered or n – 1 characters have been read. The newline is included in the string and terminated with a \0 before it is stored.
- The return values of `fgets()` are the same as for `gets()`.

Input and Output
The fgets() function

- Example:

```
#include <stdio.h>
#define MAXLEN 10
main()
{
   char buffer[MAXLEN];
   puts("Enter text a line at a time; enter a blank to exit.");
   while (1)
   {
      fgets(buffer, MAXLEN, stdin);
      if (buffer[0] == `\n') break;
      puts(buffer);
    }
   return 0;
}
```

Input and Output
The fgets() function

- Input:
  - Enter text a line at a time; enter a blank to exit.
  - Peter is doing a C course(\n)
  - Peter is
  - doing a C
  - course
- If a line of length greater than `MAXLEN` is entered, the first `MAXLEN - 1` characters are read by the first call to `fgets()`. The remaining characters remain in the keyboard buffer and are read by the next call to `fgets()`.

Input and Output
Formatted input

- Previously discussed input functions take one or more characters from an input stream and put them in memory. No interpretation or formatting of the input has been done, and no way to input numeric variables.
- For example, how would you input the value 10.54 from the keyboard and assign it to a type float variable?
- The answer is: use the `scanf()` and `fscanf()` functions.
- These two functions are identical, except that `scanf()` always uses `stdin`, whereas for `fscanf()` we can specify the input stream (convenient for input from files).

Input and Output
The scanf() function

- The scanf () function has a variable number of arguments (a minimum of two).
- The first argument is a format string that uses special characters to tell scanf () how to interpret the input (the conversion specification).
- The second and additional arguments are the addresses of the variable(s) to which the input data is assigned.
- Example: scanf ("%d", &x) ;
  - It says the program to take an integer value (conversion specification %d) and assign it to the variable x, given by the address operator &).

Input and Output
The scanf() function

- Each conversion specification begins with the % character and contains optional and required components in a certain order.
- The scanf () function applies the conversion specifications in the format string, in order, to the input fields. An input field is a sequence of non-white-space characters that ends when the next white space is encountered or when the field width, if specified, is reached.
- Input from scanf() is buffered; no characters are actually received from stdin until the user presses Enter. At this point the entire line of characters is processed **sequentially** by scanf().
- scanf() is completed only when enough input has been received to match the specifications in the format string, and processes only enough characters from stdin to satisfy its format string. Extra characters, if any, remain waiting in stdin. These characters can cause problems.

Input and Output
The scanf() function

scanf(%<*><field width><precision modifier>type specifier, … &…)

- The optional assignment suppression flag <*> immediately follows the % and implies to perform the conversion corresponding to the current conversion specifier but to ignore the result (not assign it to any variable, i.e. skip that field).
- The next component, the <field width>, is also optional. It is a decimal number specifying the width, in characters, of the input field, i.e. it specifies how many characters from stdin should be examined for the current conversion. If it is omitted, the input field extends to the next white space.
- The next component is the optional <precision modifier>, a single character that can be h, l, or L. If present, the precision modifier changes the meaning of the type specifier that follows it.
- The only compulsory component of the conversion specifier (besides the %) is the type specifier. It consists of one or more characters that interpret the input. These characters are listed and described in the table given below.

Input and Output
The scanf() function

The type specifier characters

Type	Argument	Meaning
d	*int	A decimal integer
i	*int	An integer in decimal, octal (leading O), or hexadecimal (leading Ox) notation
o	*int	An integer in octal notation (without leading O)
x	*int	An integer in hexadecimal notation (without leading Ox)
u	*unsigned int	An unsigned decimal integer
c	*char	One or more characters are read (no terminating \0)
s	*char	A string of non-whitespace characters is read (terminating \0 is added)
e,f,g	*float	A floating-point number in decimal or scientific notation.
[...] [^...]	*char	A string accepting/not accepting the characters listed between the brackets. Input ends when finding a nonmatching character, Enter, or reaching the specified field length (terminating \0 is added).

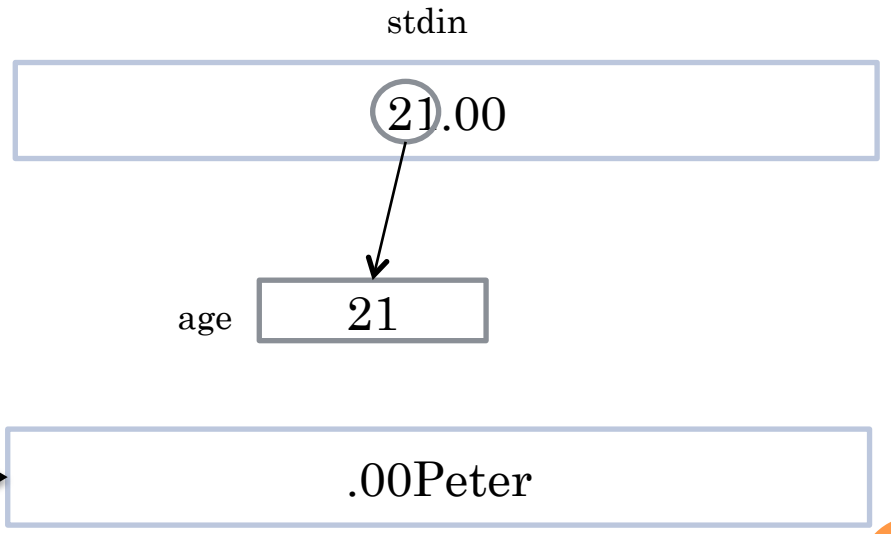
Input and Output
The scanf() function

Precision modifier	Meaning
h	In front of d, i, o, u, or x means that the argument is a pointer to type short instead of type int.
l	In front of d, i, o, u, or x means that the argument is a pointer to type long. In front of e, f, or g, means that the argument is a pointer to type double.
L	In front of e, f, or g, means that the argument is a pointer to type long double.

Input and Output
The scanf() function

```
#include <stdio.h>
 main()
 {
 int age;
 char name[20];

 /* Prompt for user's age. */
 puts("Enter your age.");
 scanf("%d", &age);

 /* Now prompt for user's name. */
 puts("Enter your first name.");
 scanf("%s", name);

 /* Display the data. */
 printf("Your age is %d.\n", age);
 printf("Your name is %s.\n", name);

 return 0;
 }
```

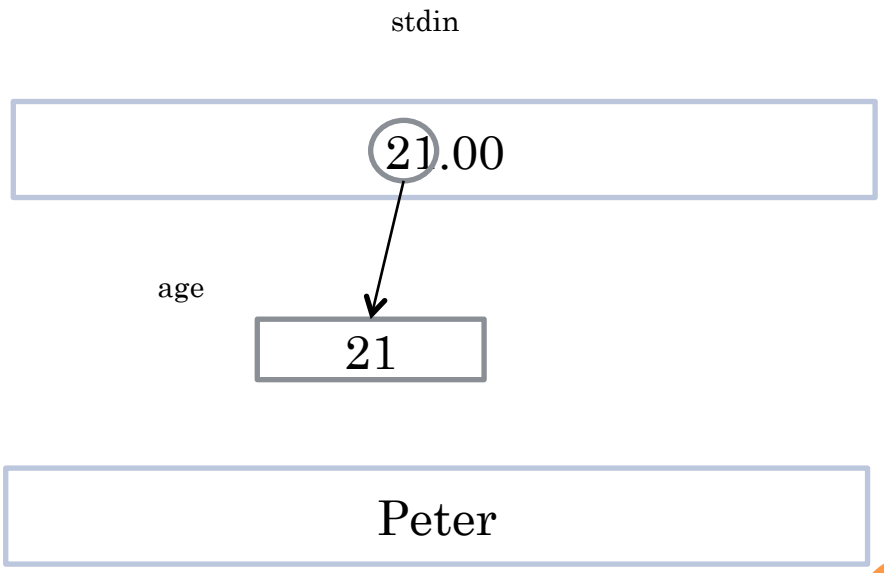
- Enter your age.
- 21.00
- Enter your first name.
- Peter
- Your age is 21.
- Your name is Peter



Input and Output
The scanf() function

```
#include <stdio.h>
 main()
 {
 int age;
 char name[20];

 /* Prompt for user's age. */
 puts("Enter your age.");
 scanf("%d", &age);

/* Clear stdin of any extra characters. */
 fflush(stdin);

 /* Now prompt for user's name. */
 puts("Enter your first name.");
 scanf("%s", name);

 /* Display the data. */
 printf("Your age is %d.\n", age);
 printf("Your name is %s.\n", name);

 return 0;
 }
```

- Enter your age.
- 21.00
- Enter your first name.
- Peter
- Your age is 21.
- Your name is Peter



Input and Output
The scanf() function

- Homework: Explain what is happening in the program given below and give the types and values of the variables upon its execution :

```
#include <stdio.h>
main()
{
int i1, i2;
long l1;
double d1;
char buf1[80], buf2[80];
puts("Enter an integer and a floating point number.");
scanf("%ld %lf", &l1, &d1);
fflush(stdin);
puts("Enter a 5 digit integer (for example, 54321).");
scanf("%2d%3d", &i1, &i2);
fflush(stdin);
puts("Enter your first and last names separated by a space.");
scanf("%[^ ]%s", buf1, buf2);
return 0;
}
```

Input and Output
Output functions

- Output functions are divided into three hierarchy levels:
  - Character output;
  - Line output;
  - Formatted output;

Input and Output
Character Output
The putchar() function

- The function putchar() send a single character to the standard output stream stdout.
- Prototype: int putchar(int c);
- Notice that we can pass either a char type or an integer that represents and ASCII value of a character to putchar().
- Example:

```
#include <stdio.h>
main()
{
int count;
for (count = 14; count < 128; count++ ) putchar(count);
return 0;
}
```

Input and Output
String Output
The puts() function

- It is more usual to output strings than single characters. The library function puts() sends the string to the stdout stream (screen).
- The function fputs() is identical to puts(), except that it sends a string to a specified stream.
- Prototype: int puts(char *cp);
- *cp is the pointer to the first character of the string.
- The function displays the entire string up to, but not including the terminating null character, adding a newline at the end.
- It returns a positive value if successful or EOF (-1) if an error occurred.

Input and Output
Formatted Output
The printf() and fprintf() functions

- C library functions for formatted output printf() and fprintf() can be used to display numerical data, strings and single characters.
- These two functions are identical, except that printf() always sends output to stdout, whereas in fprintf() one can specify the output stream. fprintf() is generally used for output to disk files.
- The printf() function has a variable number of arguments (minimum one). The first and only required argument is the format string, which tells printf() how to format the output. The optional arguments are variables and expressions whose values we want to display.

Input and Output
Formatted Output
The printf() and fprintf() functions

- The format string can contain the following:
  1. Zero, one, or more conversion commands that tell printf() how to display a value in its argument list. A conversion command consists of % followed by one or more characters.
  2. Characters that are not part of a conversion command and are displayed verbatim.
- The components of the conversion command

  %[flag][field_width][.[precision]][l]conversion_char

Input and Output
Formatted Output
The printf() and fprintf() functions

The conversion character list

Conversion character	Meaning
d,i	Displays a signed integer in decimal notation
u	Displays an unsigned integer in decimal notation
o	Displays an unsigned integer in octal notation
x,X	Display an integer in unsigned hexadecimal notation (x - lowercase X – uppercase)
c	Displays a single character
e,E	Display a float or double in scientific notation
f	Display a float or double in decimal notation
g,G	The e or E format is used if the exponent is less than -3 or greater than the precision (which defaults to 6). f format is used otherwise. Trailing zeros are truncated.
s	Displays a string

Input and Output
Formatted Output
The printf() and fprintf() functions

- We can place the l modifier just before the conversion character. This modifier applies only to the conversion characters o, u, x, X, i, d, and b.
- When applied, this modifier specifies that the argument is a type long rather than a type int.
- If the l modifier is applied to the conversion characters e, E, f, g, or G, it specifies that the argument is a type double.
- If an l is placed before any other conversion character, it is ignored.

Input and Output
Formatted Output
The printf() and fprintf() functions

- The precision specifier consists of a decimal point (.) by itself or followed by a number. A precision specifier applies only to the conversion characters e, E, f, g, G, and s. It specifies the number of digits to display to the right of the decimal point or, when used with s, the number of characters to output. If the decimal point is used alone, it specifies a precision of 0.
- The field-width specifier determines the minimum number of characters output. The field-width specifier can be the following:
  1. A decimal integer not starting with 0. The output is padded on the left with spaces to fill the designated field width.
  2. A decimal integer starting with 0. The output is padded on the left with zeros to fill the designated field width.
  3. The * character. The value of the next argument (which must be an int) is used as the field width. For example, if w is a type int with a value of 10, the statement printf("%*d", w, a) prints the value of a with a field width of 10 (allows for variable format).
- If no field width is specified, or if the specified field width is narrower than the output, the output field is just as wide as needed.

Input and Output
Formatted Output
The printf() and fprintf() functions

- The optional part flag immediately follows the % character. There are four available flags:
  1. – This means that the output is left-justified in its field rather than right-justified (the default).
  2. + This means that signed numbers are always displayed with a leading + or –.
  3. _ A space means that positive numbers are preceded by a space.
  4. # This applies only to x, X, and o conversion characters. It specifies that nonzero numbers are displayed with a leading 0X or 0x (for x and X) or a leading 0 (for o).

Input and Output
Formatted Output
The printf() and fprintf() functions

- Write the output produced by the execution of this program:

```
#include <stdio.h>
char *m1 = "Binary";
char *m2 = "Decimal";
char *m3 = "Octal";
char *m4 = "Hexadecimal";
main()
{
float d1 = 10000.123;
int n, f;
printf("%5f\n", d1);
printf("%10f\n", d1);
printf("%15f\n", d1);
printf("%20f\n", d1);
printf("%25f\n", d1);

puts("\n Press Enter to continue...");
fflush(stdin);
getchar();

for (n=5;n<=25; n+=5) printf("%*f\n", n, d1);

puts("\n Press Enter to continue...");
fflush(stdin); getchar();

printf("%05f\n", d1);
printf("%010f\n", d1);
printf("%015f\n", d1);
printf("%020f\n", d1);
printf("%025f\n", d1);

puts("\n Press Enter to continue...");
fflush(stdin); getchar();

printf("%–15s%–15s%–15s", m2, m3, m4);
for (n = 1;n< 20; n++)
printf("\n%–15d%–#15o%–#15X", n, n, n);

puts("\n Press Enter to continue...");
fflush(stdin); getchar();

printf("%s%s%s%s%n", m1, m2, m3, m4, &n);
printf("\n\nThe last printf() output %d characters.
\n", n);
return 0;
}
```