1.2 Data Input and Output
The Standard C Environment

- The **C** environment assumes the keyboard to be the standard input device referred to as **stdin**

- The VDU is assumed to be the standard output device referred to as **stdout**

- The VDU also serves as the standard error device, and is referred to as **stderr**
Input/Output Functions

- C supports Input/Output operations through functions written in C, that are part of the standard C library.

- These input/output functions may be incorporated into any program by a programmer.

- Any input or output operation happens as a stream of characters.

- The standard I/O functions are available for character-based I/O, or for string-based I/O.
Input/Output Functions

- The standard I/O functions are buffered, i.e., each device has an associated buffer through which any input or output operation takes place.

- After an input operation from the standard input device has occurred, care must be taken to clear the standard input buffer. Otherwise, the previous contents of the buffer may interfere with subsequent input.

- After an output operation, the buffer need not be cleared since subsequent output data will flush the previous buffer contents.
INPUT / OUTPUT FUNCTIONS

FORMATTED INPUT
Ex: scanf

FORMATTED OUTPUT
Ex: printf

UNFORMATTED INPUT
Ex: getchar
getch
geta

UNFORMATTED OUTPUT
Ex: putchar
putch
puts
**Unformatted I/O**

- Should include the standard I/O *header* file `#include <stdio.h>` for input output operations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getchar()</code></td>
<td>Reads a character from the keyboard; usually waits for carriage return.</td>
</tr>
<tr>
<td><code>getche()</code></td>
<td>Reads a character with echo; does not wait for carriage return; not defined by Standard C, but a common extension.</td>
</tr>
<tr>
<td><code>getch()</code></td>
<td>Reads a character without echo; does not wait for carriage return; not defined by Standard C, but a common extension.</td>
</tr>
<tr>
<td><code>putchar()</code></td>
<td>Writes a character to the screen.</td>
</tr>
<tr>
<td><code>gets()</code></td>
<td>Reads a string from the keyboard.</td>
</tr>
<tr>
<td><code>puts()</code></td>
<td>Writes a string to the screen.</td>
</tr>
</tbody>
</table>
Character-Based I/O

- `getch()` is used to accept a character from standard input.
  By default, it accepts characters from the keyboard, and returns the character.

- `putch()` displays the character on to standard output.
  It takes one argument, namely, the character to be output.

- `fflush()` clears the buffer associated with the particular device.
Example

```c
#include <conio.h>
#include <stdio.h>
main( )
{
    char ch;
    ch = getch( );
    fflush(stdin);
    putch(ch);
}
```
getchar() and putchar()

- A macro call is similar to a function call. It consists of a macro name followed by a comma-separated argument list enclosed in a pair of parentheses.
- The macro `getchar()` by default accepts a character from the keyboard and returns the character.
- The macro `putchar()` is used to display the character on standard output.

It takes the character to be output as an argument.
```
ch = getchar();
fflush(stdin);
putchar(ch);
```
Example

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

int main(void)
{
    char ch;

    printf("Enter some text (type a period to quit).\n");
do {
    ch = getchar();

    if(islower(ch)) ch = toupper(ch);
    else ch = tolower(ch);

    putchar(ch);
} while (ch != '.');

return 0;
}
```
String-Based I/O

- `gets( )` accepts as a parameter, a string variable, or a string literal (enclosed in double quotes) from the keyboard.

- `puts( )` accepts a string variable, or a string literal to be displayed to standard output.

After displaying the output, the `puts( )` function causes the cursor to be positioned at the beginning of the next line.
```c
#include <stdio.h>
#include <conio.h>

main()
{
    char str[11];
    puts("Enter a string of maximum 10 characters");
    gets(str);
    fflush(stdin);
    puts(str);
}
```
The functions `printf()` and `scanf()` perform formatted output and input—they can read and write data in various formats that are under your control.

- `printf()` function writes data to the console.
- `scanf()` function, its complement, reads data from the keyboard.
- Both functions can operate on any of the built-in data types, plus null-terminated character strings.
Formatted Input and Output

Keyboard → Standard Input File → Memory

Standard Output File → Monitor → Standard Output File
- **Input**
  
  `scanf("%d", &a);` - Gets an integer value from the user and stores it under the name `a`

  
  `scanf("%d %d", &x, &y);` - ???

- **Output**

  `printf("%d",a);` - Prints the value present in variable `a` on the screen

  
  `printf("Enter x y : ");` - ????
Formatted Output - `printf()`

- Monitor
- Data destination
- Text Stream
- `printf(…)`
- Integer
- Program

234

2 3 4
**printf Function**

- prints information to the screen
- requires two arguments
  - control string
    - Contains text, conversion specifiers or both
  - Identifier to be printed

**Example**

double angle = 45.5;
printf("Angle = %.2f degrees \n", angle);

Output:
Angle = 45.50 degrees
printf(control string, arg1, arg2, ..argn);

printf() function returns ?????
the number of characters written or a negative value if an error occurs

#include <stdio.h>

main()
{
    int i = 0;
    i = printf("abcde\n");
    printf("total characters printed \d\n", i);
}

Conversion Specification

<table>
<thead>
<tr>
<th>%</th>
<th>Flag</th>
<th>Minimum Width</th>
<th>Precision</th>
<th>Size</th>
<th>Code</th>
</tr>
</thead>
</table>
Examples...

- `printf("%d%c%f", 23, 'z', 4.1);`
  
  23z4.100000

- `printf("%d %c %f", 23, 'z', 4.1);`
  
  23 z 4.100000

```c
int i, j;
float x, y;
i = 10;
j = 20;
x = 43.2892f;
y = 5527.0f;
printf("i = %d, j = %d, x = %f, y = %f\n", i, j, x, y);
```

- **Output:**

  i = 10, j = 20, x = 43.289200, y = 5527.000000
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Output Type</th>
<th>Specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>short, int</td>
<td>int</td>
<td>%i, %d</td>
</tr>
<tr>
<td>int</td>
<td>short</td>
<td>%hi, %hd</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>%li, %ld</td>
</tr>
<tr>
<td>int</td>
<td>unsigned int</td>
<td>%u</td>
</tr>
<tr>
<td>int</td>
<td>unsigned short</td>
<td>%hu</td>
</tr>
<tr>
<td>long</td>
<td>unsigned long</td>
<td>%lu</td>
</tr>
<tr>
<td><strong>Floating-Point Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float, double</td>
<td>double</td>
<td>%f, %e, %E, %g, %G</td>
</tr>
<tr>
<td>long double</td>
<td>long double</td>
<td>%LF, %Le, %LE, %Lg, %LG</td>
</tr>
<tr>
<td><strong>Character Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>char</td>
<td>%c</td>
</tr>
</tbody>
</table>
printf() in brief...

Displays information on screen
Returns the number of characters printed
Displays the text you put inside the double quotes
Requires the backslash character - an escape sequence - to Display some special characters
Displays values of variables by using the % conversion character
Output of -145

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Value Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>%i</td>
<td>-145</td>
</tr>
<tr>
<td>%4d</td>
<td>-145</td>
</tr>
<tr>
<td>%3i</td>
<td>-145</td>
</tr>
<tr>
<td>%6i</td>
<td>__-145</td>
</tr>
<tr>
<td>%-6i</td>
<td>-145__</td>
</tr>
<tr>
<td>%8i</td>
<td>____-145</td>
</tr>
<tr>
<td>%-8i</td>
<td>-145____</td>
</tr>
</tbody>
</table>
Output of 157.8926

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Value Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>%f</td>
<td>157.892600</td>
</tr>
<tr>
<td>%6.2f</td>
<td>157.89</td>
</tr>
<tr>
<td>%7.3f</td>
<td>157.893</td>
</tr>
<tr>
<td>%7.4f</td>
<td>157.8926</td>
</tr>
<tr>
<td>%7.5f</td>
<td>157.89260</td>
</tr>
<tr>
<td>%e</td>
<td>1.578926e+02</td>
</tr>
<tr>
<td>%3.3E</td>
<td>1.579E+02</td>
</tr>
</tbody>
</table>
int sum = 65;
double average = 12.368;
char ch = ‘b’;

Show the output line generated by the following:

```c
printf("Sum = %5i; Average = %7.1f \n", sum, average);
printf("Sum = %4i \n Average = %8.4f \n", sum, average);
printf("Sum and Average \n\n %d % .1f \n", sum, average);
printf("Character is %c; Sum is %c \n", ch, sum);
printf("Character is %i; Sum is %i \n", ch, sum);
```
Solution:

Sum = 65; Average = 12.4
Sum = 65
Average = 12.3680

Sum and Average
65 12.4

Character is b; Sum is A
Character is 98; Sum is 65
Flags with printf( )

- Trailing zeroes
+ sign, either + or – will precede
0 leading zeros
#0 octal data items to be preceded by 0
#x hexa data items to be preceded by 0x
#f decimal point to be present
#e decimal point to be present
#g decimal point to be present, prevents truncation of trailing zeroes
Escape sequences

- You can represent any member of the execution character set by an escape sequence.

- These sequences are primarily used to put nonprintable characters in character and string literals.

- To put such characters as tab, carriage return, and backspace into an output stream.
C includes the backslash character - an escape sequence - to display some special characters

```c
#include <stdio.h>

int main(void) {
    printf("\n\t\nThis is a test.\n");
    return 0;
}
```

The program outputs a new line and a tab and then prints the string This is a test.
Tab \texttt{\textbackslash t} and NewLine \texttt{\textbackslash n} Character

- \texttt{printf("\%d\t\%c \t\%5.1f\n", 23, 'z', 14.2)};
- \texttt{printf("\%d\t\%c \t\%5.1f\n", 107, 'A', 53.6)};
- \texttt{printf("\%d\t\%c \t\%5.1f\n", 1754, 'F', 122.0)};
- \texttt{printf("\%d\t\%c \t\%5.1f\n", 3, 'P', 0.1)};

\begin{tabular}{ccc}
23 & z & 14.2 \\
107 & A & 53.6 \\
1754 & F & 122.0 \\
3 & P & 0.1 \\
\end{tabular}

- \texttt{printf("The number \%d is my favorite number.", 23)};

The number 23 is my favorite number.
Examples

- Another common escape sequence is ", which represents the " character:
  
  ```c
  printf(""Hello!"\"");
  "Hello!"
  
  To print a single \ character, put two \ characters in the string:
  
  ```c
  printf("\"\"");
  "\""
<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>\w</td>
<td>Vertical tab</td>
</tr>
<tr>
<td>\a</td>
<td>Alert</td>
</tr>
<tr>
<td>?</td>
<td>Question mark</td>
</tr>
<tr>
<td>\N</td>
<td>Octal constant (where N is an octal constant)</td>
</tr>
<tr>
<td>\xN</td>
<td>Hexadecimal constant (where N is a hexadecimal constant)</td>
</tr>
</tbody>
</table>
Things to Remember!!!

- Compilers aren’t required to check that the number of conversion specifications in a format string matches the number of output items.

- `printf("%d %d\n", i);`
  
  Too many conversion specifications

- `printf("%d\n", i, j);`
  
  Too few conversion specifications

- `printf("%f %d\n", i, x);
  
  /*** WRONG ***/`
Time to Test Your C Skills

- `printf("%d %d %d\n", 44, 55)`
  
  ```
  44 55 0
  //Three conversion specifications but only two values
  ```

- `printf("%d %d\n", 44, 55, 66)`
  
  ```
  44 55
  //printf ignores the third value
  ```
Formatting Input - `scanf()`

'2' '3' '4' '.' '2' → `scanf(…)` → 234.2
**scanf Function**

- inputs values from the keyboard
- required arguments
  - control string
  - memory locations that correspond to the specifiers in the control string
- Example:
  ```c
  double distance;
  char unit_length;
  scanf("%lf %c", &distance, &unit_length);
  ```

* It is very important to use a specifier that is appropriate for the data type of the variable


- `scanf()` is the **general-purpose** console input routine. It can read all the built-in data types and automatically convert numbers into the proper internal format.
- `scanf()` returns the **number of data items** successfully assigned a value. If an error occurs, it returns **EOF**.
- The **control_string** determines how values are read into the variables pointed to in the argument list.

<table>
<thead>
<tr>
<th>%</th>
<th>Flag</th>
<th>Maximum width</th>
<th>Size</th>
<th>Code</th>
</tr>
</thead>
</table>

There is no precision, if found (it goes to error state)
You Must Pass `scanf( )` ADDRESSES

- All the variables used to receive values through `scanf( )` must be passed by their addresses.
- This means that all arguments must be pointers.

- To read an integer into the variable count:
  ```c
  scanf("%d", &count);
  ```

- To read a string into the character array `str`:
  ```c
  scanf("%s", str);
  ```

In this case, `str` is already a pointer and need not be preceded by the `&` operator.
Example

```c
int i, j;
float x, y;
scanf("%d%d%f%f", &i, &j, &x, &y);
```

Sample input:
- 1 -20 .3 -4.0e3

/* scanf will assign 1, –20, 0.3, and –4000.0 to i, j, x, and y, respectively */
<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Reads a floating-point value (C99 only).</td>
</tr>
<tr>
<td>%c</td>
<td>Reads a single character.</td>
</tr>
<tr>
<td>%d</td>
<td>Reads a decimal integer.</td>
</tr>
<tr>
<td>%i</td>
<td>Reads an integer in either decimal, octal, or hexadecimal format.</td>
</tr>
<tr>
<td>%e</td>
<td>Reads a floating-point number.</td>
</tr>
<tr>
<td>%f</td>
<td>Reads a floating-point number.</td>
</tr>
<tr>
<td>%g</td>
<td>Reads a floating-point number.</td>
</tr>
<tr>
<td>%o</td>
<td>Reads an octal number.</td>
</tr>
<tr>
<td>%s</td>
<td>Reads a string.</td>
</tr>
<tr>
<td>%x</td>
<td>Reads a hexadecimal number.</td>
</tr>
<tr>
<td>%p</td>
<td>Reads a pointer.</td>
</tr>
<tr>
<td>%n</td>
<td>Receives an integer value equal to the number of characters read so far.</td>
</tr>
<tr>
<td>%u</td>
<td>Reads an unsigned decimal integer.</td>
</tr>
<tr>
<td>%[ ]</td>
<td>Scans for a set of characters.</td>
</tr>
<tr>
<td>%%</td>
<td>Reads a percent sign.</td>
</tr>
</tbody>
</table>
Example

Try specifying field width with scanf.........

```c
int a, b, c;
scanf("%3d %3d %3d", &a, &b, &c);
```

**Inputs:**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>123</td>
<td>456</td>
<td>789</td>
</tr>
<tr>
<td>123456789</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>5678</td>
<td>9</td>
</tr>
</tbody>
</table>
Assignment suppression character: “*d”

```c
scanf(“%d %*d %d”, &a, &b, &c);
```

I/p:

2 3 4

2 is read and assigned to a
3 is read and but not assigned to b
4 is read and assigned to c
EOF

- White spaces, width specifications, EOF, errors (Invalid Characters), stops the scanf function.
- In ASCII, whitespace characters are space (' ')
- If the user signals that there is no more input by keying EOF, then scanf terminates the input process.

- If scanf encounters an invalid character when it is trying to convert the input to the stored data type, it stops.

- Finding a non-numeric character when it is trying to read a number.
As it searches for a number, scanf ignores **white-space** (space, horizontal and vertical tab, form-feed, and new-line)

A call of scanf that reads four numbers:
```c
scanf("%d%d%f%f", &i, &j, &x, &y);
```

The numbers can be on one line or spread over several lines:
```plaintext
1
-20 .3
-4.0e3
```

scanf sees a stream of characters (\n represents new-line):
- ••1\n-20•••.3\n•••-4.0e3\n
- ssrsrssssrszsrsrrrrrrrrrrrr (s = skipped; r = read)
- scanf “peeks” at the final new-line without reading it.
More Examples

```c
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- scanf would process the new input AS FOLLOWS:
  - `%d`. Stores 1 into i and puts the - character back.
  - `%d`. Stores -20 into j and puts the . character back.
  - `%f`. Stores 0.3 into x and puts the - character back.
  - `%f`. Stores -4.0 × 10³ into y and puts the new-line character back.
Time to test your C Skills

- int a = 0;
  scanf("%d", a);
  printf("%d\n", a);

input: 234
output: 0

Why Output is 0?

/* & is missing in scanf(..., &a);*/
What is printed is the original contents of the variable, in this case 0*/
Can you Guess the Output???

```c
/* Adds two fractions */
#include <stdio.h>

int main(void)
{
    int num1, denom1, num2, denom2, result_num, result_denom;

    printf("Enter first fraction: ");
    scanf("%d/%d", &num1, &denom1);

    printf("Enter second fraction: ");
    scanf("%d/%d", &num2, &denom2);

    result_num = num1 * denom2 + num2 * denom1;
    result_denom = denom1 * denom2;
    printf("The sum is %d/%d\n", result_num, result_denom);

    return 0;
}
```

OUTPUT

```
Enter first fraction: 5/6
Enter second fraction: 3/4
The sum is 38/24
```
Thankyou!!!!!!
<table>
<thead>
<tr>
<th>Code</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Hexadecimal output in the form 0xh.hhhhp+d (C99 only).</td>
</tr>
<tr>
<td>%A</td>
<td>Hexadecimal output in the form 0Xh.hhhhp+d (C99 only).</td>
</tr>
<tr>
<td>%c</td>
<td>Character.</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal integers.</td>
</tr>
<tr>
<td>%i</td>
<td>Signed decimal integers.</td>
</tr>
<tr>
<td>%e</td>
<td>Scientific notation (lowercase e).</td>
</tr>
<tr>
<td>%E</td>
<td>Scientific notation (uppercase E).</td>
</tr>
<tr>
<td>%f</td>
<td>Decimal floating point.</td>
</tr>
<tr>
<td>%g</td>
<td>Uses %e or %f, whichever is shorter.</td>
</tr>
<tr>
<td>%G</td>
<td>Uses %E or %F, whichever is shorter.</td>
</tr>
<tr>
<td>%o</td>
<td>Unsigned octal.</td>
</tr>
<tr>
<td>%s</td>
<td>String of characters.</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal integers.</td>
</tr>
<tr>
<td>%x</td>
<td>Unsigned hexadecimal (lowercase letters).</td>
</tr>
<tr>
<td>%X</td>
<td>Unsigned hexadecimal (uppercase letters).</td>
</tr>
<tr>
<td>%p</td>
<td>Displays a pointer.</td>
</tr>
<tr>
<td>%n</td>
<td>The associated argument must be a pointer to an integer. This specifier causes the number of characters written (up to the point at which the %n is encountered) to be stored in that integer.</td>
</tr>
<tr>
<td>%%</td>
<td>Prints a % sign.</td>
</tr>
</tbody>
</table>