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,

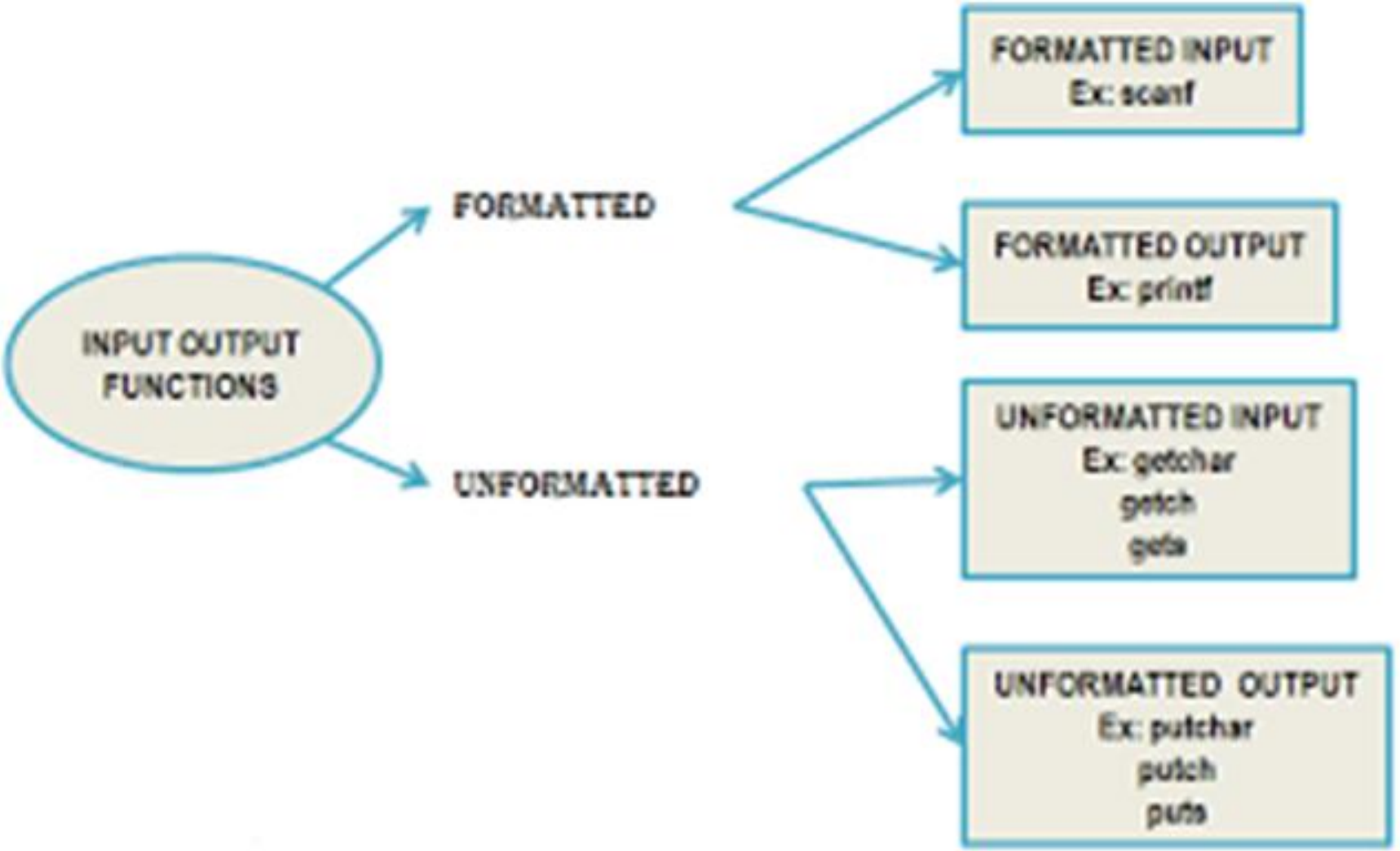
  i.e., each device has an associated buffer through which any input or output operation takes place

- 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



Unformatted I/O

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

Function	Operation
getchar( )	Reads a character from the keyboard; usually waits for carriage return.
getche( )	Reads a character with echo; does not wait for carriage return; not defined by Standard C, but a common extension.
getch( )	Reads a character without echo; does not wait for carriage return; not defined by Standard C, but a common extension.
putchar( )	Writes a character to the screen.
gets( )	Reads a string from the keyboard.
puts( )	Writes a string to the screen.

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

```
#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 to standard output.

  It takes the character to be output as an argument.

```
ch = getchar( );
fflush(stdin);
putchar(ch);
```

Example

```
#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.

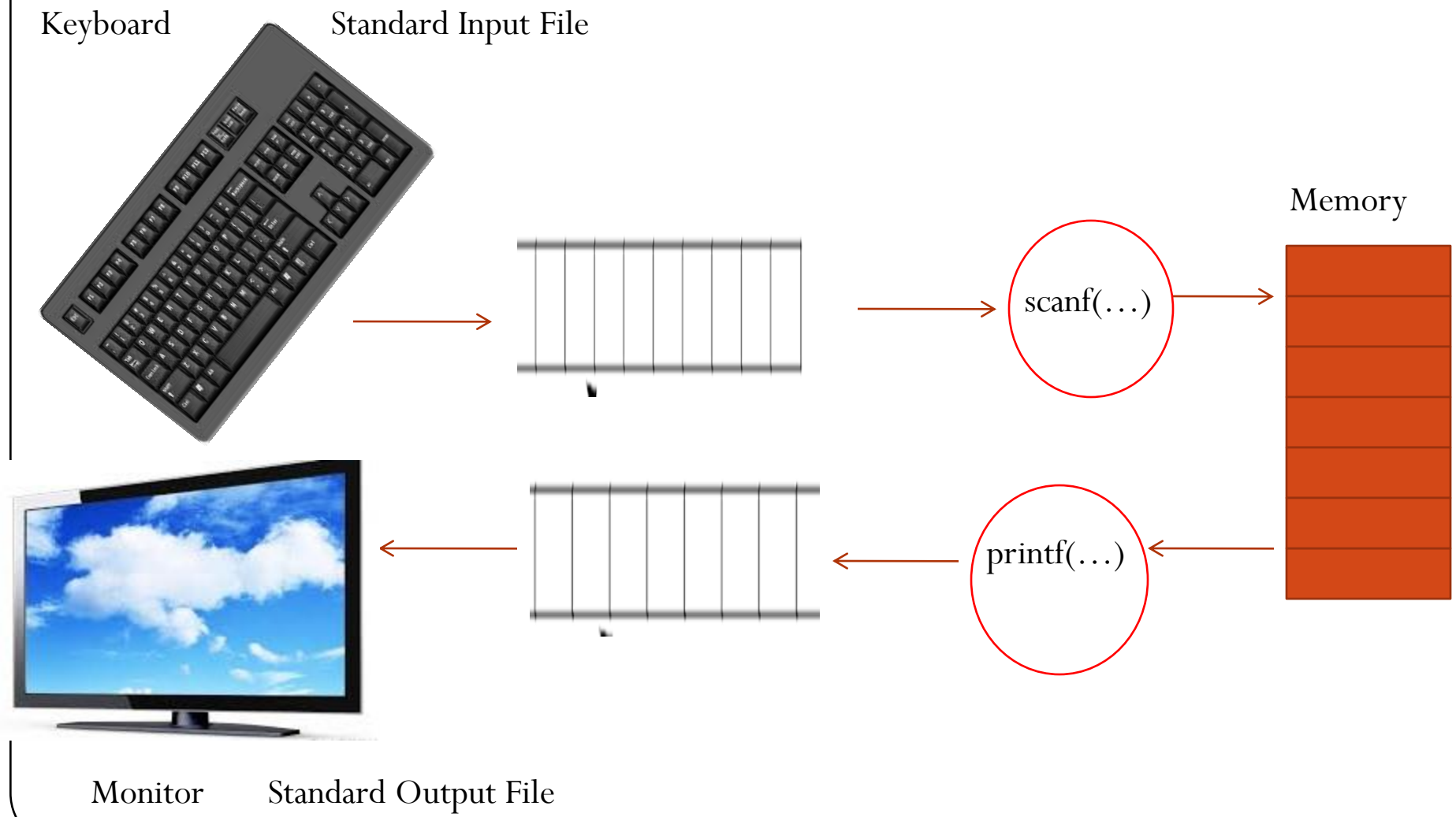
Example

```
#include <stdio.h>
#include <conio.h>
main( )
{
  char str[11];
  puts("Enter a string of maximum 10 characters");
  gets(str);
  fflush(stdin);
  puts(str);
 }
```

Formatted Console I/O

- 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



- Input

  scanf(“%d”,&a); -Gets an integer value from the user and stores it under the name *a*

  scanf(“%d %d”, &x, &y); - ???

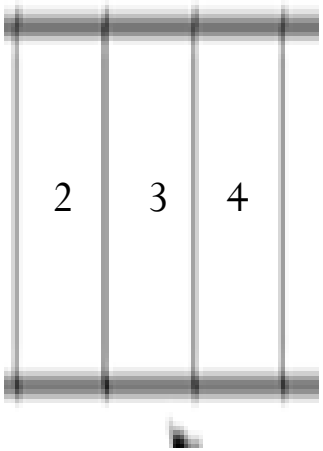
- Output

  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(…)

234

Integer

Program

- **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

%	Flag	Minimum Width	Precision	Size	Code

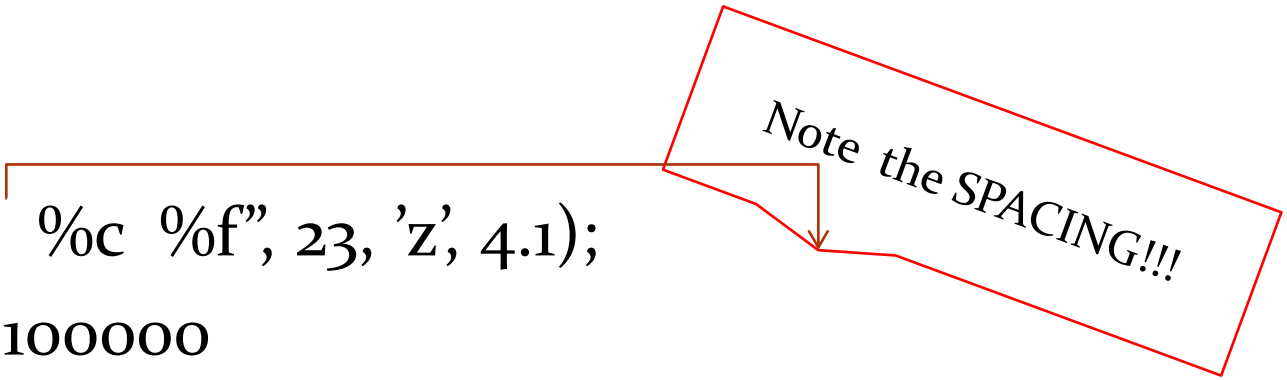
Examples…

- printf(“%d%c%f”, 23, ’z’, 4.1);

  23z4.100000

- printf(“%d %c %f”, 23, ’z’, 4.1);

  23 z 4.100000




```
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
```

Variable Type	Output Type	Specifier
	Integer Values	
short, int	int	%i, %d
int	short	%hi, %hd
long	long	%li, %ld
int	unsigned int	%u
int	unsigned short	%hu
long	unsigned long	%lu
	Floating-Point Values	
float, double	double	%f, %e, %E, %g, %G
long double	long double	%LF, %Le, %LE, %Lg, %LG
	Character Values	
char	char	%c

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

Specifier	Value Printed
%i	-145
%4d	-145
%3i	-145
%6i	__-145
%-6i	-145__
%8i	____-145
%-8i	-145____

Output of 157.8926

Specifier	Value Printed
%f	157.892600
%6.2f	157.89
%7.3f	**157.893**
%7.4f	157.8926
%7.5f	157.89260
%e	1.578926e+02
%.3E	1.579E+02

%2hd	Short integer-2 print positions
%4d	Integer-4 print positions
%7.2	Float -7 print positions nnnn.dd
%10.3Lf	Long double 10 positions nnnnnn.dd

int sum = 65;
double average = 12.368;
char ch = 'b';

Show the output line generated by the following :

```
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 preceed
0	leading zeros
#0	octal data items to be preceeded by 0
#x	hexa data items to be preceeded 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

```
#include <stdio.h>

int main(void)
{
  printf(''\n\tThis is a test.");

  return 0;
}
```

The program outputs a new line and a tab
and then prints the string **This is a test.**

Tab '\t' and NewLine '\n' Character

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

```
23      z       14.2
107     A       53.6
1754    F       122.0
3       P       0.1
```

- 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:

```
printf("\"Hello!\"");
```

```
"Hello!"
```

To print a single \ character, put two \ characters in the string:

```
printf("\\");
```

```
\
```

Code	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\ \	Backslash
\v	Vertical tab
\a	Alert
\?	Question mark
\N	Octal constant (where N is an octal constant)
\xN	Hexadecimal constant (where N is a hexadecimal constant)

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

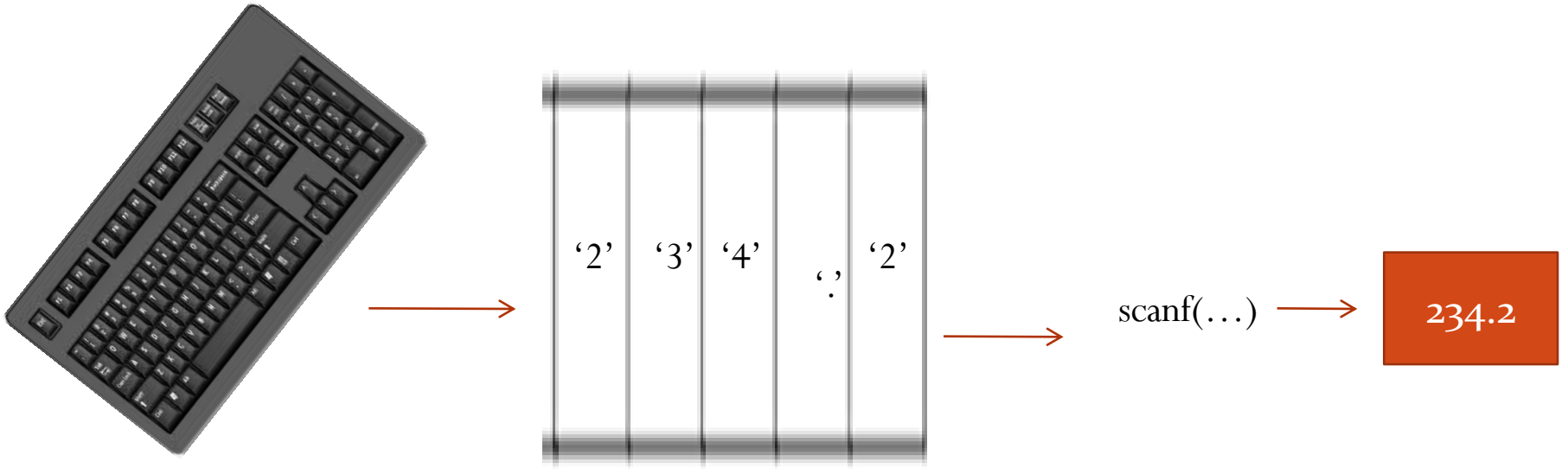
  //Three conversion specifications but only two values

- printf("%d %d\n", 44, 55, 66)

  44 55

  //printf ignores the third value

Formatting Input - scanf()



- **scanf** Function
  - inputs values from the keyboard
  - required arguments
    - control string
    - memory locations that correspond to the specifiers in the control string
- Example:

```
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(control string, arg1, arg2, …..argn);

- **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.

%	Flag	Maximum width		Size	Code

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:

  scanf("%d", **&count**);

- To read a string into the character array str:

  scanf("%s", **str**);

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

Example

```
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 */

Code	Meaning
%a	Reads a floating-point value (C99 only).
%c	Reads a single character.
%d	Reads a decimal integer.
%i	Reads an integer in either decimal, octal, or hexadecimal format.
%e	Reads a floating-point number.
%f	Reads a floating-point number.
%g	Reads a floating-point number.
%o	Reads an octal number.
%s	Reads a string.
%x	Reads a hexadecimal number.
%p	Reads a pointer.
%n	Receives an integer value equal to the number of characters read so far.
%u	Reads an unsigned decimal integer.
%[ ]	Scans for a set of characters.
%%	Reads a percent sign.

Example

Try specifying field width with scanf…….

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

Inputs:

```
1                   2                   3
123                  456                 789
123456789
1234                 5678                9
```

Assignment suppression character: "*d"

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

EOF

- 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:

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

- The numbers can be on one line or spread over several lines:

1

-20 .3

-4.0e3

scanf sees a stream of characters (\n represents new-line):

- ••1\n-20•••.3\n•••-4.0e3\n
- ssrsrrrsssrrssssrrrrrr (s = skipped; r = read)
- scanf “peeks” at the final new-line without reading it.

More Examples

1-20.3-4.0e3\n

- How do you call using scanf()?

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???

```
/* 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!!!!!

Code	Format
%a	Hexadecimal output in the form 0x*h.hhhh*p+*d* (C99 only).
%A	Hexadecimal output in the form 0X*h.hhhh*P+*d* (C99 only).
%c	Character.
%d	Signed decimal integers.
%i	Signed decimal integers.
%e	Scientific notation (lowercase e).
%E	Scientific notation (uppercase E).
%f	Decimal floating point.
%g	Uses %e or %f, whichever is shorter.
%G	Uses %E or %F, whichever is shorter.
%o	Unsigned octal.
%s	String of characters.
%u	Unsigned decimal integers.
%x	Unsigned hexadecimal (lowercase letters).
%X	Unsigned hexadecimal (uppercase letters).
%p	Displays a pointer.
%n	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.
%%	Prints a % sign.