

# HIGH-LEVEL PROGRAMMING I

Intro to C Programming (Part 3/3) by Prasanna Ghali

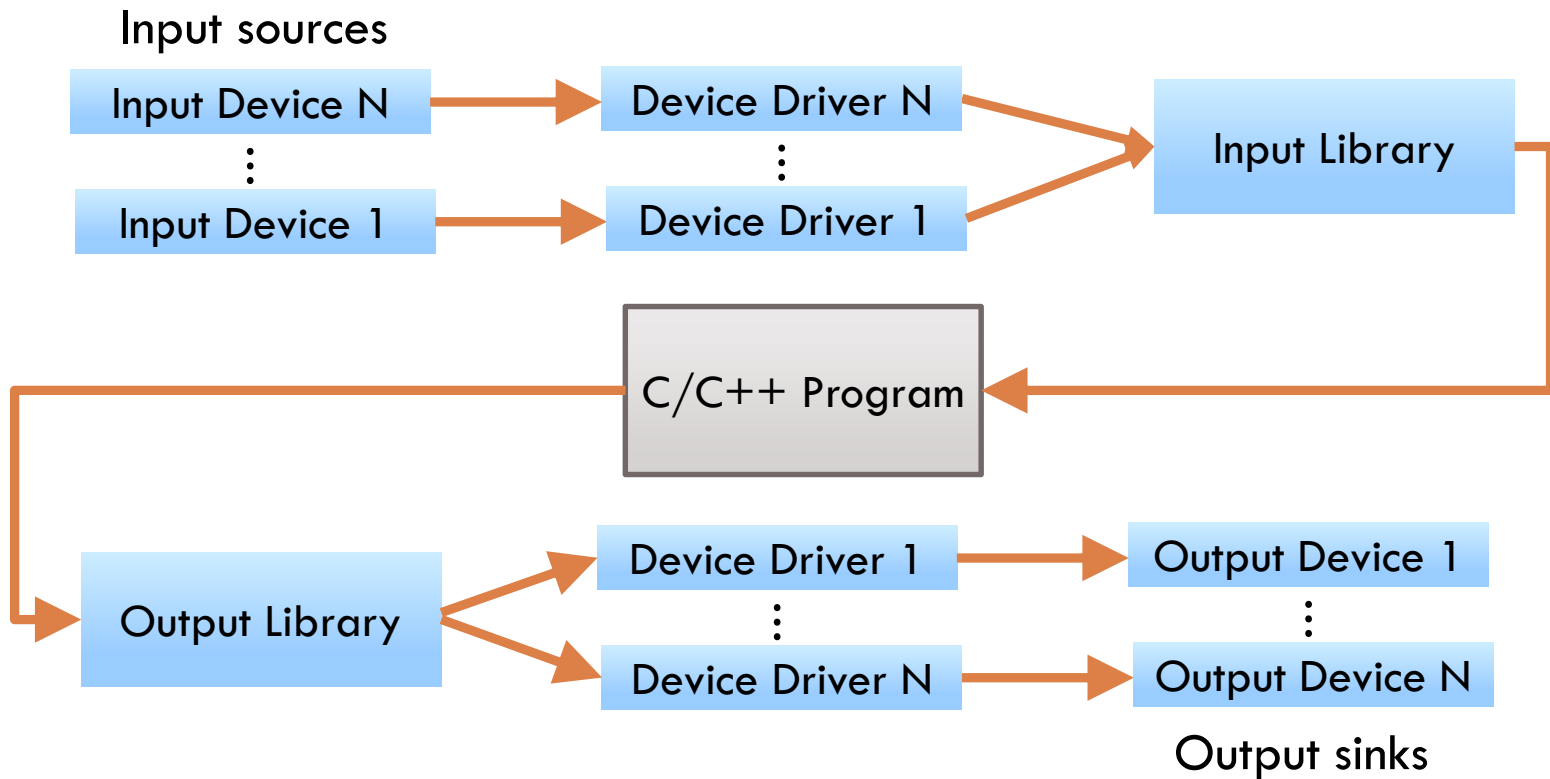
# Outline

2

- Writing values to standard output stream
- Reading values from standard input stream

# Input and Output

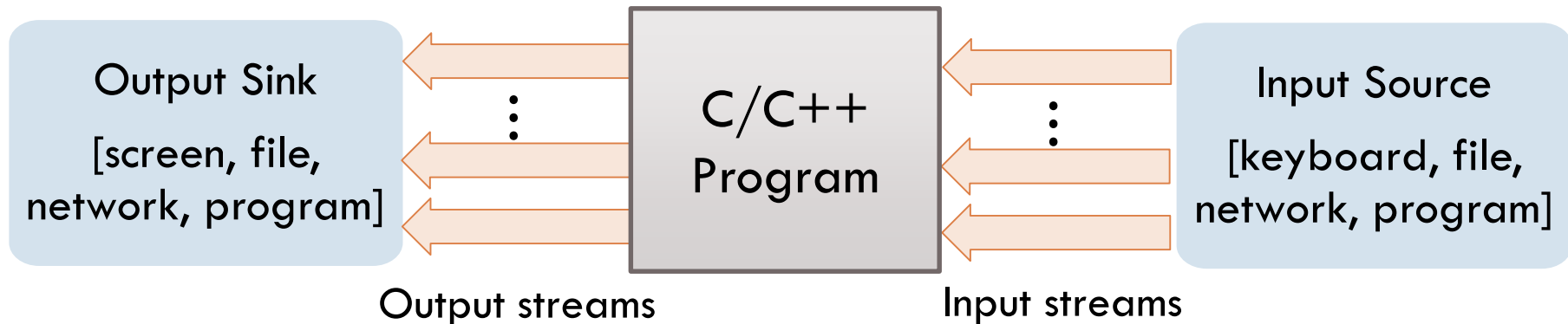
3



# Stream Model

4

- *Stream* is abstraction for sequence of bytes consumed by program as input and generated by program as output



# Header file

5

- `<stdio.h>` must be included to access C input/output functions
- All these functions are compatible with 7-bit ASCII byte and UTF-8 encoding

# File Pointers

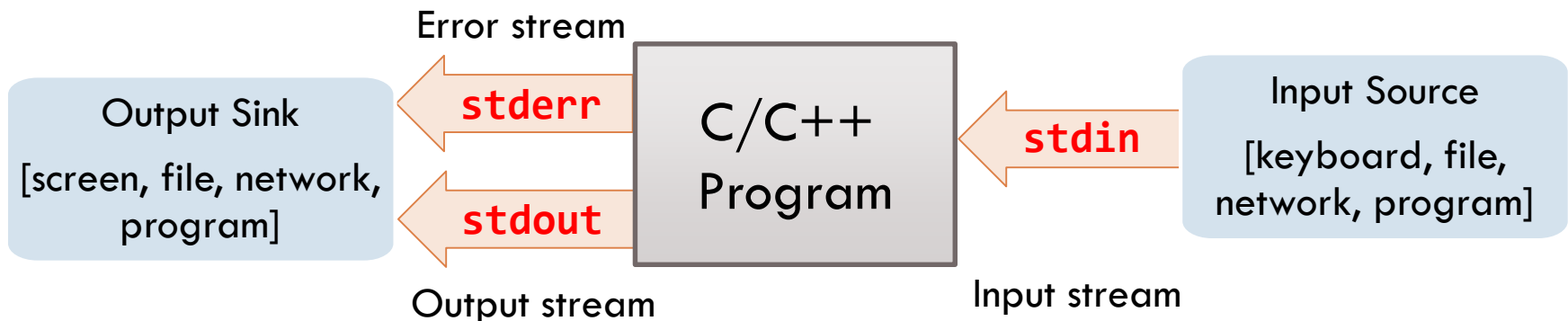
6

- Streams accessed thro' objects of pointer type `FILE*`
  - ▣ `FILE*` referred to as *stream* or *file pointer* type
  - ▣ `<stdio.h>` declares structure type `FILE` – however programmers don't care about implementation details
  - ▣ File pointer points to structure that contains information about file: buffer location, current character position in buffer, whether file is being read from or written to, whether errors or end of file have occurred

# Standard Streams (1 / 2)

7

- I/O library defines three streams ready to use by C/C++ programs - they've *static extent* [lifetime throughout program's execution] and *external linkage* [available from any source file in program]



# Standard Streams (2/2)

8

File pointer	Stream	Default meaning
<b>stdin</b>	standard input	Keyboard
<b>stdout</b>	standard output	Screen
<b>stderr</b>	standard error	Screen



# printf: String Output

9

```
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return 0;
}
```

Call to `printf` displays following line:

Hello World

function to print to  
*standard output*

② function argument

①

`printf("Hello World\n");`

③ format string

Literal characters in format string are printed as is

# Escape Sequences

10

- Backslash (`\`) in a string is called *escape character*
- C/C++ combine `\` with next character to attach special meaning to combo of characters

Sequence	Character Represented
<code>\a</code>	alert (bell) character
<code>\b</code>	backspace
<code>\n</code>	newline
<code>\t</code>	horizontal tab
<code>\\</code>	backslash
<code>\'</code>	single quote
<code>\"</code>	double quote

# printf: Formatted Output (1 / 7)

11

```
#include <math.h>
#include <stdio.h>

int main(void) {
    double px = 0.0, py = 0.0;
    double qx = 3.0, qy = 4.0;
    double w = qx - px, h = qy - py;
    double dist = sqrt(w*w + h*h);
    printf("Distance is %f\n", dist);
    return 0;
}
```

Call to `printf` displays  
following line:

Distance is 5.000000

# printf: Formatted Output (2/7)

12

function to print to  
standard output

② function arguments

① `printf("Distance is %f\n", dist);`

③ format string

④  
print list

⑤

- 1) *Format specifier or conversion specifier* controls how output is printed to standard output
- 2) Literal characters are printed as is
- 3) Character following `%` is abbreviation for type of data it represents and **must match** with corresponding argument
- 4) `%f` means print floating-point value using fixed-point notation

# printf: Formatted Output (3/7)

13

## Floating-Point Conversion Specifiers for printf

Conversion specifier	Description
<b>f</b> or <b>F</b>	Display floating-point value using <i>fixed-point notation</i>
<b>e</b> or <b>E</b>	Display floating-point value using <i>exponential notation</i>
<b>g</b> or <b>G</b>	Display floating-point value using either <i>fixed-point</i> or <i>exponential notation</i> depending on value's magnitude
<b>L</b>	Place before any floating-point format specifier to display <b>long double</b> value

# printf: Formatted Output (4/7)

14

print 6 digits after decimal point: 1234567.890000

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

```
int main(void) {
```

```
    double d = 1234567.89;
```

```
    printf("Printing double value using %f modifier: %f\n", d);
```

```
    printf("Printing double value using %e modifier: %e\n", d);
```

```
    printf("Printing double value using %E modifier: %E\n", d);
```

```
    printf("Printing double value using %g modifier: %g\n", d);
```

```
    printf("Printing double value using %G modifier: %G\n", d);
```

```
    return 0;
```

```
}
```

print exponential form: 1.234568e+06

print % character

Exponential form: 1.234568E+06

print value using %f or %e specifier, depending on value's size

# printf: Formatted Output (5/7)

15

```
#include <math.h>
#include <stdio.h>

int main(void) {
    double px = 0.0, py = 0.0;
    double qx = 3.0, qy = 4.0;
    double w = qx - px, h = qy - py;
    double dist = sqrt(w*w + h*h);
    printf("Distance between (%f, %f) and (%f, %f) is %f\n",
           px, py, qx, qy, dist);
    return 0;
}
```



Distance between (0.000000, 0.000000) and (3.000000, 4.000000) is 5.000000

# printf: Formatted Output (6/7)

16

```
#include <stdio.h>

int abs(int num) {
    if (num < 0) {
        num = -num;
    }
    return num;
}

int main(void) {
    int x = -10;
    printf("Abs(%d) is %i\n", x, abs(x));
    return 0;
}
```

means print *integer*  
value of **short** or **int**

means print *decimal* value of **short** or **int**



# printf: Formatted Output (7/7)

17

## Integer Conversion Specifiers for printf

Conversion specifier	Description
<b>d</b>	Display as <i>signed integer</i>
<b>i</b>	Display as <i>signed integer</i>
<b>u</b>	Display as <i>unsigned integer</i>
<b>o</b>	Display as <i>unsigned octal integer</i>
<b>x</b> or <b>X</b>	Display as <i>unsigned hexadecimal integer</i> with hexadecimal digits printed as <b>a-f</b> or printed as <b>A-F</b>
<b>h</b> or <b>l</b> or <b>ll</b>	<b>Length modifiers</b> – place <b>before</b> any integer conversion specifier to display <b>short int</b> or <b>long int</b> or <b>long long int</b> values

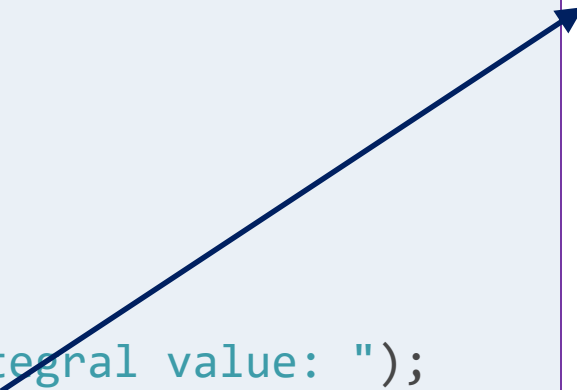
# scanf: Formatted Input (1 / 6)

18

```
#include <stdio.h>

int abs(int num) {
    if (num < 0) {
        num = -num;
    }
    return num;
}

int main(void) {
    int x;
    printf("Enter integral value: ");
    scanf("%d", &x);
    printf("Abs(%d) is %i\n", x, abs(x));
    return 0;
}
```

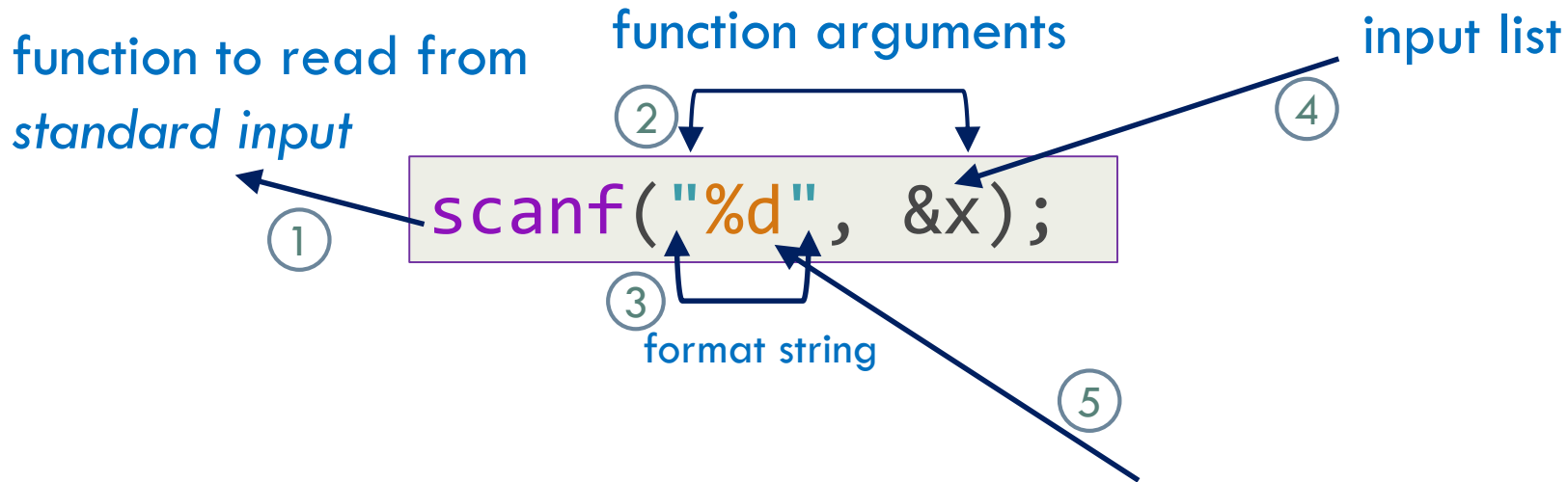


`scanf` function allows you to enter values from standard input during program execution

- 1) Can input all types of data
- 2) Input specific characters from standard input
- 3) Skip specific characters from standard input

# scanf: Formatted Input (2/6)

19



- 1) Format specifier or conversion specifier controls how input is read from standard input
- 2) Character following % is abbreviation for type of data it represents and **must match** with corresponding argument in input list
- 3) %d means read value of type **int**

# scanf: Formatted Input (3/6)

20

```
int x = 10;  
scanf("%d", &x);
```

& is address-of-operator

1000	10	x
1001		
1002		
1003		

- 1) Every variable has *two values* associated with it
- 2) 1<sup>st</sup> value is memory address at which variable is given storage
- 3) 2<sup>nd</sup> value is actual value stored at that memory location
- 4) Variable **x** is given birth at some memory location with **int** value **10** stored at that location
- 5) Expression **&x** evaluates to value that is address of memory location where **x** is given storage and is of type *pointer to int*
- 6) Function **scanf** needs this address to place integer read from standard input into memory location where **x** is given storage

# scanf: Formatted Input (4/6)

21

## Integer Conversion Specifiers for scanf

Conversion specifier	Specifier
<b>d</b>	Read an optionally <i>signed decimal integer</i> . The corresponding argument is pointer to an <b>int</b> .
<b>i</b>	Read an optionally signed decimal, octal, or hexadecimal integer. The corresponding argument is pointer to an <b>int</b> .
<b>o</b>	Read an <i>octal integer</i> . The corresponding argument is pointer to an <b>unsigned int</b> .
<b>u</b>	Read an <i>unsigned decimal integer</i> . The corresponding argument is pointer to an <b>unsigned int</b> .
<b>x</b> or <b>X</b>	Read a <i>hexadecimal integer</i> . The corresponding argument is pointer to an <b>unsigned int</b> .
<b>h</b> or <b>l</b> or <b>ll</b>	Place before any integer format specifier to indicate <b>short int</b> or <b>long int</b> or <b>long long int</b> value to be input

# scanf: Formatted Input (5/6)

22

## Floating-Point Conversion Specifiers for scanf

Conversion specifier	Specifier
<b>f</b> or <b>e</b> or <b>E</b> or <b>g</b> or <b>G</b>	Read a <i>floating-point</i> value. The corresponding argument is pointer to a floating-point variable.
<b>l</b> or <b>L</b>	Place before any of above floating-point conversion specifiers to indicate that a <b>double</b> or <b>long double</b> value is to be input. The corresponding argument is a pointer to a <b>double</b> or <b>long double</b> variable.

# scanf: Formatted Input (6/6)

23

```
double distance(double px, double py, double qx, double qy) {
    double w = qx - px;
    double h = qy - py;
    return sqrt(w*w + h*h);
}

int main(void) {
    printf("Enter coordinates of point P: ");
    double px, py;
    scanf("%lf %lf", &px, &py);
    printf("Enter coordinates of point Q: ");
    double qx, qy;
    scanf("%lf %lf", &qx, &qy);

    printf("Distance between (%f, %f) and (%f, %f) is %f\n",
           px, py, qx, qy, distance(px, py, qx, qy));
    return 0;
}
```

# Summary

24

- `printf` is C standard library function for writing characters to standard output stream
- `scanf` is C standard library function for reading characters to standard input stream
- Don't memorize *anything* related to these functions
  - ▣ Too many conversion specifiers – the most basic ones become part of every programmer's vocabulary
  - ▣ There's more to know about `printf` for printing tables and other formatted data
  - ▣ For more information: Use your text book and bookmark [this page](#) for `printf` and [this page](#) for `scanf`