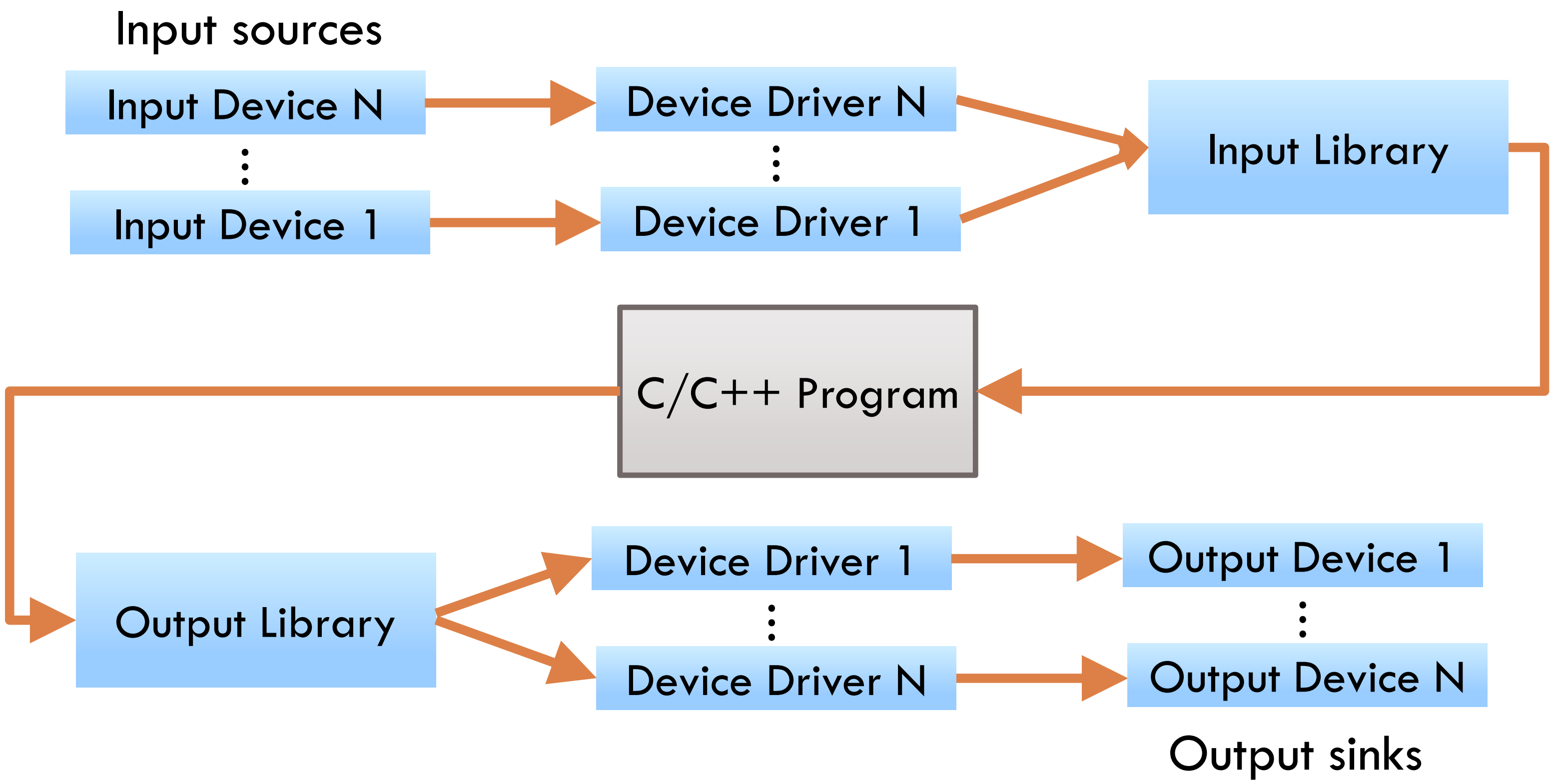
HIGH-LEVEL PROGRAMMING I

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

Outline

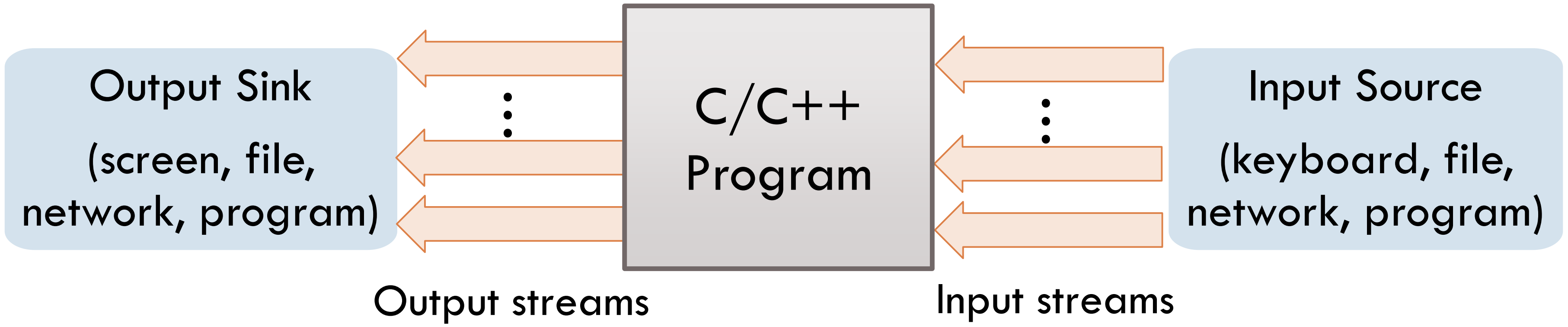
- Writing values to standard output stream
- Reading values from standard input stream

Input and Output



Stream Model

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


Output Sink
(screen, file, network, program)
C/C++ Program
Input Source
(keyboard, file, network, program)
Output streams
Input streams

Header file

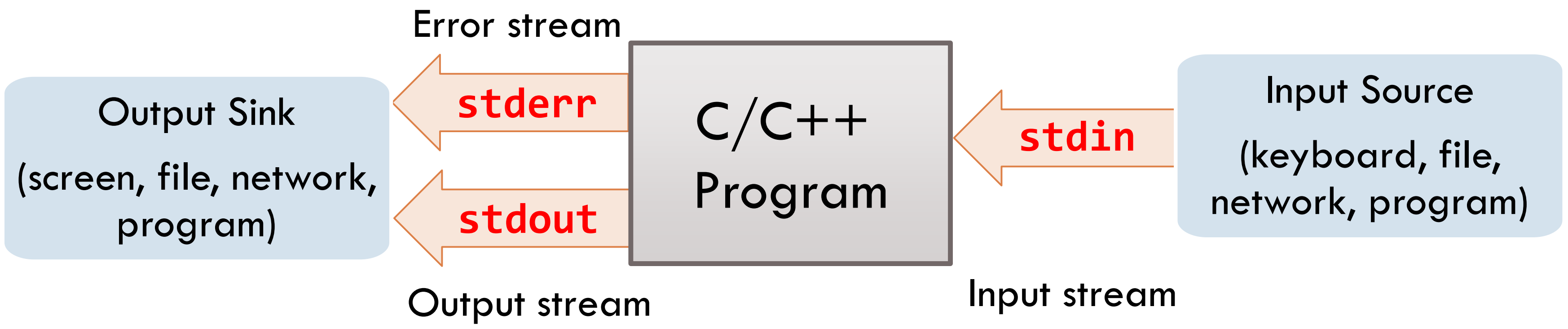
- `<stdio.h>` must be included to access C input/output functions
- All these functions are compatible with 7-bit ASCII byte and UTF-8 encoding

File Pointers

- 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)

- 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)

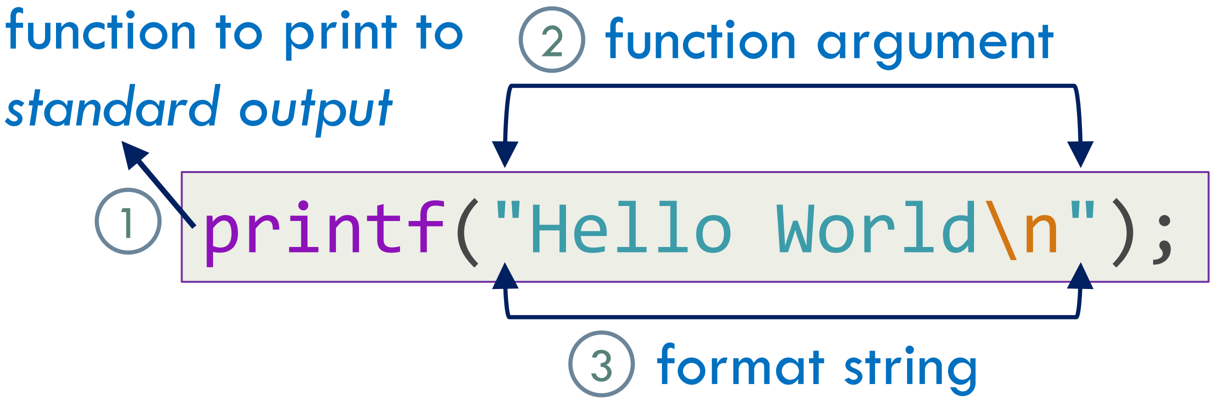
File pointer	Stream	Default meaning
`stdin`	standard input	Keyboard
`stdout`	standard output	Screen
`stderr`	standard error	Screen

printf: String Output

```c
#include <stdio.h>

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

Call to printf displays following line:

```
Hello World
```



Literal characters in format string are printed as is

Escape Sequences

- 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
`\a`	`alert (bell) character`
`\b`	`backspace`
`\n`	`newline`
`\t`	`horizontal tab`
`\\`	`backslash`
`\'`	`single quote`
`\"`	`double quote`

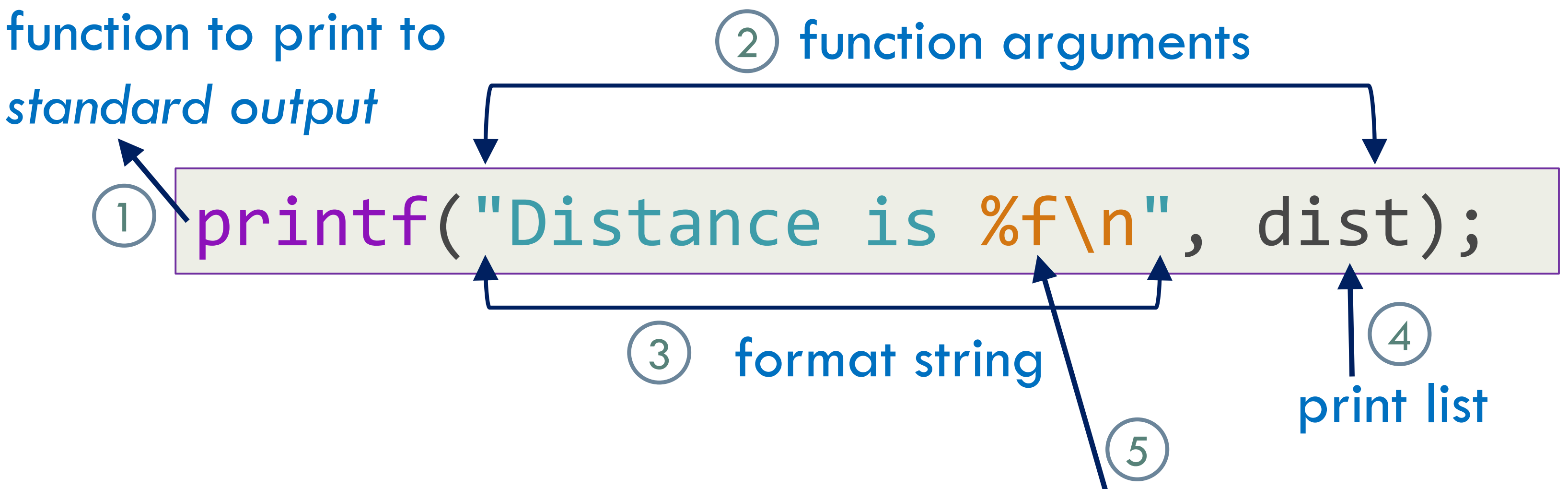
printf: Formatted Output (1/7)

```c
#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)



```
printf("Distance is %f\n", dist);
```

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)

Floating-Point Conversion Specifiers for `printf`

Conversion specifier	**Description**
`f` or `F`	Display floating-point value using *fixed-point notation*
`e` or `E`	Display floating-point value using *exponential notation*
`g` or `G`	Display floating-point value using either *fixed-point* or *exponential notation* depending on value’s magnitude
`L`	Place before any floating-point format specifier to display `long double` value

printf: Formatted Output (4/7)

print 6 digits after decimal point: 1234567.890000

```c
#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)

```c
#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)

```
#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)

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

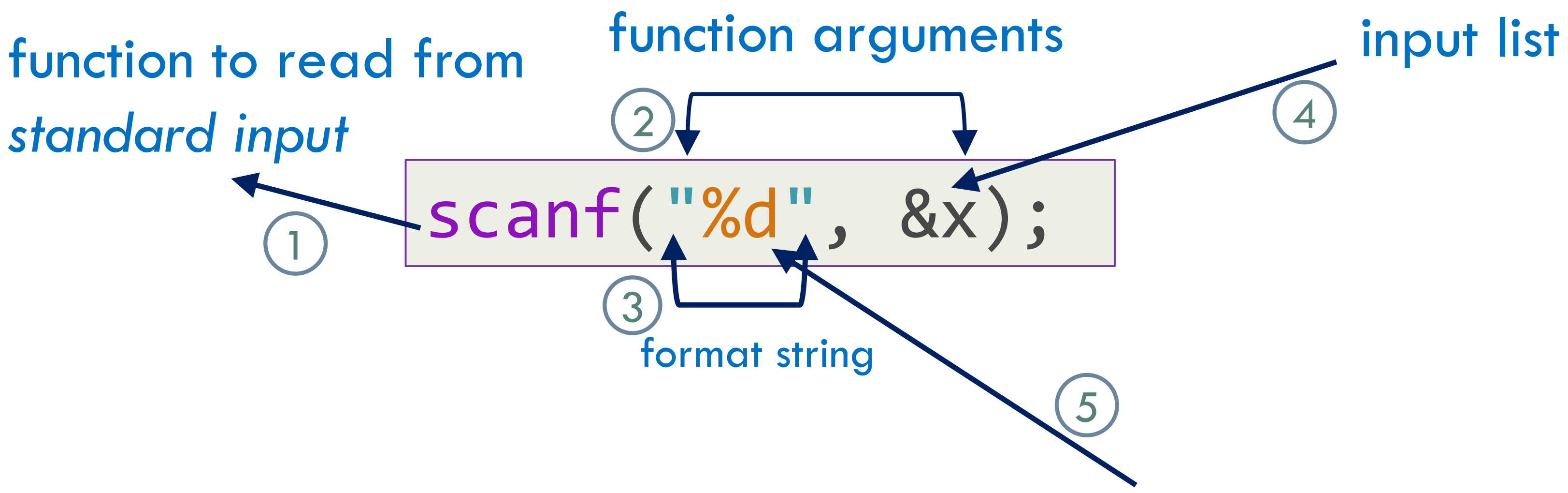
scanf: Formatted Input (1/6)

```c
#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)



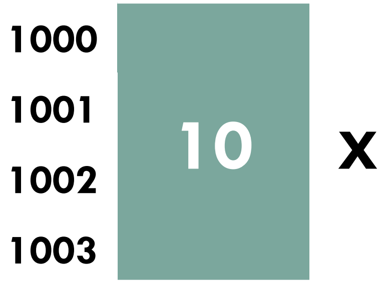
```c
scanf("%d", &x);
```

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)

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

& is address-of-operator



1) Every variable has *two values* associated with it
2) 1st value is memory address at which variable is given storage
3) 2nd 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)

Integer Conversion Specifiers for scanf

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

scanf: Formatted Input (5/6)

Floating-Point Conversion Specifiers for `scanf`	
Conversion specifier	**Specifier**
`f` or `e` or `E` or `g` or `G`	Read a *floating-point value*. The corresponding argument is pointer to a floating-point variable.
`l` or `L`	Place before any of above floating-point conversion specifiers to indicate that a `double` or `long double` value is to be input. The corresponding argument is a pointer to a `double` or `long double` variable.

scanf: Formatted Input (6/6)

```c
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

- `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`