

Programming in C/C++: A Hands-on Introduction

Teaching Assistant

Boston University

Feb. 1st, 2025

Department of Electrical and Computer Engineering
Boston University



Coding Principle 1:

Practice makes perfect.

Coding Principle 2:

Always follow the standards.

Reference: <https://google.github.io/styleguide/>

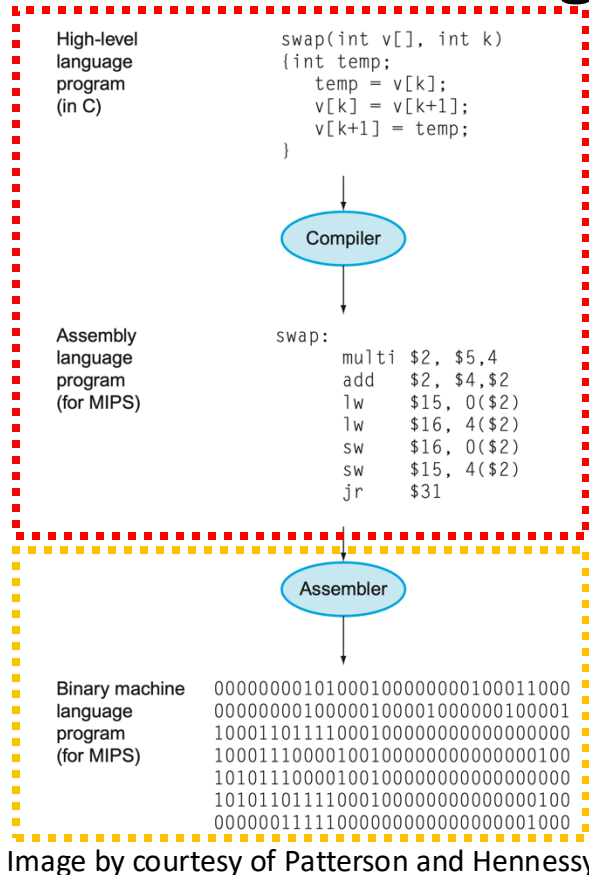
Coding Principle 3:

The beauty is in simplicity.

Outline

- The Bigger Picture
- Types of C files
- C Program Structure
- Syntax of C
- C++: An Extension to the C Language
- Resources

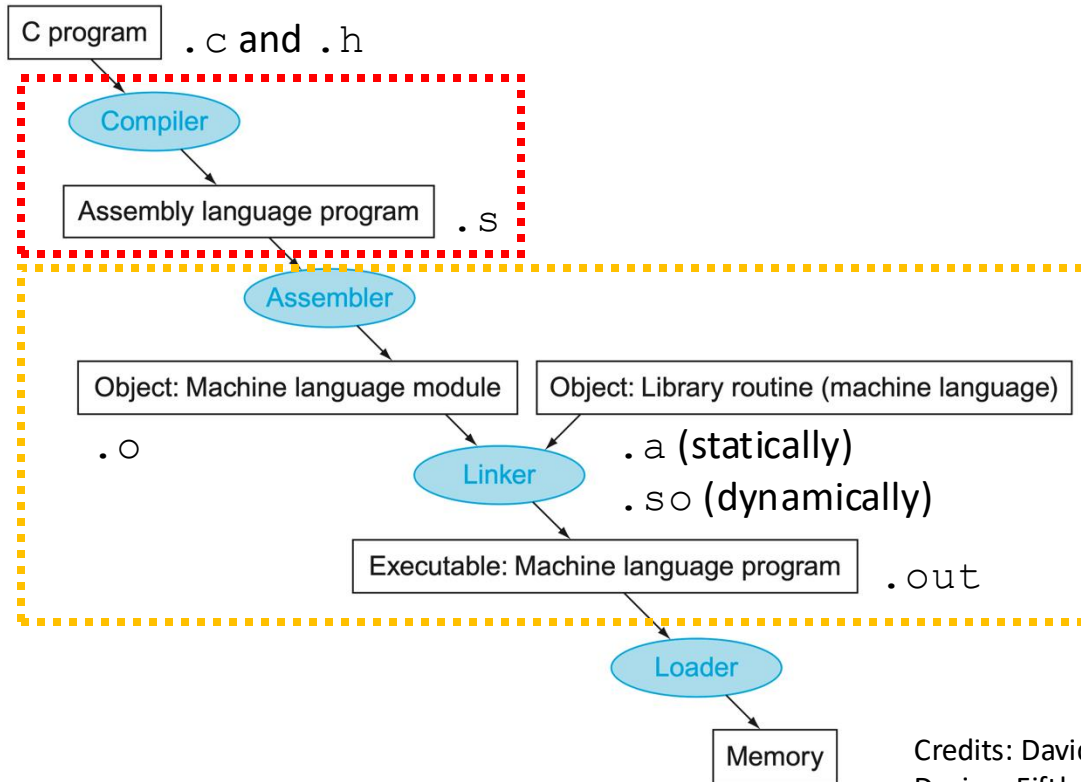
Part 1 – The Bigger Picture



- **C program:** compiled into **assembly language** and then assembled into **binary machine code**
- **Compiler:** The compiler transforms the C program into an assembly language program, a **symbolic form** of what the machine understands.
- **Assembler:** The assembler turns the assembly language program into an **object file (symbol table)**, a combination of machine language instructions, data, and information needed to place instructions properly in memory.

Credits: David Patterson and John Hennessy, Computer Organization and Design, Fifth Edition: The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Part 1 – The Bigger Picture



- **Linker:** combines independently assembled machine language programs and resolves all undefined labels into an **executable file (binary machine code)**.
- **Loader:** A systems program that places an object program in the main memory so that it is ready to execute.
- Detect changes at the **file level**, only recompiling the modified .c file.

Credits: David Patterson and John Hennessy, Computer Organization and Design, Fifth Edition: The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Part 2 – Types of C files

myheader.h

```

1  #ifndef MYHEADER_H
2  #define MYHEADER_H
3
4  void sayHello();
5
6  #endif

```

Name of your header file

Your own defined function statement

main.c

```

1  #include <stdio.h>
2  #include "myheader.h"
3
4  void sayHello() {
5      printf("Hello, World!\n");
6  }
7
8  int main() {
9      sayHello();
10     return 0;
11 }

```

Details of your own defined function

The main function

- **myheader.h: Header file**, containing **function prototypes** and various **pre-processor statements**. They are used to allow source code files to access externally-defined functions.
- **main.c: Source file**, containing **function definitions** and the **entire program logic**.
- **#include**: request to use a header file in our program

C Read Files

C Structures

C Structures

C Enums

C Enums

C Memory

C Memory Management ▾

C Reference

C Reference

C Keywords

C <stdio.h>

C <stdlib.h>

C <string.h>

C <math.h>

C <ctype.h>

C Examples

C Examples

C Real-Life Examples

C Exercises

C Quiz

C Compiler

C Syllabus

C Study Plan

C Certificate

C stdio (stdio.h) Library

[< Previous](#)[Next >](#)

C stdio Functions

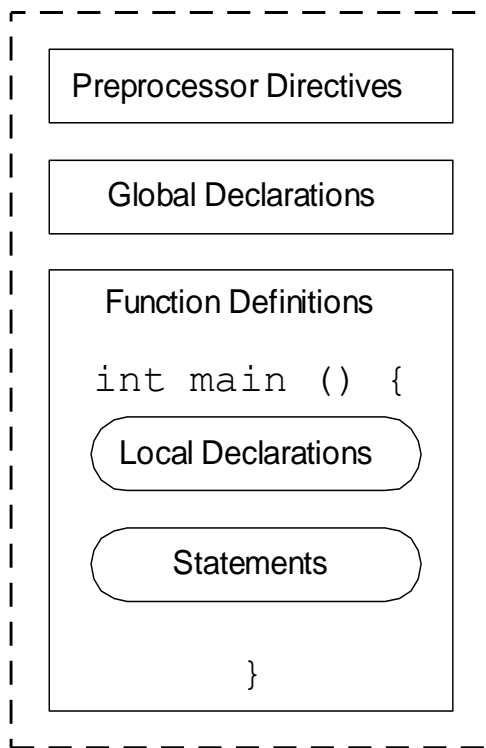
Credits: https://www.w3schools.com/c/c_ref_stdio.php

The `<stdio.h>` header provides a variety of functions for input, output and file handling.

A list of all stdio functions can be found in the table below:

Function	Description
<code>fclose()</code>	Closes a file
<code>feof()</code>	Returns a true value when the position indicator has reached the end of the file
<code>ferror()</code>	Returns a true value if a recent file operation had an error
<code>fgetc()</code>	Returns the ASCII value of a character in a file and advances the position indicator
<code>fgets()</code>	Reads a line from a file and advances the position indicator
<code>fopen()</code>	Opens a file and returns a file pointer for use in file handling functions
<code>fprintf()</code>	Writes a formatted string into a file
<code>fputc()</code>	Writes a character into a file and advances the position indicator
<code>fputs()</code>	Writes a string into a file and advances the position indicator
<code>fread()</code>	Reads data from a file and writes it into a block of memory

Part 3 – C Program Structure



- **Program** defined by:
 - global declarations
 - function definitions
- May contain **preprocessor directives**
- **Always has one function** named `main`, and may contain others

Part 3 – C Program Structure

```
1  #include <stdio.h>
2
3  int x; // Global variable
4
5  int main() {
6      int y; // Local variable
7
8      printf("Enter x and y: ");
9
10     // Input x and y through your keyboard
11     scanf("%d %d", &x, &y);
12
13     // Print the sum value
14     printf("Sum is %d\n", x + y);
15
16     return 0; // Added return statement for proper termination
17 }
18
```

Preprocessor Directive

Global Declaration

Local Declaration

Statements

Part 3 – C Program Structure – PD

Preprocessor Directives:

- Begin with #
- Instruct the compiler to perform some transformation to the file before compiling

- Example: `#include <stdio.h>`
 - add the **header** file `stdio.h` to this file
 - `.h` for header file
 - `stdio.h` defines useful input/output functions

Preprocessor Directive

```
1  #include <stdio.h>
2
3  int x; // Global variable
4
5  int main() {
6      int y; // Local variable
7
8      printf("Enter x and y: ");
9
10     // Input x and y through your keyboard
11     scanf("%d %d", &x, &y);
12
13     // Print the sum value
14     printf("Sum is %d\n", x + y);
15
16     return 0; // Added return statement for proper termination
17 }
18
```

Part 3 – C Program Structure – Declarations

Declarations:

➤ Global

- visible throughout program
- describes data used throughout program

➤ Local

- visible within function
- describes data used only in function

```
1  #include <stdio.h>
2
3  int x; // Global variable
4
5  int main() {
6      int y; // Local variable
7
8      printf("Enter x and y: ");
9
10     // Input x and y through your keyboard
11     scanf("%d %d", &x, &y);
12
13     // Print the sum value
14     printf("Sum is %d\n", x + y);
15
16     return 0; // Added return statement for proper termination
17 }
18
```

Global Declaration

Local Declaration

Part 3 – C Program Structure – Functions

Functions and Program Extraction:

- Consists of **header** and **body**
 - header: `int main()`
 - body: contained between { and }
 - starts with **location declarations**
 - followed by a series of **statements**
- More than one function may be defined
- **Every program has one function `main` and `main` is executed**
- **Procedural programming**: executed in order and **defined functions** are called (invoked)

```
1  #include <stdio.h>
2  #include "myheader.h"
3
4  void sayHello() {
5      printf("Hello, World!\n");
6  }
7
8  int main() {
9      sayHello();
10     return 0;
11 }
```

Part 3 – C Program Structure – Comments

Comments:

- **Single line:** `//`
- **Multiple lines:** Text between `/*` and `*/`
- Used to “document” the code for the human reader
- Ignored by the compiler (not part of the program)
- Have to be careful
 - comments may cover multiple lines
 - ends as soon as `*/` encountered
 - so no internal comments: `/* An internal comment */`

Part 3 – C Program Structure – Comments

```
1  #include <stdio.h>
2
3  /* This comment covers
4     multiple lines
5     in the program.
6     */
7
8  int main() {
9      // No local declarations
10
11     printf("Too many comments\n");
12     return 0; // Added return statement for proper program termination
13 } // end of main
14
```

Comment multiple lines

Comment one line

Part 3 – C Program Structure – Comments

Comments:

- **Global**: start of program, outlines overall solution, may include structure chart
- **Module**: when using separate files, an indication of what each file solves
- **Function**: inputs, return values, and logic used in defining function
- Add documentation for **key (tough to understand)** comments
- **Names of variables**: should be chosen to be meaningful, make program readable

Part 4 – Syntax of C – Basics

```
1  #include <stdio.h>
2
3  int x; // Global variable
4
5  int main() {
6      int y; // Local variable
7
8      printf("Enter x and y: ");
9
10     // Input x and y through your keyboard
11     scanf("%d %d", &x, &y);
12
13     // Print the sum value
14     printf("Sum is %d\n", x + y);
15
16     return 0; // Added return statement for proper termination
17 }
18
```

- Rules that define C language
 - **Specify which tokens are valid**
 - Also indicate the **expected order of tokens**
- Some types of tokens:
 - reserved words: include `printf`
 - identifiers: `x`, `y`
 - literal constants: `5`, `'a'`, `5.0`
 - punctuation: `{` `}` `;` `<` `>` `#` `/*` `*/`

Part 4 – Syntax of C – Naming Rules

```
1  #include <stdio.h>
2
3  int x; // Global variable
4
5  int main() {
6      int y; // Local variable
7
8      printf("Enter x and y: ");
9
10     // Input x and y through your keyboard
11     scanf("%d %d", &x, &y);
12
13     // Print the sum value
14     printf("Sum is %d\n", x + y);
15
16     return 0; // Added return statement for proper termination
17 }
18
```

- Names used for objects in C
- Rules for identifiers in C
 - first char alphabetic [a-z, A-Z]
or underscore (_)
 - has only alphabetic, digit, underscore chars
 - cannot duplicate a reserved word
 - case (upper/lower) matters

Part 4 – Syntax of C – Naming Rules

Valid

sum

c4_5

A_NUMBER

timeofflight

TRUE

_split_name

Invalid

7of9

x-name

name with spaces

1234a

int

AXYZ&

Part 4 – Syntax of C – Variables

- Variables declared in global or local declaration sections
- Syntax: *Type Name*;
- Examples:

```
int sum;
```

```
float avg;
```

```
char dummy;
```

Part 4 – Syntax of C – Variables

- Indicates **how much memory** to set aside for the variable
- Also determines how that space will be interpreted
- Basic types: `char`, `int`, `float`, `double`, `bool`
 - specify the amount of space (bytes) to set aside
 - what can be stored in that space
 - what operations can be performed on those vars

Part 4 – Syntax of C – Variables

➤ Can create **multiple variables** of the same type in one statement:

```
int x, y, z;
```

is a shorthand for

```
int x;
```

```
int y;
```

```
int z;
```

- stylistically, the latter is often preferable

Part 4 – Syntax of C – Variables

- Giving a variable an **initial value**
- Variables **not necessarily initialized** when declared
- Can initialize in the declaration:
- Syntax: *Type Name = Value;*
- Example:

```
int x = 0;
```

```
int x, y, z = 0;
```


Part 4 – Syntax of C – `void`

- Type name: `void`
- Possible values: none
- Operations: none
- Useful as a **placeholder**
- Meaning: No value is present. It does not provide a result value to its caller. It has no values and no operations. It is used to represent nothing.

Part 4 – Syntax of C – Integers

Type name: `int`, `short int`, `long int`

<u>Type</u>	<u>Bytes</u>	<u>Bits</u>	<u>Min Val</u>	<u>Max Val</u>
<code>short int</code>	2	16	-32768	32767
<code>int</code>	4	32	-2147483648	2147483647
<code>long int</code>	4	32	-2147483648	2147483647

Note: `long long int` and `unsigned long long int`: 8 bytes

Credits: <https://www.geeksforgeeks.org/data-types-in-c/>

Part 4 – Syntax of C – Unsigned Integers

- Type name: `unsigned int`
- No negative values
- `unsigned int`
 - possible values: 0 to 65536
- Representation: binary number

Part 4 – Syntax of C – Floating Points

- `float`: 4 bytes, 32 bits
- `double`: 8 bytes, 64 bits
- `long double`: 10 bytes, 80 bits
- Representation:
 - magnitude (some number of bits) plus exponent (remainder of bits)
 - 3.26×10^4 for 32600.0

Part 4 – Syntax of C – Characters

- Type name: `char`
- Possible values: keys that can be typed at the keyboard
- Representation: each character assigned a value (8-bit ASCII)
 - A - binary number 65
 - a - binary number 97
 - b - binary number 98
 - 2 - binary number 50

Part 4 – Syntax of C – Characters

➤ Single key stroke between quote char ‘

➤ Examples: ‘A’, ‘a’, ‘b’, ‘1’, ‘@’

➤ Some special chars:

- \0 : null char
- \t : tab char
- \n : newline char
- \ : single quote char
- \\ : backslash char

Part 4 – Syntax of C – Constants

- Literal constants - tokens representing values from type
- Defined constants
 - syntax: `#define Name Value`
 - preprocessor command, *Name* replaced by *Value* in program
 - example: `#define MAX_NUMBER 100`

Part 4 – Syntax of C – Constants

- Memory constants
 - declared similar to variables, type, and name
 - `const` added before the declaration
 - Example: `const float PI = 3.14159;`
 - Can be used as a variable, but **one that cannot be changed**
 - Since the value cannot be changed, **it must be initialized**

Part 4 – Syntax of C – Format Strings

- Format string may contain one or more field specifications
 - **Syntax:** `%Code` or `%[Width].[Precision]Code`
 - **Code:**
 - `c` - data printed as **character**
 - `i` or `d` - data printed as **integer**
 - `f` - data printed as **floating-point** value
 - For each field specification, have one data value after the format string, separated by commas

Part 4 – Syntax of C – Format Strings

```
printf("%c %d %f\n", 'A', 35, 4.5);
```

produces

A 35 4.50000

Can have variables in place of literal constants
(value of variable printed)

Part 4 – Syntax of C – Format Strings

- When printing numbers, generally use `Width/Precision` to determine format
 - `Width`: how many character spaces to use in printing the field (minimum, if more needed, more used)
 - `Precision`: for floating point numbers, **how many characters appear after the decimal point**, width counts decimal point, number of digits after decimal, remainder before decimal

Part 4 – Syntax of C – Format Strings

```
printf("%5d%8.3f\n", 753, 4.1678);
```

produces

```
753 4.168
```

values are right justified (aligned)

If not enough characters in width, minimum number used

use 1 width to indicate minimum number of chars should be used

Part 4 – Syntax of C – Repetition Control

```
for (initialize expression; test expression; update expression)
```

```
{
```

```
    //
```

```
    // body of for loop
```

```
    //
```

```
}
```

```
1  #include <stdio.h> // Include the standard input-output library
2
3  // Driver code
4  int main()
5  {
6      int i = 0; // Declare an integer variable i, initialized to 0
7
8      // For loop to print "Hello World" 10 times
9      for (i = 1; i <= 10; i++) // Loop starts at 1 and continues while i is less than or equal to 10
10     {
11         printf("Hello World\n"); // Print "Hello World" followed by a newline
12     }
13
14     return 0; // Return 0, indicating that the program finished successfully
15 }
16
```

Part 4 – Syntax of C – Repetition Control

```
while (test_expression)
{
    // body of the while loop
    update_expression;
}
```

```
1  #include <stdio.h> // Include standard input-output library
2
3  // Driver code
4  int main()
5  {
6      // Initialization expression
7      int i = 1; // Variable i is initialized to 1
8
9      // Test expression
10     while(i <= 10) // Loop will run as long as i is less than or equal to 10
11     {
12         // Loop body
13         printf("Hello World\n"); // Print "Hello World" followed by a newline
14
15         // Update expression
16         i++; // Increment i by 1
17     }
18
19     return 0; // Return 0, indicating successful completion of the program
20 }
21
```

Part 4 – Syntax of C – Conditions

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
}  
else if (condition2) {  
    // block of code to be executed if the condition1 is false and  
    condition2 is true  
}  
else {  
    // block of code to be executed if the condition1 is false and  
    condition2 is false  
}
```

Part 4 – Syntax of C – Conditions

```
1  #include <stdio.h>
2
3  int main() {
4      int time = 22;  // The variable 'time' is initialized to 22 (represents 10 PM)
5
6      // Conditional check based on the value of 'time'
7      if (time < 10) {  // If time is less than 10 (morning)
8          printf("Good morning.");
9      } else if (time < 20) {  // If time is between 10 (inclusive) and 20 (exclusive), i.e., day
10         printf("Good day.");
11     } else {  // If time is 20 or more (evening)
12         printf("Good evening.");
13     }
14
15     return 0;
16 }
17
```


Part 4 – Syntax of C – Conditions

```
switch (expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

```
1  #include <stdio.h>
2
3  int main() {
4      int day = 4; // The variable 'day' is initialized to 4 (represents Thursday)
5
6      // Switch statement to handle different days of the week
7      switch (day) {
8          case 1: // If 'day' is 1
9              printf("Monday");
10             break;
11          case 2: // If 'day' is 2
12              printf("Tuesday");
13              break;
14          case 3: // If 'day' is 3
15              printf("Wednesday");
16              break;
17          case 4: // If 'day' is 4
18              printf("Thursday");
19              break;
20          case 5: // If 'day' is 5
21              printf("Friday");
22              break;
23          case 6: // If 'day' is 6
24              printf("Saturday");
25              break;
26          case 7: // If 'day' is 7
27              printf("Sunday");
28              break;
29          default: // If 'day' doesn't match any case (invalid input)
30              printf("Invalid day");
31      }
32
33      return 0;
34  }
```

Part 4 – Syntax of C – Structures

- A way to **group several related variables into one place**
- Each variable in the structure is known as a **member of the structure**

```
1  #include <stdio.h>
2
3  // Define a structure named myStructure
4  struct myStructure {
5      int myNum;    // An integer variable
6      char myLetter; // A character variable
7  };
8
9  int main() {
10     // Declare a variable 's1' of type 'struct myStructure'
11     struct myStructure s1;
12
13     return 0;
14 }
15
```

```
1  #include <stdio.h>
2
3  // Define a structure named myStructure
4  struct myStructure {
5      int myNum;    // Integer variable
6      char myLetter; // Character variable
7  };
8
9  int main() {
10     // Declare a variable 's1' of type 'struct myStructure'
11     struct myStructure s1;
12
13     // Assign values to structure members
14     s1.myNum = 13;
15     s1.myLetter = 'B';
16
17     // Print structure members
18     printf("My number: %d\n", s1.myNum);
19     printf("My letter: %c\n", s1.myLetter);
20
21     return 0;
22 }
23
```

Optional – Syntax of C – Pointers

Creating Pointers: &

- A variable that stores the **memory address** of another variable as its value.

```
1  #include <stdio.h>
2
3  int main() {
4      int myAge = 43;      // Declare an integer variable with value 43
5      int* ptr = &myAge;  // Declare a pointer 'ptr' and assign it the address of 'myAge'
6
7      // Output the value of myAge (should print: 43)
8      printf("%d\n", myAge);
9
10     // Output the memory address of myAge using the address-of operator (&)
11     printf("%p\n", &myAge);
12
13     // Output the memory address of myAge stored in the pointer 'ptr'
14     printf("%p\n", ptr);
15
16     return 0;
17 }
18
```

Optional – Syntax of C – Pointers

Dereference Pointers: *

- A variable that stores the memory address of another variable as its value.

```
1  #include <stdio.h>
2
3  int main() {
4      int myAge = 43;      // Declare an integer variable and initialize it to 43
5      int* ptr = &myAge;  // Declare a pointer to an int and store the address of myAge
6
7      // Reference: Print the memory address of myAge using the pointer
8      printf("%p\n", ptr);
9
10     // Dereference: Print the value of myAge using the pointer
11     printf("%d\n", *ptr);
12
13     return 0;
14 }
15
```

To learn more about C:

<https://www.w3schools.com/c/index.php>

Part 5 – C++: An Extension to the C Language

```
1  #include <iostream>
2  #include <string>
3
4  int main() {
5      std::string name;
6      std::cout << "What is your name?" << std::endl;
7      std::cin >> name;
8      std::cout << "Hello " << name << "!" << std::endl;
9      return 0;
10 }
11
```

`std::`

is a namespace that contains the **standard library** components, such as **Data types**, **Functions**, and **Objects**.

Part 5 – C++: An Extension to the C Language

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  ✓ int main() {
7      string name;
8      cout << "What is your name?" << endl;
9      cin >> name;
10     cout << "Hello " << name << "!" << endl;
11     return 0;
12 }
13
```

Widely used in practice!

Part 5 – C++: An Extension to the C Language

An Example

Welcome to EC327
Introduction to Software Engineering!

Part 5 – C++: An Extension to the C Language

```
1  #ifndef NAME_INTERACTIONS_H
2  #define NAME_INTERACTIONS_H
3
4  #include <iostream>
5  #include <string>
6
7  /**
8   * Prompts the user to input their name via Standard Input and returns the entered name.
9   * @param prompt The prompt message to display for the user.
10  * @return The name entered by the user.
11  */
12  std::string readNameFromStandardIn(std::string prompt);|
13
14  #endif // NAME_INTERACTIONS_H
15
```

Part 5 – C++: An Extension to the C Language

```
1  #include <iostream>
2  #include <string>
3  #include "name_interactions.h"
4
5  ∨ std::string readNameFromStandardIn(const std::string prompt) {
6      std::string name;
7      std::cout << prompt << std::endl; // Use the prompt passed as an argument
8      std::getline(std::cin, name);      // Read the full name, including spaces
9      return name;                      // Return the captured name
10 }
11
12 ∨ int main() {
13     std::string name = readNameFromStandardIn("What is your name?"); // Custom prompt
14     std::cout << "Welcome to EC327\n Introduction to Software Engineering,\n " << name << "!" << std::endl; // Print the greeting
15     return 0;
16 }
17
```

To learn more about C++:

<https://www.w3schools.com/cpp/default.asp>

**If you wanna explore more about
C++, please check its Standards
(widely used in Companies)**

<https://www.geeksforgeeks.org/cpp-11-standard/>

Part 6 – Resources

➤ Coding standards

- Google style guide: <https://google.github.io/styleguide/>

➤ C

- Learn C basics: <https://www.w3schools.com/c/>

➤ C++

- Learn C++ basics: <https://www.w3schools.com/cpp/default.asp>
- Learn C++ standards: <https://www.geeksforgeeks.org/cpp-11-standard/>

Thank you very much for your attention!