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

Shuyue Jia

Ph.D. Student

Boston University

Feb. 1st, 2025

Department of Electrical and Computer Engineering

Boston University

BOSTON UNIVERSITY

Coding Principle 1:

Practice makes perfect.

Department of Electrical and Computer Engineering

Boston University

BOSTON UNIVERSITY

Coding Principle 2:

Always follow the standards.

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

Department of Electrical and Computer Engineering

Boston University

BOSTON UNIVERSITY

Coding Principle 3:

The beauty is in simplicity.

Department of Electrical and Computer Engineering

Boston University

BOSTON UNIVERSITY

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

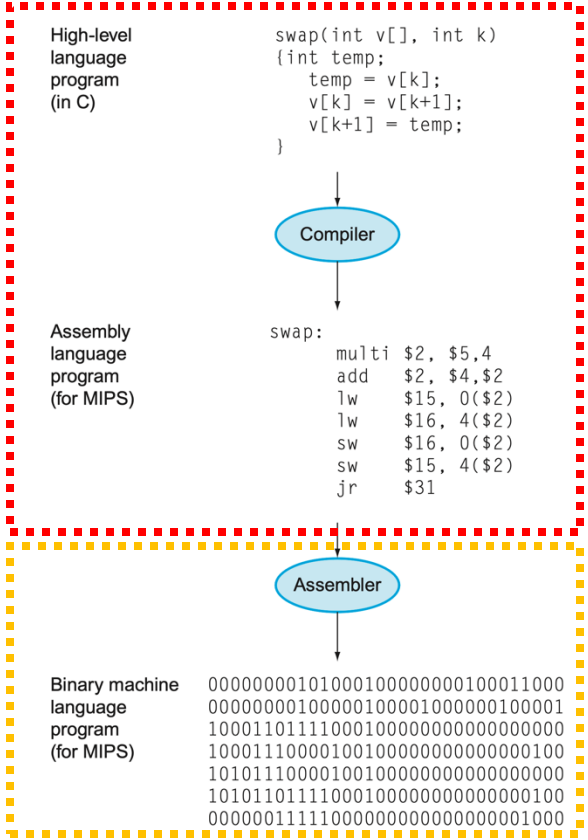


Image by courtesy of Patterson and Hennessy

- **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

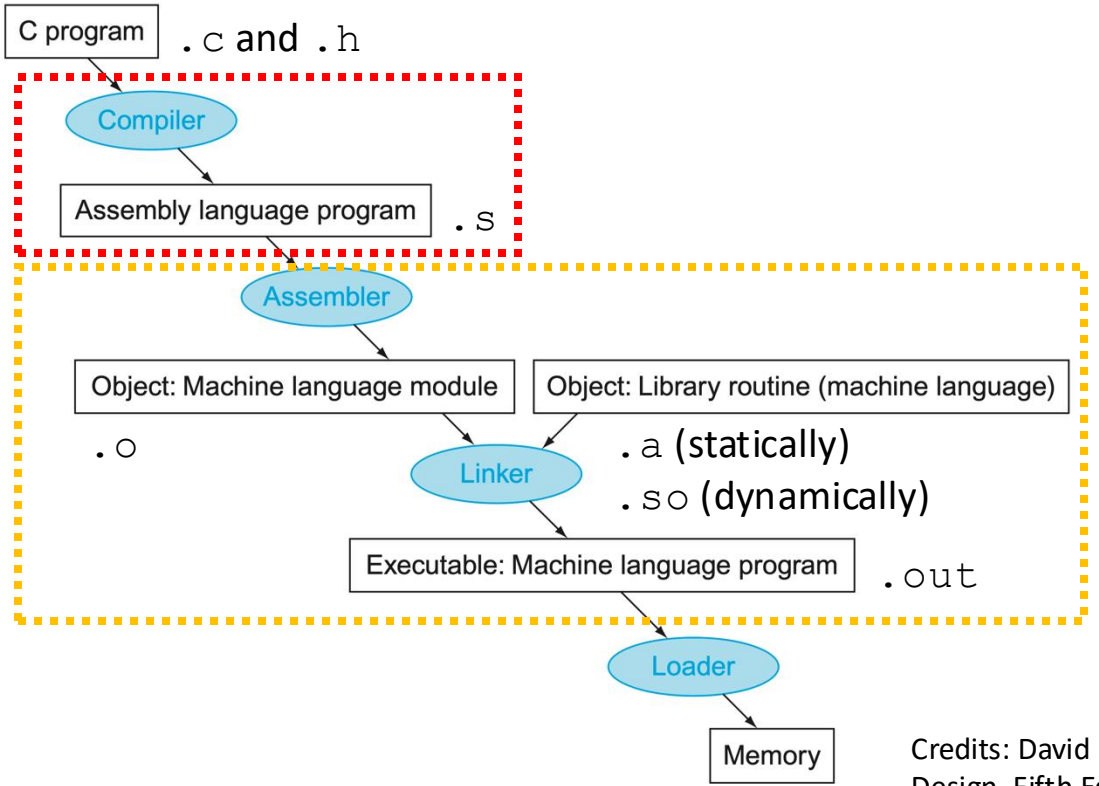


Image by courtesy of Patterson and Hennessy

- **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

Credits: https://www.geeksforgeeks.org/header-files-in-c-cpp-and-its-uses/
https://utat-ss.readthedocs.io/en/master/c-programming/c_h_files.html

C stdio (stdio.h) Library

❮ Previous Next ❯

C stdio Functions

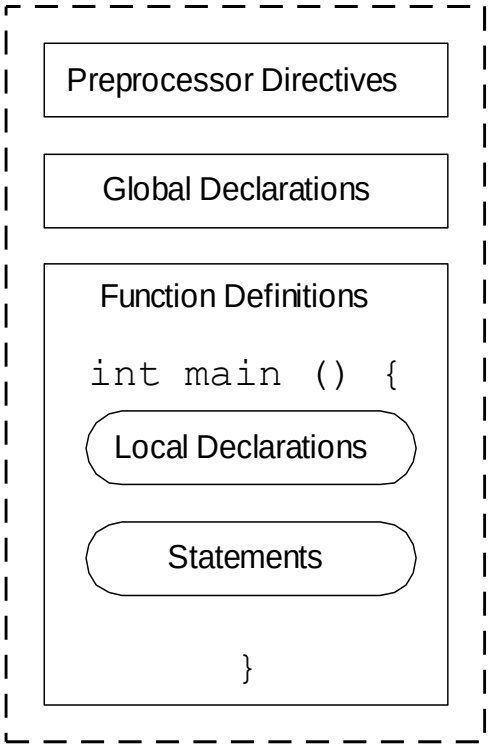
Credits: https://www.w3schools.com/c/c_ref_stdio.php

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

Credits: Rich Maclin, CS 1621: Computer Science I, University of Minnesota-Duluth

Part 3 – C Program Structure

```c
#include <stdio.h>

int x;  // Global variable

int main() {
    int y;  // Local variable

    printf("Enter x and y: ");

    // Input x and y through your keyboard
    scanf("%d %d", &x, &y);

    // Print the sum value
    printf("Sum is %d\n", x + y);

    return 0; // Added return statement for proper termination
}
```

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

```
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

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

```c
#include <stdio.h>

int x;  // Global variable

int main() {
   int y;  // Local variable

    printf("Enter x and y: ");

    // Input x and y through your keyboard
    scanf("%d %d", &x, &y);

    // Print the sum value
    printf("Sum is %d\n", x + y);

    return 0; // Added return statement for proper termination
}
```

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

```c
#include <stdio.h>

int x;  // Global variable

int main() {
    int y;  // Local variable

    printf("Enter x and y: ");

    // Input x and y through your keyboard
    scanf("%d %d", &x, &y);

    // Print the sum value
    printf("Sum is %d\n", x + y);

    return 0; // Added return statement for proper termination
}
```

- 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: `{ } ; < > # /* */`

Credits: https://www.geeksforgeeks.org/c-identifiers/

Part 4 – Syntax of C – Naming Rules

```c
#include <stdio.h>

int x;  // Global variable

int main() {
    int y;  // Local variable

    printf("Enter x and y: ");

    // Input x and y through your keyboard
    scanf("%d %d", &x, &y);

    // Print the sum value
    printf("Sum is %d\n", x + y);

    return 0; // Added return statement for proper termination
}
```

- 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

Credits: https://www.geeksforgeeks.org/c-identifiers/

Part 4 – Syntax of C – Naming Rules

Valid	**Invalid**
sum	7of9
c4_5	x-name
A_NUMBER	name with spaces
timeofflight	1234a
TRUE	int
_split_name	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`

Type	Bytes	Bits	Min Val	Max Val
short int	2	16	-32768	32767
int	4	32	-2147483648	2147483647
long int	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 on the keyboard
- Representation: each character is 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 the 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 Specifiers

- Format string may contain one or more field specifications
  - syntax: `%Code` or `%[Width].[Precision]Code`
  - `Code`:
    - `c` : data printed as **c**haracter
    - `i` or `d` : data printed as **i**nteger
    - `f` : data printed as a **f**loating-point value
  - for each field specification, have one data value after the format string, separated by commas

Credits: https://www.geeksforgeeks.org/format-specifiers-in-c/

Part 4 – Syntax of C – Format Specifiers

```
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)

Credits: https://www.geeksforgeeks.org/format-specifiers-in-c/

Part 4 – Syntax of C – Format Specifiers

- 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

Credits: https://www.geeksforgeeks.org/format-specifiers-in-c/

Part 4 – Syntax of C – Format Specifiers

```
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

Credits: https://www.geeksforgeeks.org/format-specifiers-in-c/

Part 4 – Syntax of C – Repetition Control

```
for (initialize expression; test expression; update expression)
{
    //
    // body of for loop
    //
}
```

```c
#include <stdio.h>  // Include the standard input-output library

// Driver code
int main()
{
  int i = 0;  // Declare an integer variable i, initialized to 0

  // For loop to print "Hello World" 10 times
  for (i = 1; i <= 10; i++)  // Loop starts at 1 and continues while i is less than or equal to 10
  {
    printf("Hello World\n");  // Print "Hello World" followed by a newline
  }

  return 0;  // Return 0, indicating that the program finished successfully
}
```

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

Part 4 – Syntax of C – Repetition Control

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

```c
#include <stdio.h>  // Include standard input-output library

// Driver code
int main()
{
  // Initialization expression
  int i = 1;  // Variable i is initialized to 1

  // Test expression
  while(i <= 10)  // Loop will run as long as i is less than or equal to 10
  {
    // Loop body
    printf("Hello World\n");  // Print "Hello World" followed by a newline

    // Update expression
    i++;  // Increment i by 1
  }

  return 0;  // Return 0, indicating successful completion of the program
}

```

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

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
}
```

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

Part 4 – Syntax of C – Conditions

```c
#include <stdio.h>

int main() {
  int time = 22;  // The variable 'time' is initialized to 22 (represents 10 PM)

  // Conditional check based on the value of 'time'
  if (time < 10) {  // If time is less than 10 (morning)
    printf("Good morning.");
  } else if (time < 20) {  // If time is between 10 (inclusive) and 20 (exclusive), i.e., day
    printf("Good day.");
  } else {  // If time is 20 or more (evening)
    printf("Good evening.");
  }

  return 0;
}
```

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

Part 4 – Syntax of C – Conditions

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

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

```c
#include <stdio.h>

int main() {
  int day = 4;  // The variable 'day' is initialized to 4 (represents Thursday)

  // Switch statement to handle different days of the week
  switch (day) {
    case 1:  // If 'day' is 1
      printf("Monday");
      break;
    case 2:  // If 'day' is 2
      printf("Tuesday");
      break;
    case 3:  // If 'day' is 3
      printf("Wednesday");
      break;
    case 4:  // If 'day' is 4
      printf("Thursday");
      break;
    case 5:  // If 'day' is 5
      printf("Friday");
      break;
    case 6:  // If 'day' is 6
      printf("Saturday");
      break;
    case 7:  // If 'day' is 7
      printf("Sunday");
      break;
    default:  // If 'day' doesn't match any case (invalid input)
      printf("Invalid day");
  }

  return 0;
}
```

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

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**

```c
#include <stdio.h>

// Define a structure named myStructure
struct myStructure {
  int myNum;     // An integer variable
  char myLetter; // A character variable
};

int main() {
  // Declare a variable 's1' of type 'struct myStructure'
  struct myStructure s1;

  return 0;
}
```

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

```c
#include <stdio.h>

// Define a structure named myStructure
struct myStructure {
  int myNum;     // Integer variable
  char myLetter; // Character variable
};

int main() {
  // Declare a variable 's1' of type 'struct myStructure'
  struct myStructure s1;

  // Assign values to structure members
  s1.myNum = 13;
  s1.myLetter = 'B';

  // Print structure members
  printf("My number: %d\n", s1.myNum);
  printf("My letter: %c\n", s1.myLetter);

  return 0;
}
```

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

Optional – Syntax of C – Pointers

Creating Pointers: &

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

```c
#include <stdio.h>

int main() {
  int myAge = 43;      // Declare an integer variable with value 43
  int* ptr = &myAge;  // Declare a pointer 'ptr' and assign it the address of 'myAge'

  // Output the value of myAge (should print: 43)
  printf("%d\n", myAge);

  // Output the memory address of myAge using the address-of operator (&)
  printf("%p\n", &myAge);

  // Output the memory address of myAge stored in the pointer 'ptr'
  printf("%p\n", ptr);

  return 0;
}
```

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

Optional – Syntax of C – Pointers

Dereference Pointers: *

- You can also get **the value of the variable the pointer points to**, by using the * operator

```c
#include <stdio.h>

int main() {
    int myAge = 43;      // Declare an integer variable and initialize it to 43
    int* ptr = &myAge;  // Declare a pointer to an int and store the address of myAge

    // Reference: Print the memory address of myAge using the pointer
    printf("%p\n", ptr);

    // Dereference: Print the value of myAge using the pointer
    printf("%d\n", *ptr);

    return 0;
}
```

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

To learn more about C:

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

Department of Electrical and Computer Engineering
Boston University

BOSTON UNIVERSITY

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

```cpp
#include <iostream>
#include <string>
#include "name_interactions.h"

std::string readNameFromStandardIn(const std::string prompt) {
    std::string name;
    std::cout << prompt << std::endl; // Use the prompt passed as an argument
    std::getline(std::cin, name);     // Read the full name, including spaces
    return name;                      // Return the captured name
}

int main() {
    std::string name = readNameFromStandardIn("What is your name?");  // Custom prompt
    std::cout << "Welcome to EC327\n Introduction to Software Engineering,\n " << name << "!" << std::endl;  // Print the greeting
    return 0;
}

```

To learn more about C++:

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

Department of Electrical and Computer Engineering
Boston University

BOSTON UNIVERSITY

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

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

Department of Electrical and Computer Engineering
Boston University

BOSTON UNIVERSITY

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!

Department of Electrical and Computer Engineering

Boston University

BOSTON UNIVERSITY