
**Ho Chi Minh City University of Technology**
**Faculty of Computer Science and Engineering**

# Chapter 2: C Program Structure and its Components

Introduction to Computer Programming
(C language)

Nguyễn Tiến Thịnh, Ph.D.

Email: ntthinh@hcmut.edu.vn

2022 – 2023, Semester 1

# Course Content

- C.1. Introduction to Computers and Programming
- **C.2. C Program Structure and its Components**
- C.3. Variables and Basic Data Types
- C.4. Selection Statements
- C.5. Repetition Statements
- C.6. Functions
- C.7. Arrays
- C.8. Pointers
- C.9. File Processing

# References

- [1] "*C: How to Program*", 7th Ed. – Paul Deitel and Harvey Deitel, Prentice Hall, 2012.
- [2] "*The C Programming Language*", 2nd Ed. – Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall, 1988
- and others, especially those on the Internet

# Content

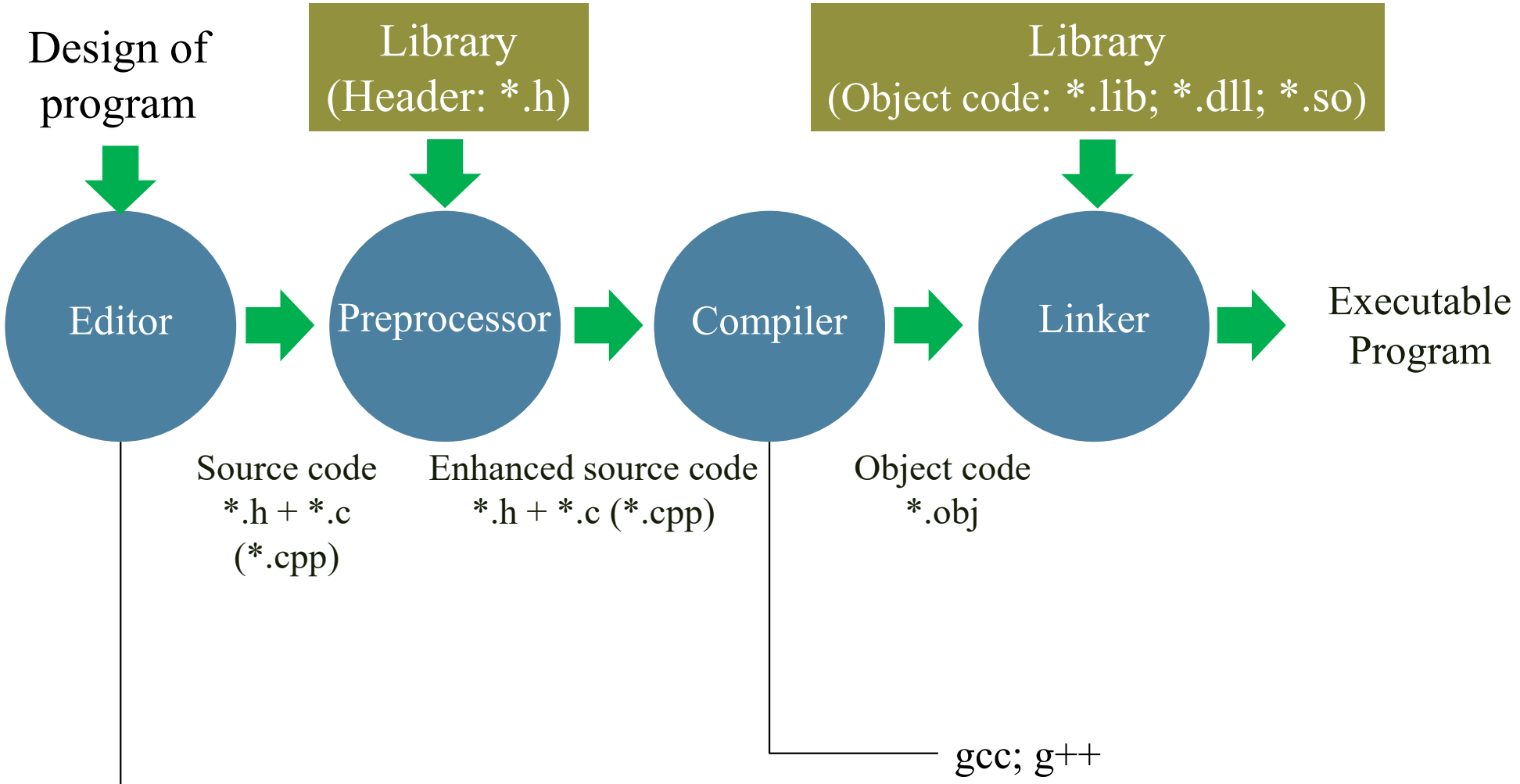
- Introduction
- A Sample C Program
- Coding Styles
- Data and Standard Output Function
- Data and Standard Input Function
- Data Processing: Simple Example
- Summary

# Introduction

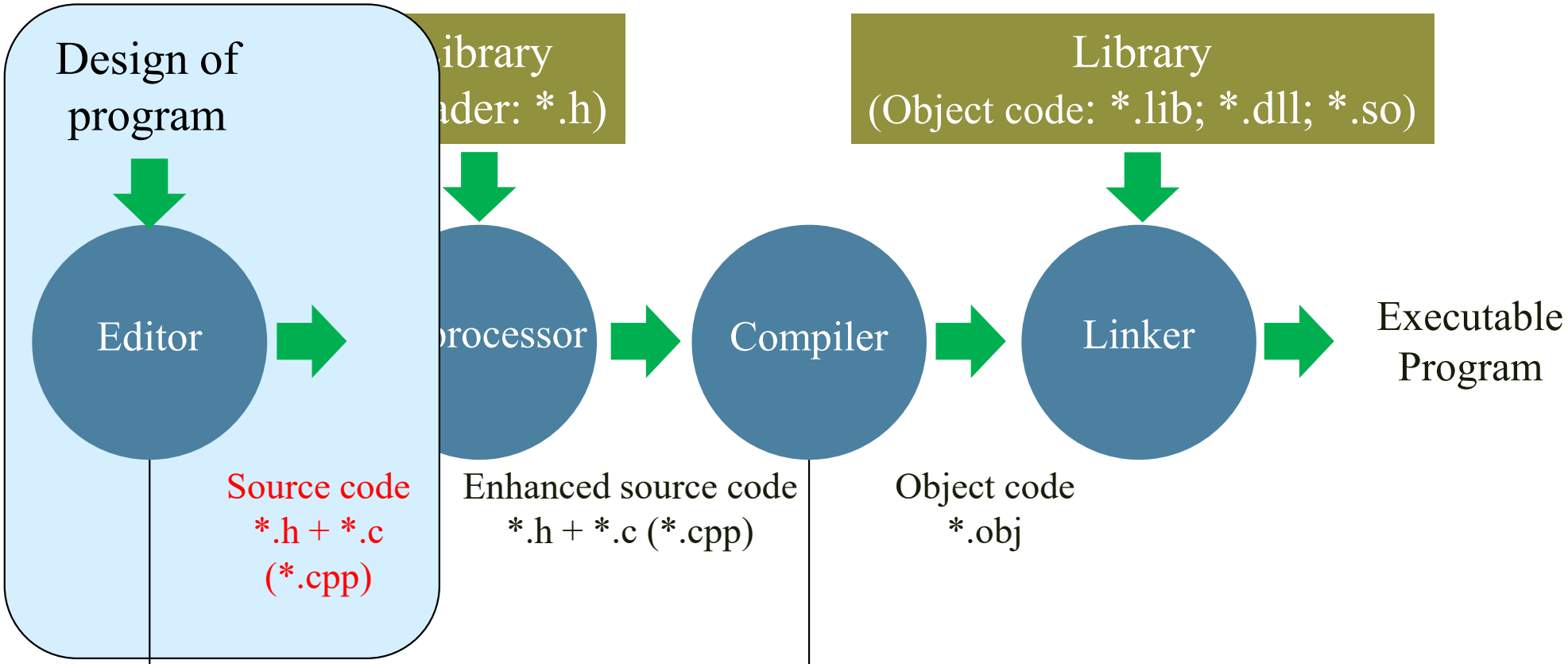
- "In our experience, C has proven to be a pleasant, expressive, and versatile language for a wide variety of programs. It is easy to learn, and it wears well as one's experience with it grows."
  - [2], Brian W. Kernighan and Dennis M. Ritchie

versatile = able to be used for many different purposes

# Introduction



# Introduction



# A Sample C Program

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {

    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

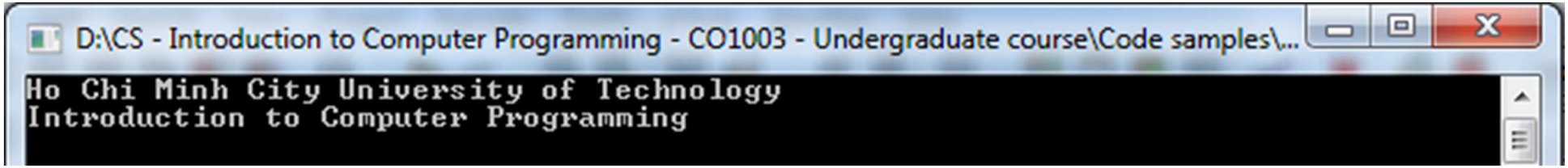
A source code file named C2_example1.c

Purpose: display our university's name and course's name on the screen

# A Sample C Program

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {

    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

A source code file named C2_example1.c



# A Sample C Program

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {
    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

Every C program requires the specific function named "main" exactly.

The body of the "main" function is enclosed in the brackets {…}.

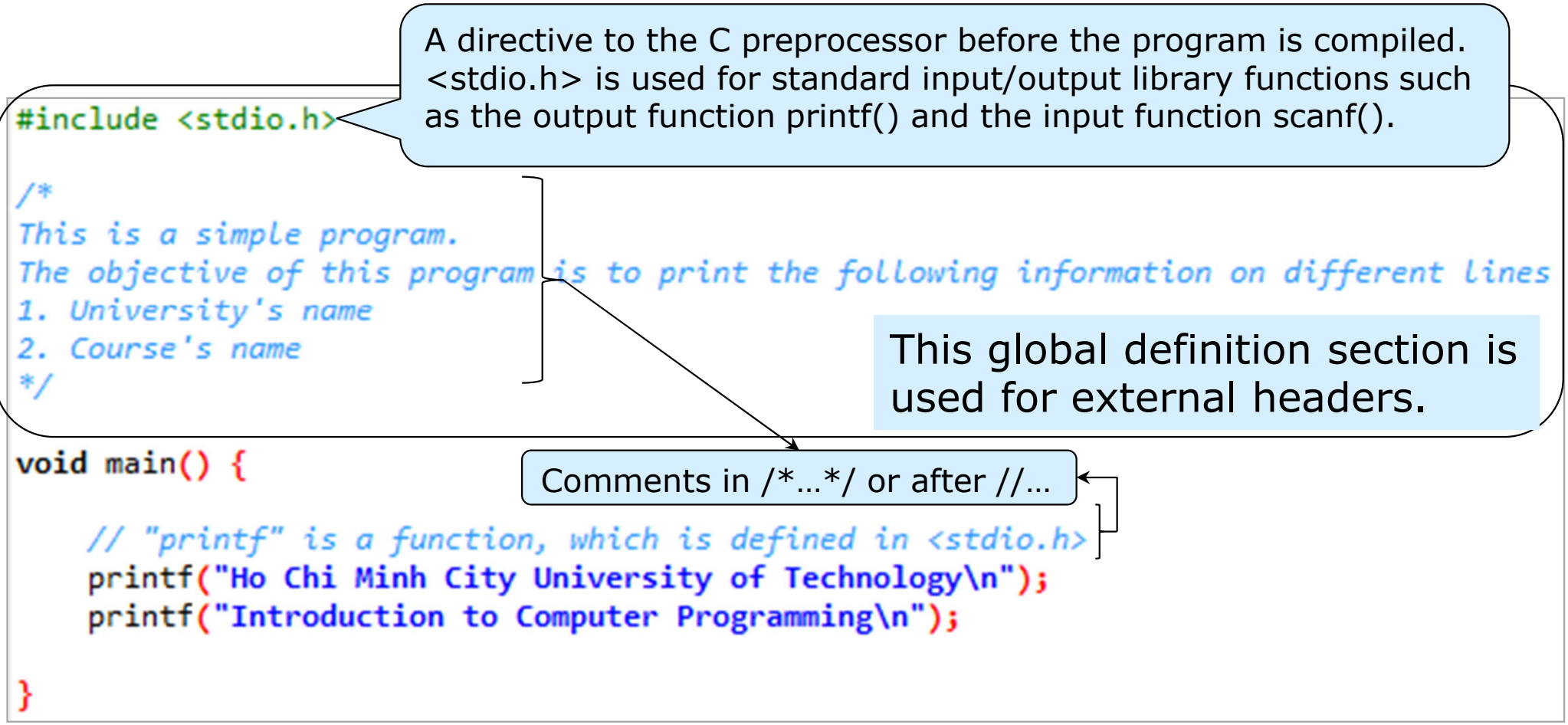
The body of the "main" function is made to serve the specific purpose.

A source code file named C2_example1.c

The specific extension of C source code files

Purpose: display our university's name and course's name on the screen

# A Sample C Program




A source code file named C2_example1.c

Purpose: display our university's name and course's name on the screen

# A Sample C Program

- File name: following the naming conventions supported by the OS with the .c extension
  - Not include some special symbols: *, /, \, …
  - Avoid using special symbols even if they are allowed
  - Prefer descriptive names
- The "main" function
  - All the files and libraries are compiled into a single executable program file with exactly one "main" function.
  - The starting point for the execution of a C program
  - Note: The C language is case-sensitive.
    - MAIN vs. Main vs. main

# A Sample C Program

## The "main" function

```
void main(){
        …
        …
}
```

Open bracket "{" to start the body section

Corresponding close bracket "}" to end the body section

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

Place for specifying the parameters of the function

**Empty** or **void**: no parameter is specified

Data type of the value returned by the "main" function

**int**: a status value: 0 = no error; 1 = error

EXIT_SUCCESS = 0; EXIT_FAILURE = 1

**void**: no value is returned

# A Sample C Program

## The "main" function

```
void main(){
        printf("Ho Chi Minh City University of Technology\n");
        printf("Introduction to Computer Programming\n");
}
```

statement

statement

Each statement

- ended with a semicolon (;)
- stretched on multiple lines with a backslash \ at the end
- able to be grouped in the brackets {}
- not consider spaces

Several statement types

- sequence: function calling, assignment, break, continue, return, …
- selection: if, if… else, switch
- repetition: for, while, do… while

# A Sample C Program

- In addition to "main", there are keywords/reserved words in the C language.

**Keywords** [1], p. 42

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| const | float | short | unsigned |
| continue | for | signed | void |
| default | goto | sizeof | volatile |
| do | if | static | while |

*Keywords added in C99*

`_Bool` `_Complex` `_Imaginary` `inline` `restrict`

**Fig. 2.15** | C's keywords.

# A Sample C Program

- The global definition section
  - Lines beginning with #
    - The #include directive tells the preprocessor to include the contents of another file.
      - #include "myHeader.h"
        - A personal header file in the current directory
      - #include <stdio.h>
        - A standard library file in the standard directory \include
    - The #define directive to set up symbolic replacements
    - The #if test at compile-time to look at the symbols in #define and turn on and off which lines the compiler uses
  - Function prototypes to validate function calls
  - Global variable declarations

# A Sample C Program

- Comments
  - /*…*/: multi-line comment
  - //…: single-line comment

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {

    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

# A Sample C Program

How to get the program run for the specified purpose? *.c => a machine code file (e.g. *.exe)

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {

    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

A source code file named C2_example1.c

Purpose: display our university's name and course's name on the screen

# A Sample C Program

Edit:

C2_example1.c

Compile:

gcc –Wall C2_example1.c –o C2_example1

-Wall: display all the warning types along with the errors if any

-o C2_example1: specify a name for the resulting executable program

Run:

C2_example1

# A Sample C Program

- GCC = GNU Compiler Collection
- The free GNU C++: gcc.gnu.org/install/binaries.html
- GCC is available for most platforms, including Linux, Mac OS X (via Xcode) and Windows—via tools like Cygwin (www.cygwin.com) and MinGW (www.mingw.org)
- Free IDEs: Micrsoft Visual Studio Express Editions, CodeBlock, Dev C++, …

# Coding Styles

Which one do you prefer?

(A)

```c
#include <stdio.h>
void main() {
printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");
}
```

(B)

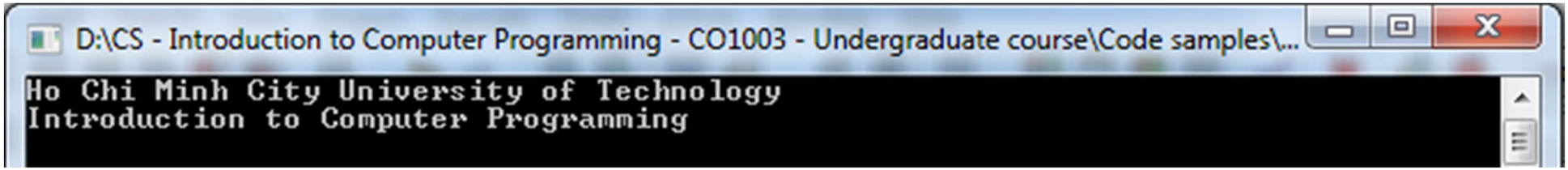
```c
#include <stdio.h>
/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/
void main() {
// "printf" is a function, which is defined in <stdio.h>
printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");
}
```

(C)

```c
#include <stdio.h>

/*
This is a simple program.
The objective of this program is to print the following information on different lines
1. University's name
2. Course's name
*/

void main() {

    // "printf" is a function, which is defined in <stdio.h>
    printf("Ho Chi Minh City University of Technology\n");
    printf("Introduction to Computer Programming\n");

}
```

# Coding Styles

- The three different formatted examples for the same purpose return the same result:




- What problems might we face with (A)?
- What problems might we face with (B)?
- What problems might we face with (C)?
- What else if the program is not defined with just a few lines; instead, with thousands of lines?

# Coding Styles

- Alignment
  - Statements are aligned level by level based on the block {} that they are contained.
- Separation (White space)
  - Each statement should placed on a single line.
  - Blank lines should be used for more clarity.
- Comment
  - A comment is added at any level for explanation, marking, tracking, …
  - Comments from the pseudo code helpful for program development
  - Never abuse comments to make a program clear

# Coding Styles

Naming conventions

- Descriptive
- Classified
- Consistent
- Agreements if any

Easy to remember, easy to be referred, easy to be communicated

Less error-prone

# Coding Styles

## Naming conventions

- What can you name in a C program except file names?

  Function: your defined modular processing unit

  Variable: a location in memory where a value can be stored for use by a program

  Symbolic constant: constant represented as a symbol (a string) that will be replaced with the value it stands for

A name should reflect the content and role of the thing you define in a C program.

# Coding Styles

- Naming conventions
  - Create a name made up of at most 31 letters, digits, underscore "_" and starting with a letter
  - Consider verbs, nouns, phrases, …
  - Consider a length of each name: short/long
  - Consider lower-case/upper-case letters
  - Consider prefix/suffix parts
    - Function: verb, phrase
    - Variable: noun, lower-case letters, short for loops
      - Global variable: prefix g_
      - Data type: suffix _ptr for pointers
    - Symbolic constant: upper-case letters, underscores

# Coding Styles

| Should ● | Should not X |
| --- | --- |
| check_for_errors() | error_checking()<br>dump_data_to_file() |
| is_any_upper_case() | any_upper_case_test() |
| get_an_upper_case() | return_an_upper_case() |
| set_first_upper_case() | change_first_upper_case() |
| Student *student_prt; | Student * student; |
| int g_thread_number; | int thread_number; |
| static const double PI = 3.14; | static const double Pi = 3.14; |
| #define SIZE 10 | #define size 10 |
| … | … |

# Coding Styles

- How to make the following program clearer?

```
main() //Add a and b
{ int c;
int a, b;

a = 5; b = 10;

c = a + b;
printf("%d\n", c);

}
```

# Coding Styles

- How to make the following program clearer?

```
main() //Add a and b
{ int c;
int a, b;

a = 5; b = 10;

c = a + b;
printf("%d\n", c);

}
```

```
#include <stdio.h>

//Add a and b
//input: a, b
//output: c printed on screen
void main()
{
    int number_a, number_b;
    int number_c;

    //prepare a and b
    number_a = 5;
    number_b = 10;

    //process for c
    number_c = number_a + number_b;

    //print c
    printf("%d\n", number_c);
}
```

# Coding Styles

Check your naming conventions in your pseudo code

How many names would you like to rename?

What else have you changed on your pseudo code?

How clearer is your pseudo code?

# Coding Styles

Practice the formatting rules and conventions to be part of your own coding habit

Make sure that your submitted programs are always well-formatted

Make sure that your future programs are readable, interpretable, able to be maintained, reused, and extended

More information about coding styles at:

- http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html
- http://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html

# Data and Standard Output Function

- Standard output function: printf()
  - #include <stdio.h>
  - Output the data to the standard output stream
    - A sequence of bytes flows from main memory to an output device: display screen, printer, disk drive, network connection, and so on.
    - Normally and automatically, the standard output stream is connected to the screen.

Examples:

```
printf("Introduction to Computer Programming\n");
printf("\n(x,y) = (%5.2f,%5.2f)\n", x, y);
```

# Data and Standard Output Function

Formatting capabilities of the printf function

- Rounding floating-point values to an indicated number of decimal places.
- Aligning a column of numbers with decimal points appearing one above the other.
- Right justification and left justification of outputs.
- Inserting literal characters at precise locations in a line of output.
- Representing floating-point numbers in exponential format.
- Representing unsigned integers in octal and hexadecimal format.
- Displaying all types of data with fixed-size field widths and precisions.

# Data and Standard Output Function

```
printf( format-control-string, other-arguments );
```

- *format-control-string* describes the output format.
- *other-arguments* (optional) correspond to each conversion specification in *format-control-string.*

Each conversion specification begins with a percent sign (%) and ends with a conversion specifier.

There can be many conversion specifications in one format control string.

Examples:

```
printf("Introduction to Computer Programming\n");

printf("\n(x,y) = (%5.2f,%5.2f)\n", x, y);
```

The 1st conversion specification: %5.2f => argument: x

The 2nd conversion specification: %5.2f => argument: y

# Data and Standard Output Function

```
printf( format-control-string, other-arguments );
```

format-control-string is summarized as follows:

%[flag][width][.precision]specifier

specifier = d/i, u, o, x/X, f/F, e/E, g/G, a/A, c, s, p, n, %

precision = .number, .*

width = number, *

flag = +, -, space, #, 0

# Data and Standard Output Function

| Conversion specifier | Description |
|---|---|
| `d` | Display as a signed decimal integer. |
| `i` | Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.] |
| `o` | Display as an unsigned octal integer. |
| `u` | Display as an unsigned decimal integer. |
| `x` or `X` | Display as an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed. |
| `h` or `l` (letter `l`) | Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed, respectively. Letters `h` and `l` are more precisely called length modifiers. |

| Conversion specifier | Description |
|---|---|
| `e` or `E` | Display a floating-point value in exponential notation. |
| `f` | Display floating-point values in fixed-point notation. [*Note:* In C99, you can also use `F`.] |
| `g` or `G` | Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`), based on the magnitude of the value. |
| `L` | Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed. |

# Data and Standard Output Function

| Conversion specifier | Description |
|---|---|
| p | Display a pointer value in an implementation-defined manner. |
| n | Store the number of characters already output in the current `printf` statement. A pointer to an integer is supplied as the corresponding argument. Nothing is displayed. |
| % | Display the percent character. |
| c | Display a character. |
| s | Display a sequence of characters until a terminating null \0 character is encountered. |

# Data and Standard Output Function

Flags to supplement its output formatting capabilities,

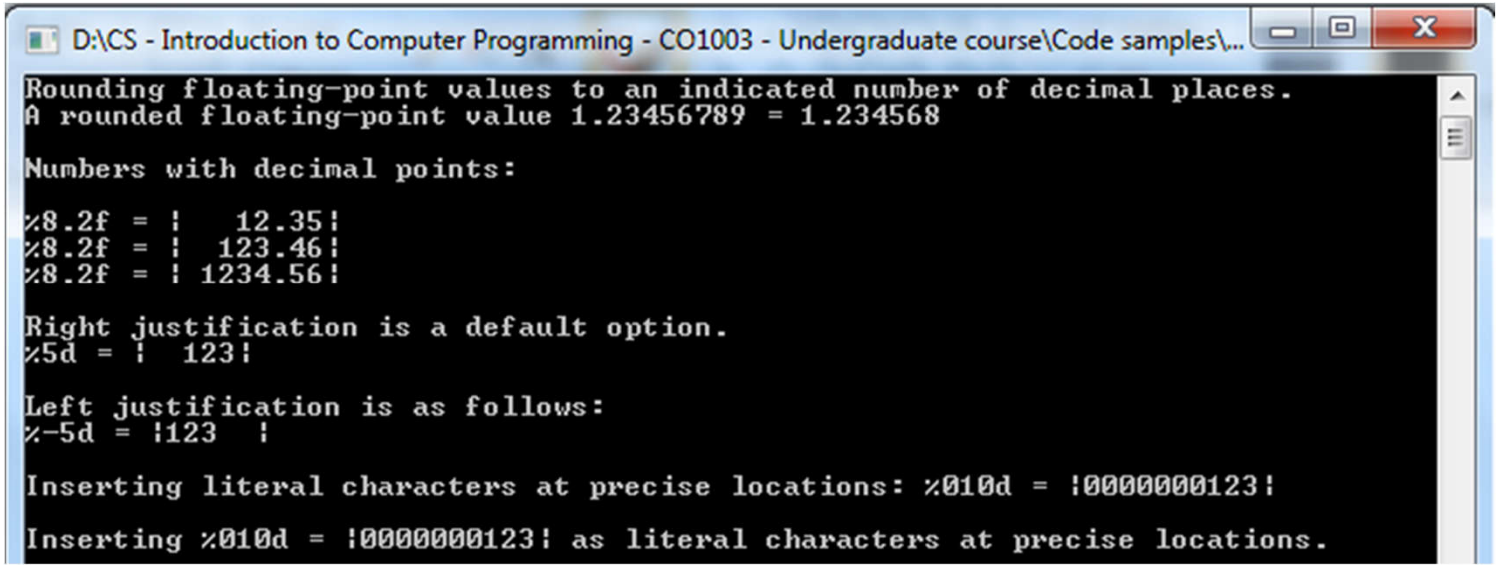
placed immediately to the right of the percent sign

| Flag | Description |
|---|---|
| - (minus sign) | Left justify the output within the specified field. |
| + (plus sign) | Display a plus sign preceding positive values and a minus sign preceding negative values. |
| *space* | Print a space before a positive value not printed with the + flag. |
| # | Prefix `0` to the output value when used with the octal conversion specifier `o`. |
| | Prefix `0x` or `0X` to the output value when used with the hexadecimal conversion specifiers `x` or `X`. |
| | Force a decimal point for a floating-point number printed with `e`, `E`, `f`, `g` or `G` that does not contain a fractional part. (Normally the decimal point is printed only if a digit follows it.) For `g` and `G` specifiers, trailing zeros are not eliminated. |
| 0 (zero) | Pad a field with leading zeros. |

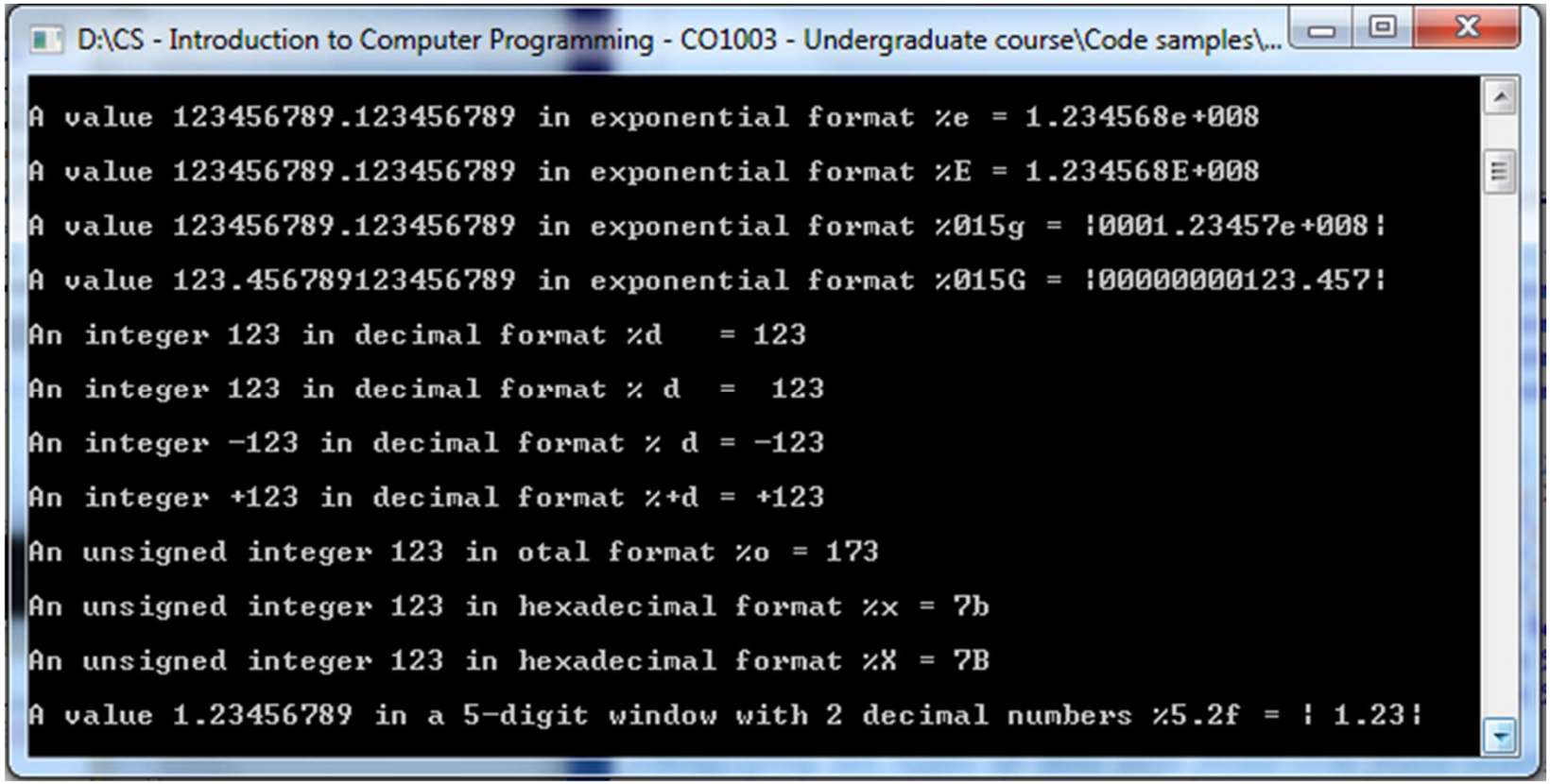
# Data and Standard Output Function

| Escape sequence | Description |
|---|---|
| \' (single quote) | Output the single quote (') character. |
| \" (double quote) | Output the double quote (") character. |
| \? (question mark) | Output the question mark (?) character. |
| \\ (backslash) | Output the backslash (\) character. |
| \a (alert or bell) | Cause an audible (bell) or visual alert. |
| \b (backspace) | Move the cursor back one position on the current line. |
| \f (new page or form feed) | Move the cursor to the start of the next logical page. |
| \n (newline) | Move the cursor to the beginning of the next line. |
| \r (carriage return) | Move the cursor to the beginning of the current line. |
| \t (horizontal tab) | Move the cursor to the next horizontal tab position. |
| \v (vertical tab) | Move the cursor to the next vertical tab position. |

```c
#include <stdio.h>

void main() {

    //Rounding floating-point values to an indicated number of decimal places.
    printf("Rounding floating-point values to an indicated number of decimal places.\n");
    printf("A rounded floating-point value 1.23456789 = %f", 1.23456789);

    //Aligning a column of numbers with decimal points appearing one above the other.
    printf("\n\nNumbers with decimal points: \n");
    printf("\n%%8.2f = |%8.2f|", 12.3456);
    printf("\n%%8.2f = |%8.2f|", 123.456);
    printf("\n%%8.2f = |%8.2f|", 1234.56);

    //Right justification and left justification of outputs.
    printf("\n\nRight justification is a default option.\n");
    printf("%%5d = |%5d|", 123);

    printf("\n\nLeft justification is as follows:\n");
    printf("%%-5d = |%-5d|", 123);

    //Inserting literal characters at precise locations in a line of output.
    printf("\n\nInserting literal characters at precise locations: %%010d = |%010d|", 123);
    printf("\n\nInserting %%010d = |%010d| as literal characters at precise locations.", 123);
```

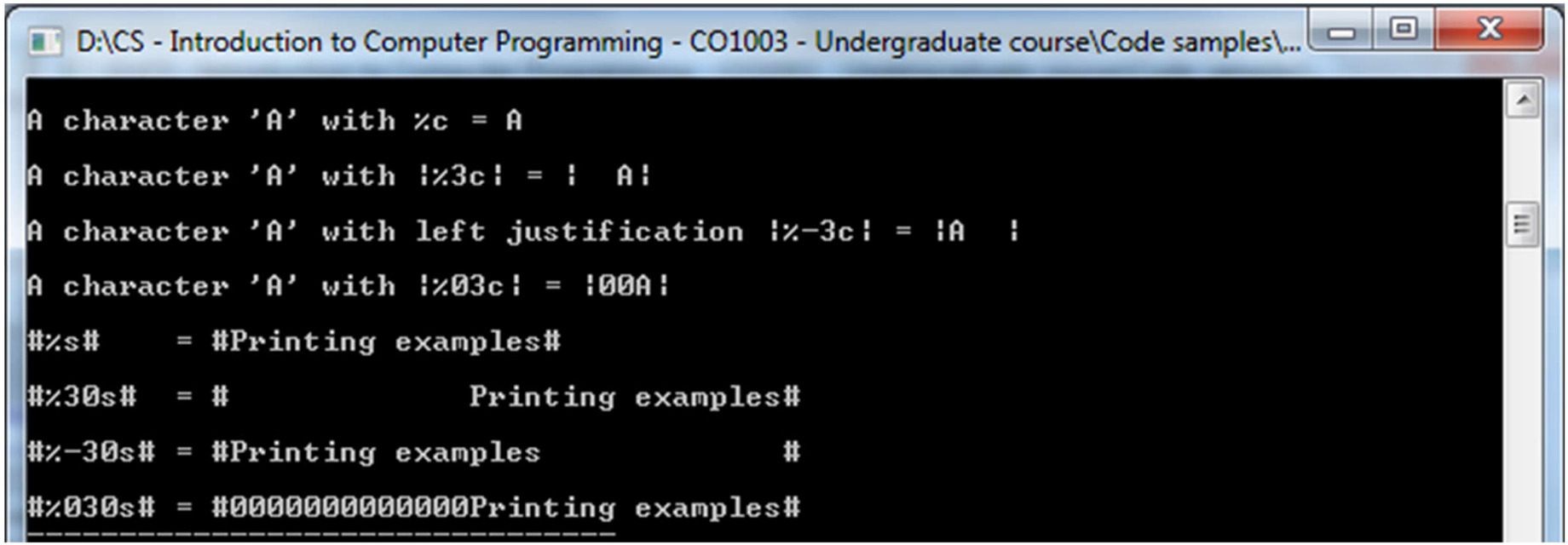


```
//Representing floating-point numbers in exponential format.
printf("\n\nA value 123456789.123456789 in exponential format %%e = %e", 123456789.123456789);
printf("\n\nA value 123456789.123456789 in exponential format %%E = %E", 123456789.123456789);
printf("\n\nA value 123456789.123456789 in exponential format %%015g = |%015g|", 123456789.123456789);
printf("\n\nA value 123.456789123456789 in exponential format %%015G = |%015G|", 123.456789123456789);

//Representing unsigned integers in octal and hexadecimal format.
printf("\n\nAn integer 123 in decimal format %%d   = %d", 123);
printf("\n\nAn integer 123 in decimal format %% d  = % d", 123);
printf("\n\nAn integer -123 in decimal format %% d = % d", -123);
printf("\n\nAn integer +123 in decimal format %%+d = %+d", 123);
printf("\n\nAn unsigned integer 123 in otal format %%o = %o", 123);
printf("\n\nAn unsigned integer 123 in hexadecimal format %%x = %x", 123);
printf("\n\nAn unsigned integer 123 in hexadecimal format %%X = %X", 123);

//Displaying all types of data with fixed-size field widths and precisions.
printf("\n\nA value 1.23456789 in a 5-digit window with 2 decimal numbers %%5.2f = |%5.2f|", 1.23456789);
```



# Data and Standard Output Function

```
    //Printing strings and characters
    printf("\n\nA character \'A\' with %%c = %c", 'A');
    printf("\n\nA character \'A\' with |%%3c| = |%3c|", 'A');
    printf("\n\nA character \'A\' with left justification |%%-3c| = |%-3c|", 'A');
    printf("\n\nA character \'A\' with |%%03c| = |%03c|", 'A');

    printf("\n\n#%%s#     = #%s#", "Printing examples");
    printf("\n\n#%%30s#   = #%30s#", "Printing examples");
    printf("\n\n#%%-30s# = #%-30s#", "Printing examples");
    printf("\n\n#%%030s# = #%030s#", "Printing examples");
}
```



# Data and Standard Input Function

- Standard input function: scanf()
  - #include <stdio.h>
  - Input the data from the standard input stream
    - A sequence of bytes flows from an input device: keyboard, disk drive, network connection, and so on to main memory.
    - Normally and automatically, the standard input stream is connected to the keyboard.

```
scanf( format-control-string, other-arguments );
```

*format-control-string* describes the formats of the input.

*other-arguments* are pointers to variables in which the input will be stored.

# Data and Standard Input Function

The conversion specifiers used to input all types of data

| Conversion specifier | Description |
|---|---|
| *Integers* | |
| `d` | Read an optionally signed decimal integer. The corresponding argument is a pointer to an `int`. |
| `i` | Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an `int`. |
| `o` | Read an octal integer. The corresponding argument is a pointer to an unsigned `int`. |
| `u` | Read an unsigned decimal integer. The corresponding argument is a pointer to an unsigned `int`. |
| `x` or `X` | Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned `int`. |
| `h` or `l` | Place before any of the integer conversion specifiers to indicate that a `short` or `long` integer is to be input. |

```
printf("Input a (un)signed decimal integer:");
scanf("%d", ...);
```

# Data and Standard Input Function

The conversion specifiers used to input all types of data

| | |
|---|---|
| *Floating-point numbers* | |
| `e`, `E`, `f`, `g` or `G` | Read a floating-point value. The corresponding argument is a pointer to a floating-point variable. |
| `l` or `L` | Place before any of the 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. |
| *Characters and strings* | |
| `c` | Read a character. The corresponding argument is a pointer to a `char`; no null (`'\0'`) is added. |
| `s` | Read a string. The corresponding argument is a pointer to an array of type `char` that is large enough to hold the string and a terminating null (`'\0'`) character—which is automatically added. |

```
printf("Input a floating-point value:");
scanf("%f", ...);

printf("Input a character:");
scanf("%c", ...);

printf("Input a string:");
scanf("%s", ...);
```

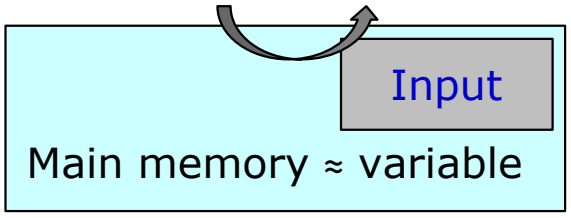
# Data and Standard Input Function

The conversion specifiers used to input all types of data

| | |
|---|---|
| *Scan set* | |
| [*scan characters*] | Scan a string for a set of characters that are stored in an array. |
| *Miscellaneous* | |
| p | Read an address of the same form produced when an address is output with %p in a printf statement. |
| n | Store the number of characters input so far in this call to scanf. The corresponding argument is a pointer to an int. |
| % | Skip a percent sign (%) in the input. |

Input data flow
from an input device
to main memory.

Input device = keyboard



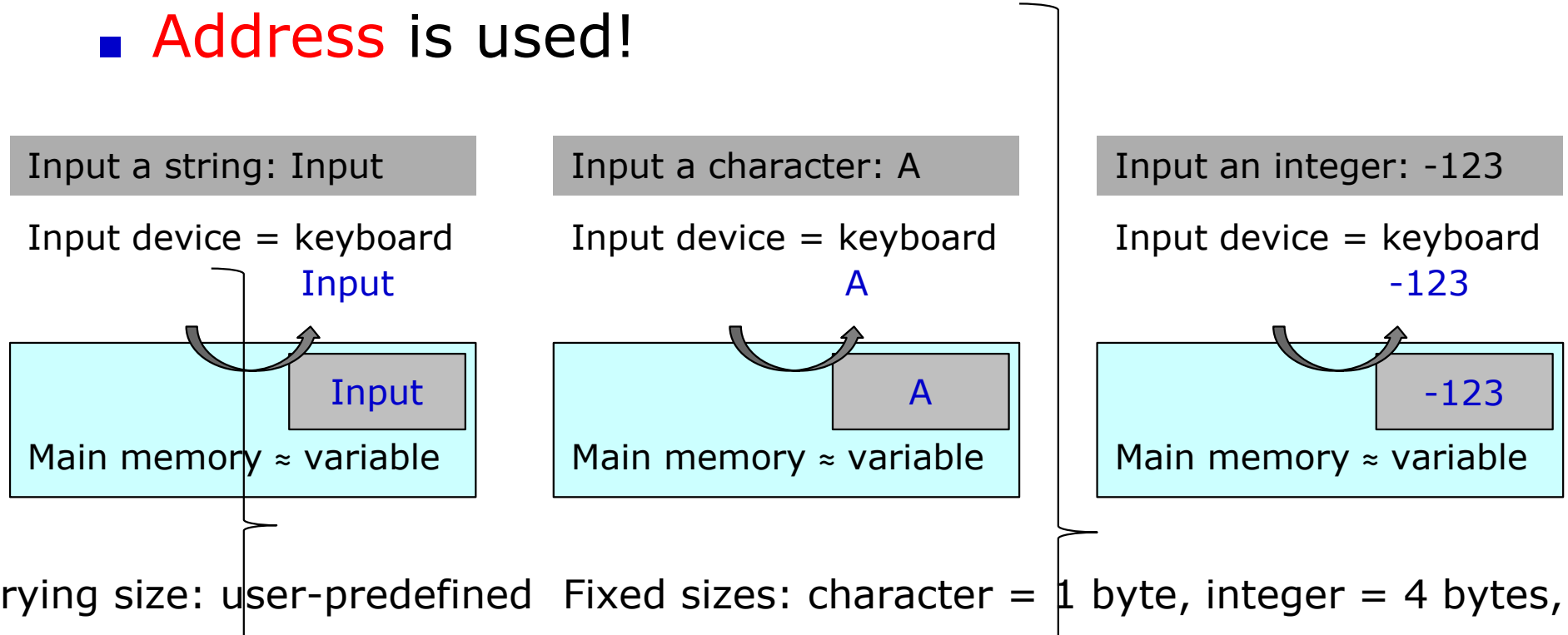

```
printf("Input a (un)signed decimal integer:");
scanf("%d", ...);
printf("Input a floating-point value:");
scanf("%f", ...);
printf("Input a character:");
scanf("%c", ...);
printf("Input a string:");
scanf("%s", ...);
```

?

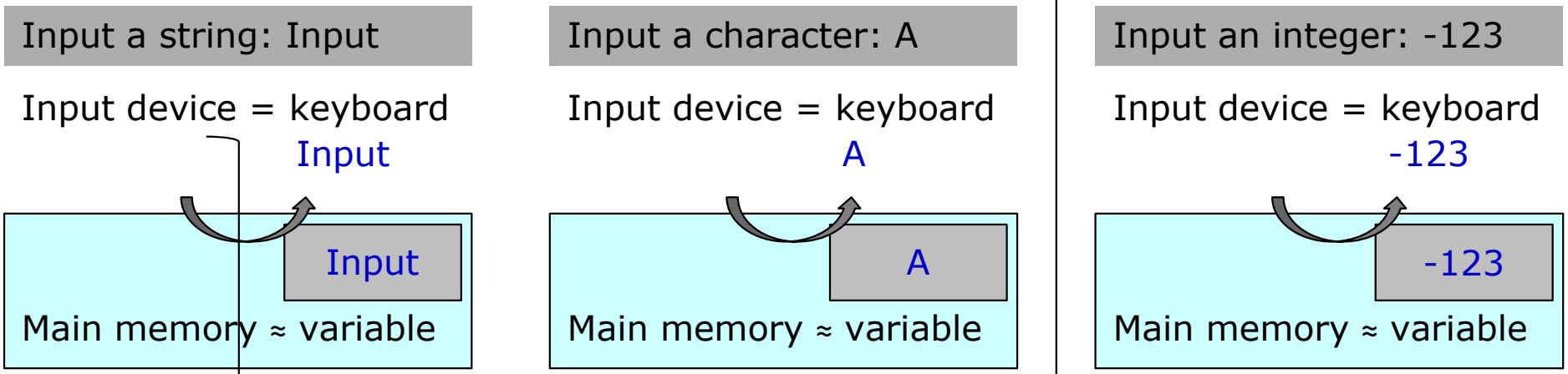
# Data and Standard Input Function

How to refer to the place in main memory where the input data will be stored?

- Memory is addressable contiguously.
- Address is used!



# Data and Standard Input Function




Varying size: user-predefined

Fixed sizes: character = 1 byte, integer = 4 bytes, …

```
char aString[5];
…
scanf("%s", aString)

printf("%s", aString)
```

```
char aChar;
…
scanf("%c", &aChar)

printf("%c", aChar)
```

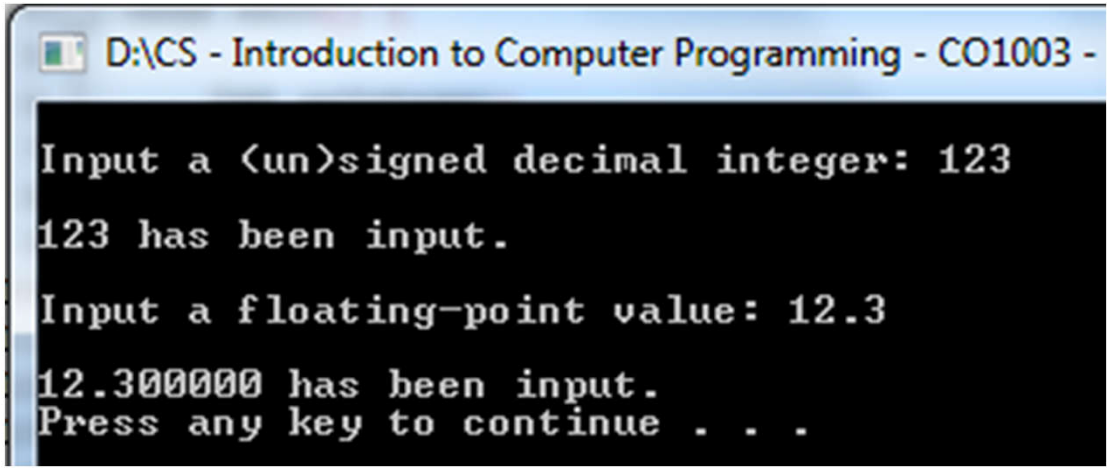
```
int anInteger;
…
scanf("%d", &anInteger)

printf("%d", anInteger)
```

In C, a string is defined as an array (sequence) of characters. The memory of this array is referred by the address of the first character represented by the name of the array.

# Data and Standard Input Function

- Put them altogether

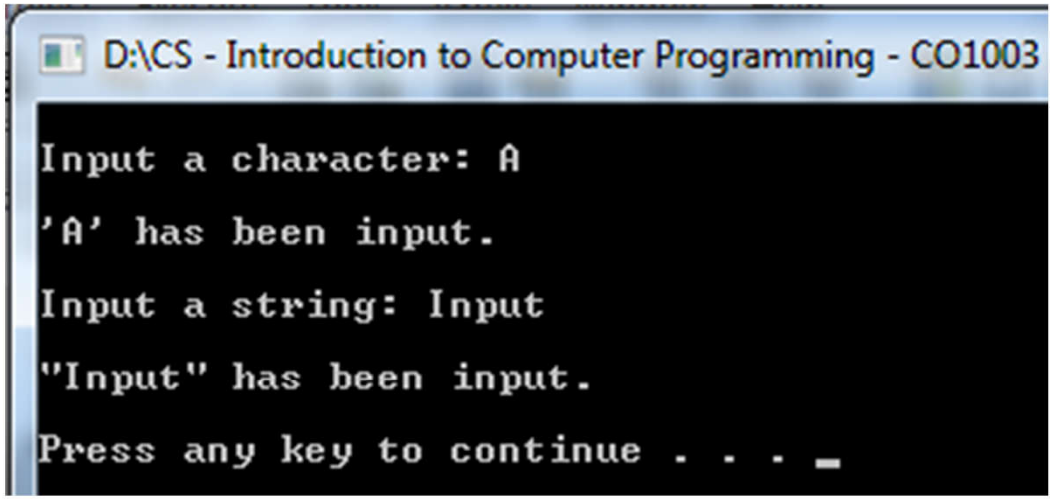
```c
#include <stdio.h>
#include <stdlib.h>

void main() {

    int anInteger;
    float aFloat;

    printf("\nInput a (un)signed decimal integer: ");
    scanf("%d", &anInteger);
    printf("\n%d has been input.", anInteger);

    printf("\n\nInput a floating-point value: ");
    scanf("%f", &aFloat);
    printf("\n%f has been input.\n", aFloat);

    system("pause");
}
```



# Data and Standard Input Function

- Put them altogether

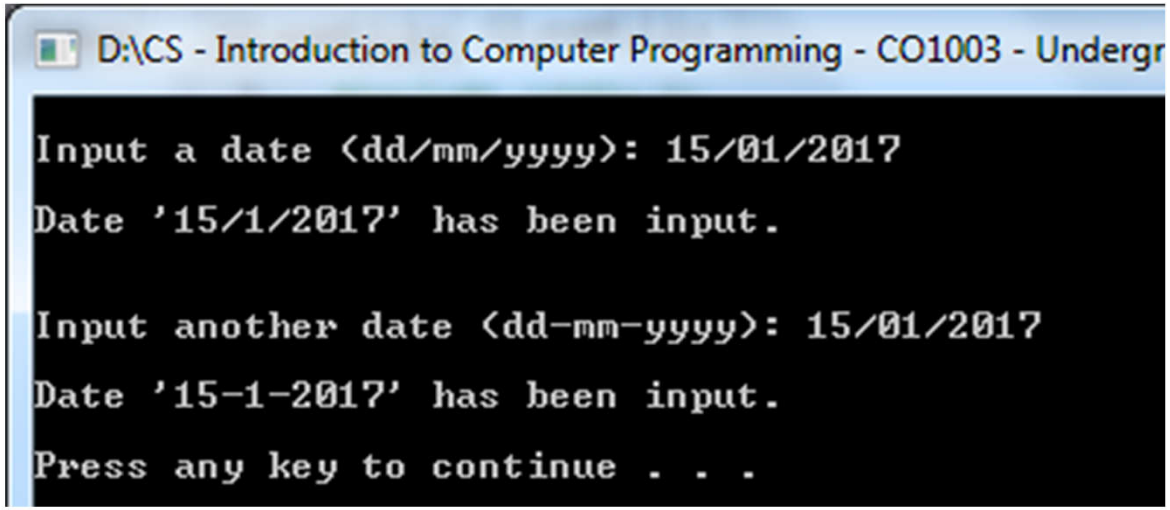
```c
#include <stdio.h>
#include <stdlib.h>

void main() {

    char aChar;
    char aString[5];

    printf("\nInput a character: ");
    scanf("%c", &aChar);
    printf("\n\'%c\' has been input.", aChar);

    printf("\n\nInput a string: ");
    scanf("%s", aString);
    printf("\n\"%s\" has been input.\n\n", aString);

    system("pause");
}
```



# Data and Standard Input Function

- Input a date in the form of: dd/mm/yyyy

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

void main() {

    int aDay;
    int aMonth;
    int aYear;

    printf("\nInput a date (dd/mm/yyyy): ");
    //scanf("%d %d %d", &aDay, &aMonth, &aYear);
    scanf("%d/%d/%d", &aDay, &aMonth, &aYear);
    printf("\nDate \'%d/%d/%d\' has been input.\n\n", aDay, aMonth, aYear);

    printf("\nInput another date (dd-mm-yyyy): ");
    scanf("%d%*c%d%*c%d", &aDay, &aMonth, &aYear);
    printf("\nDate \'%d-%d-%d\' has been input.\n\n", aDay, aMonth, aYear);

    system("pause");
}
```

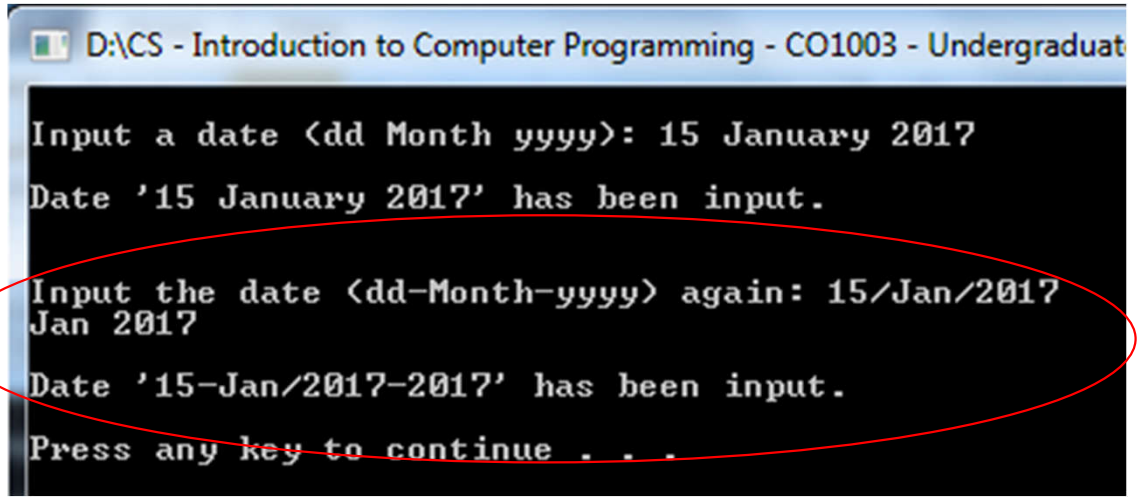


# Data and Standard Input Function

- Input a date in the form of: dd Month yyyy

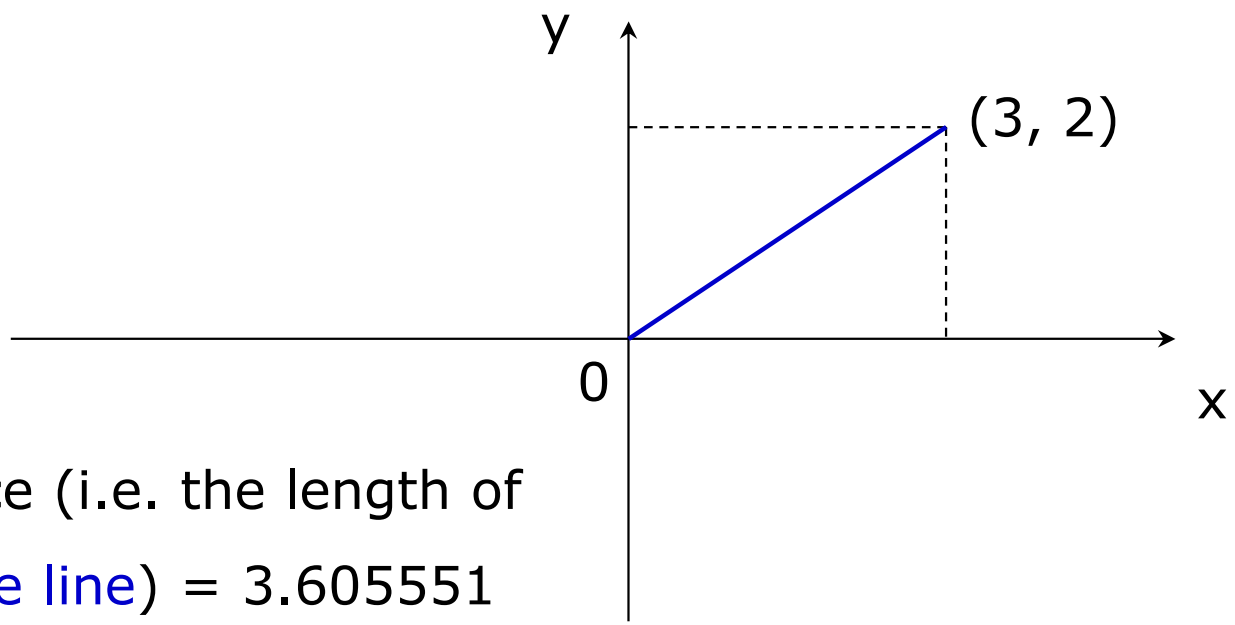
```c
#include <stdio.h>
#include <stdlib.h>

void main() {

    int aDay;
    char aMonth[8];
    int aYear;

    printf("\nInput a date (dd Month yyyy): ");
    scanf("%d %s %d", &aDay, aMonth, &aYear);
    //scanf("%d/%s/%d", &aDay, aMonth, &aYear);
    printf("\nDate \'%d %s %d\' has been input.\n\n", aDay, aMonth, aYear);

    printf("\nInput the date (dd-Month-yyyy) again: ");
    scanf("%d%*c%s%*c%d", &aDay, aMonth, &aYear);
    printf("\nDate \'%d-%s-%d\' has been input.\n\n", aDay, aMonth, aYear);

    system("pause");
}
```

???



# Data Processing: Simple Example

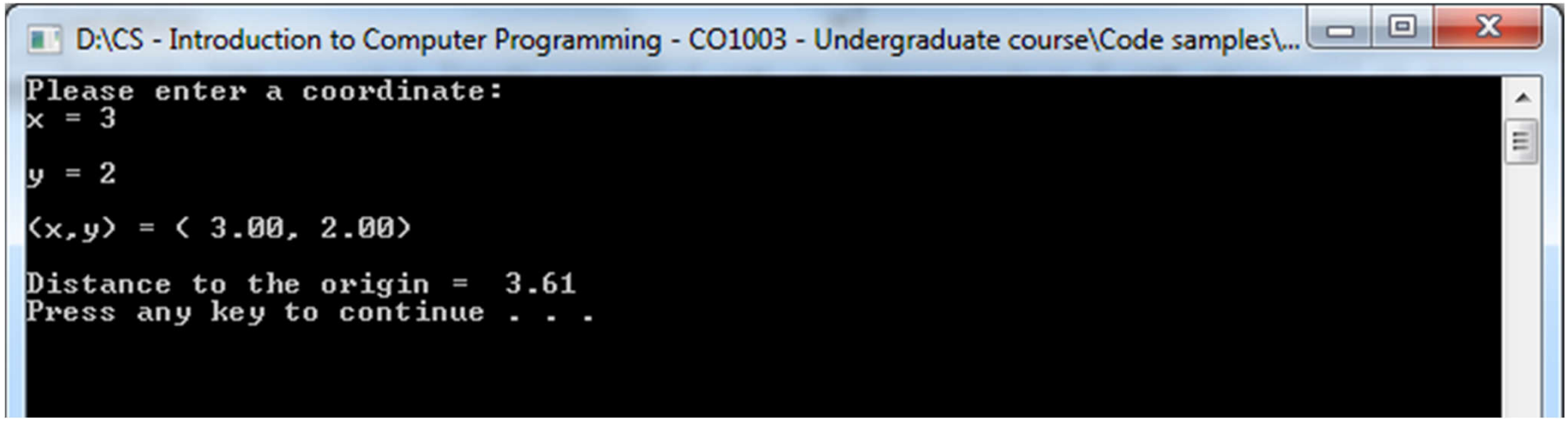
- Program for computing the distance between the center and a given point in a 2D space
  - Input: x, y as a coordinate of a given point
  - Output: the distance along with the given point on screen




Distance (i.e. the length of the blue line) = 3.605551

# Data Processing: Simple Example

- Program for computing the distance between the center and a given point in a 2D space
  - Input: x, y as a coordinate of a given point
  - Output: the distance along with the given point on screen



# Data Processing: Simple Example

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

void main(){
    float x, y;

    printf("%s", "Please enter a coordinate:");

    printf("%s", "\nx = ");
    scanf("%f", &x);

    printf("%s", "\ny = ");
    scanf("%f", &y);

    printf("\n(x,y) = (%5.2f,%5.2f)\n", x, y);
    printf("\nDistance to the origin = %5.2f\n", sqrt(x*x + y*y));

    system("pause");
}
```

- Function calling:

| | |
|---|---|
| printf() | stdio.h |
| scanf() | stdio.h |
| sqrt() | math.h |
| system() | stdlib.h |

# Summary

- Start programming with the C language
  - The components of a standard C program
  - Input & Output vs. Read & Write vs. scanf & printf
- Make your programs not only executable but also readable
  - Coding styles

# Chapter 2: C Program Structure and its Components