Lecture 8

PROBLEM SOLVING

OUTLINE

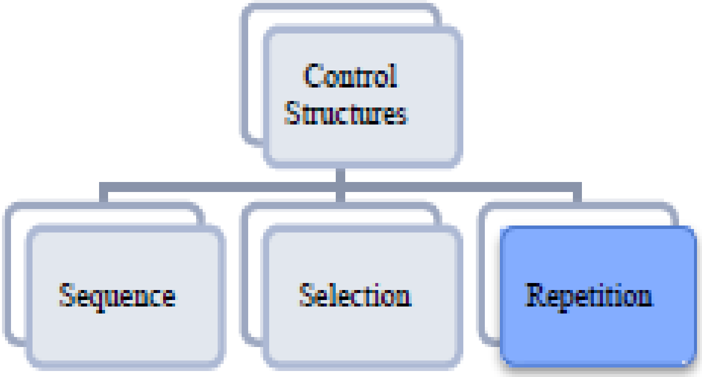
□ This lecture covers:

✓ Repetition Control Structure

✓ Programming Terminologies

✓ Introduction to C Programming

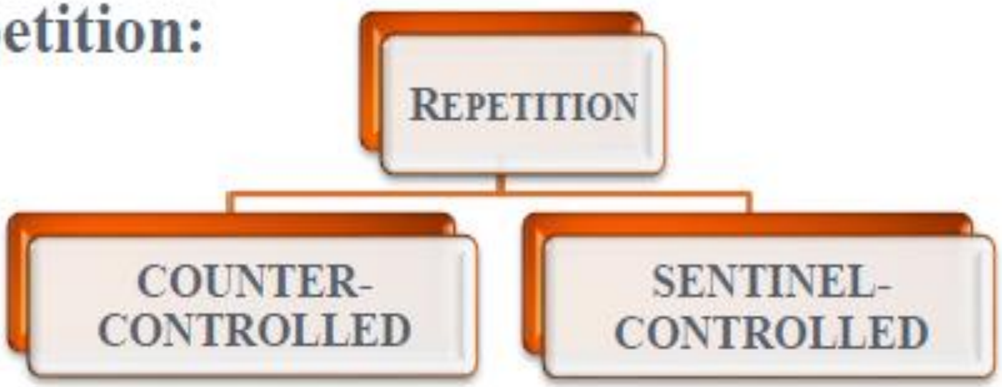
REPETITION(LOOPS)



❑Repeating a series of instructions over and over until some event occurs.

❑For example: if we wish to read 100 numbers, and then compute the average.

Two types of repetition:



❑**Three types of statements:**

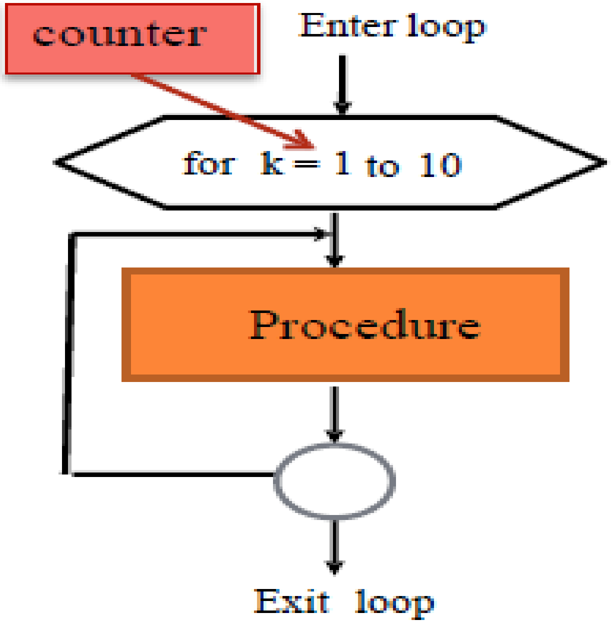
✓**For** statement

✓**While** statement

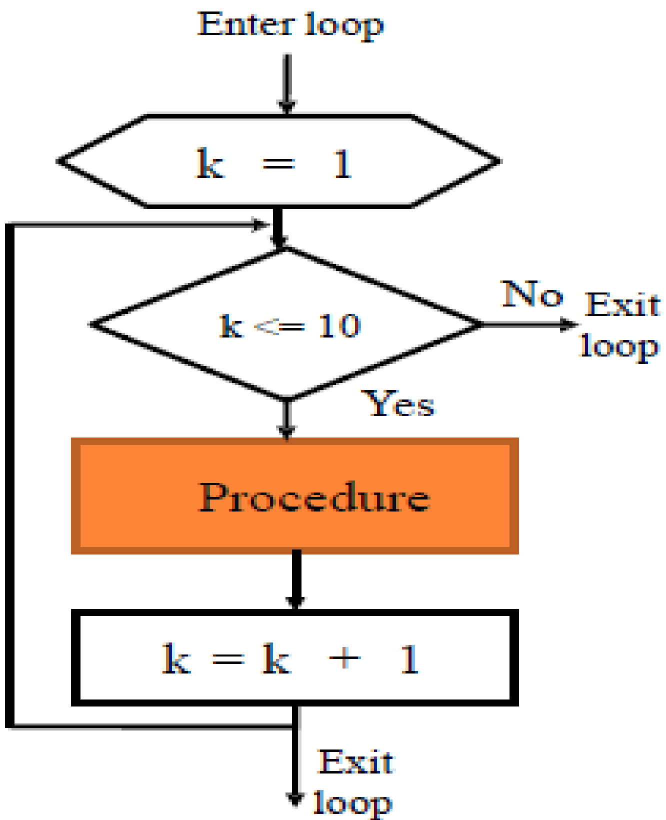
✓**Do .. While** statement

COUNTER-CONTROLLEDREPETITION

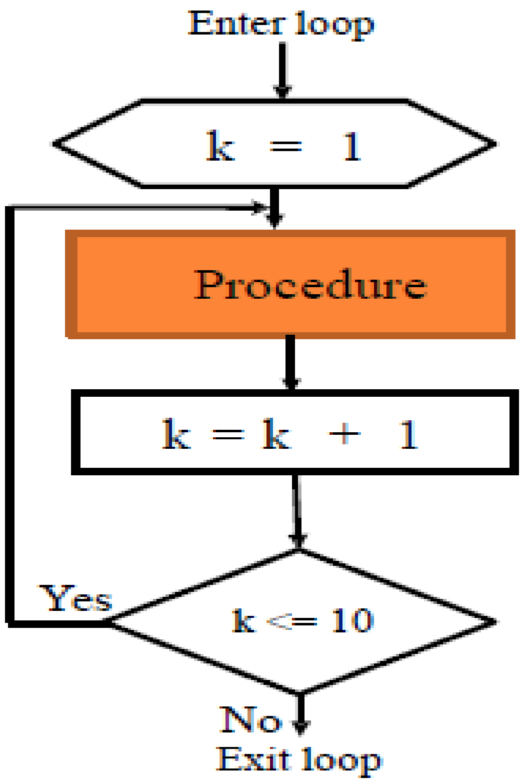
- ❑ Number of repetitions is known before it begins.



for statement



while statement



Do .. while statement

COUNTER-CONTROLLEDREPETITION

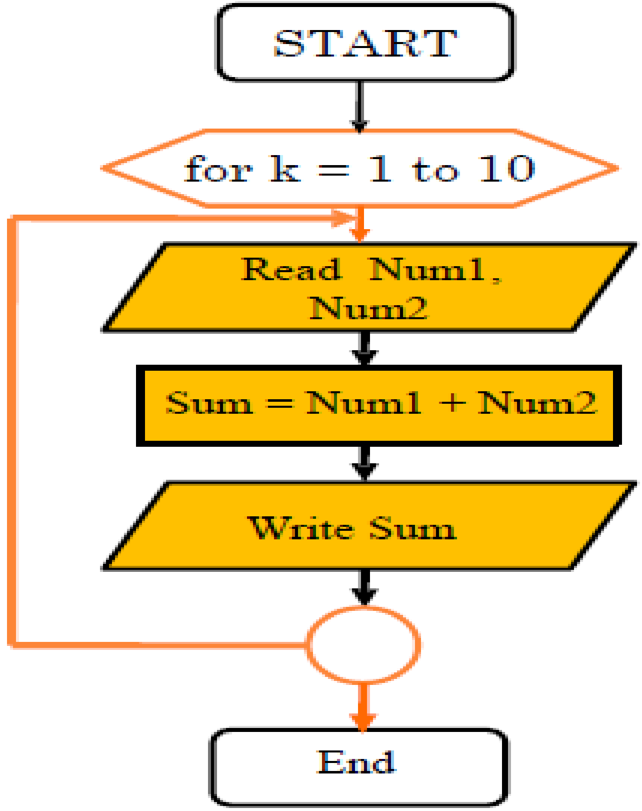
Example 8:

Summation of two numbers for 10 iterations

Pseudocode

```
For k=1 to 10
Read Num1,Num2
Compute Sum as Num1+Num2
Write Sum
End for
```

1. Solve the problem for one case.
2. Specify the statements to be repeated.



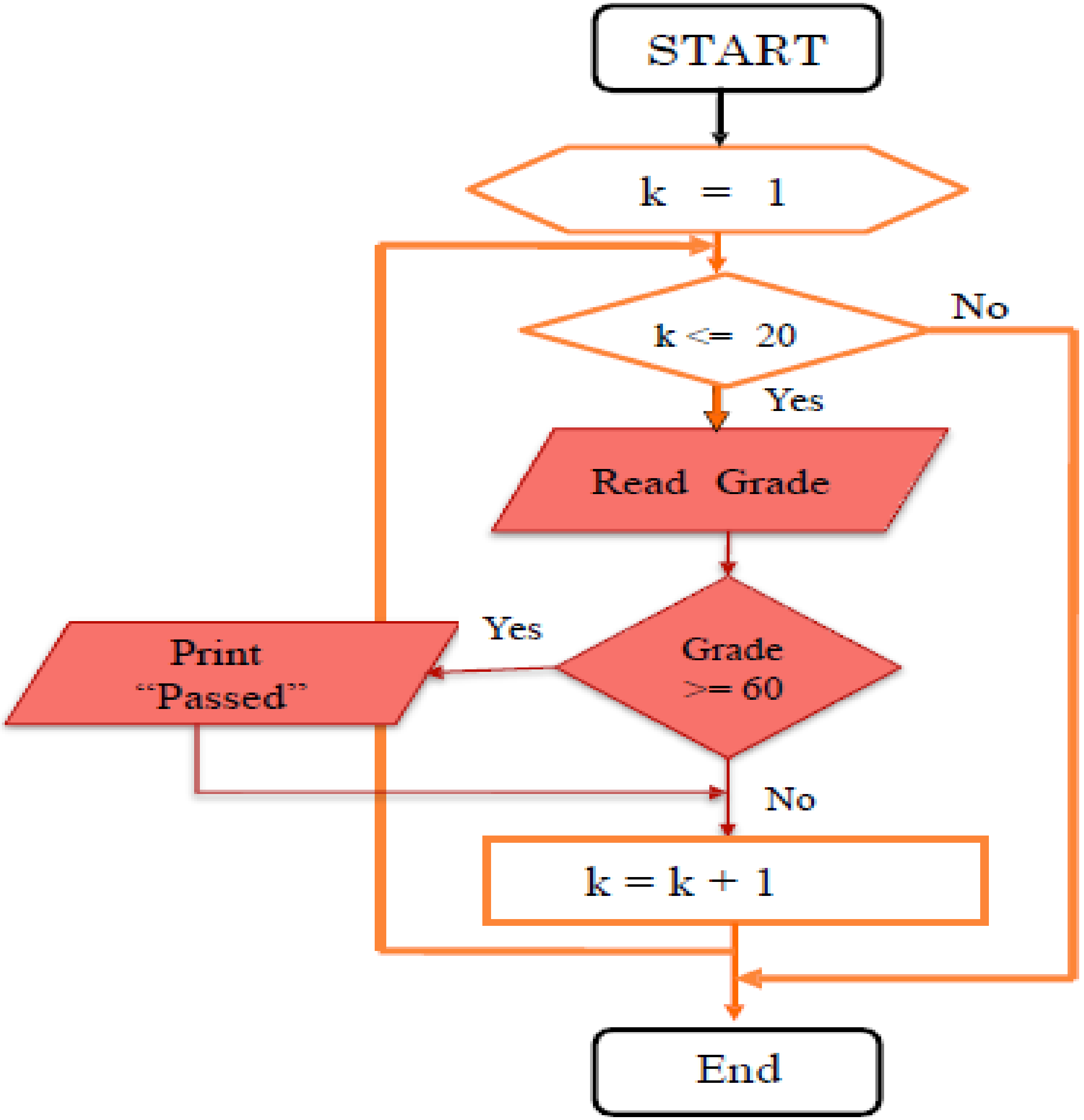
Example 9:

Reads 20 students' grades, and write "passed"

if a grade is greater than or equal 60.

Pseudocode

```
Read Grade
Initialize k=1
While k<=20
Read Grade
If Grade>=60
Print " Passed "
k=k+1
End while
```



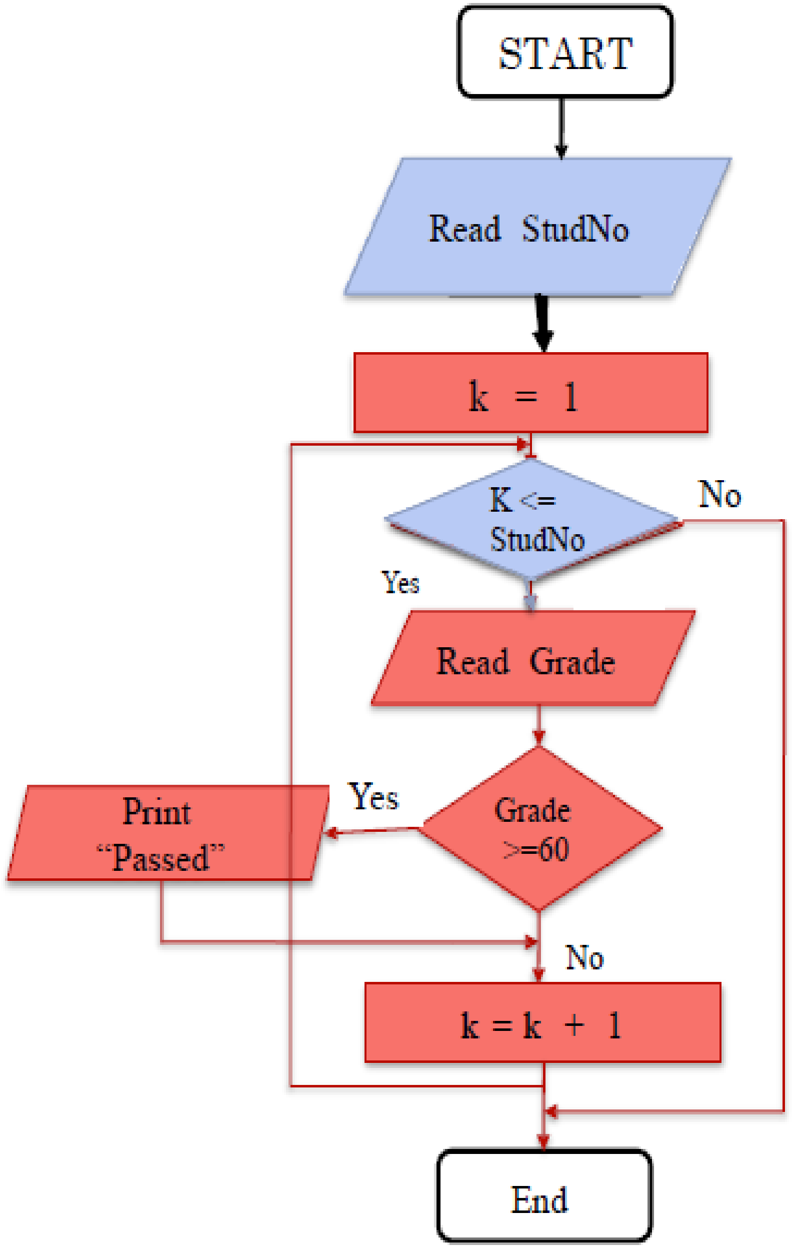
Example 10:

(READ DEFINED BY THE USER)

Repeat Example 9 but reads the number of students

Pseudocode

```
Read the Number of students (Stud No) initialize k=1
While k <=Stud No
Read Grade
If Grade >=60
Print “ Passed ”
End if
k=k+1
End
```

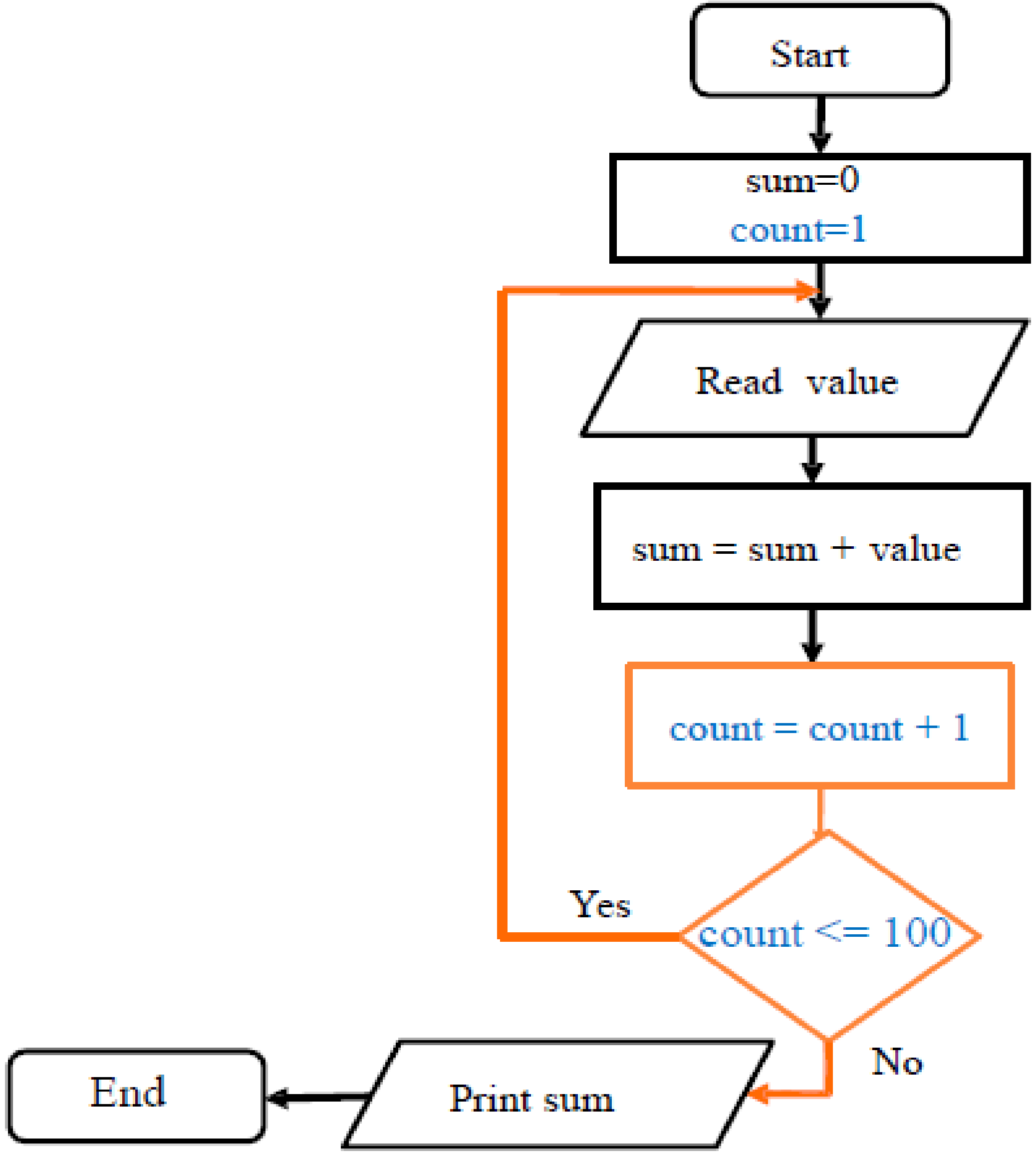


Example 11:

Summation of 100 values using do .. while

Pseudocode

```
Initialize sum=0
Initialize count=1
Do
Read value
Add value to sum
i.e. (sum=sum+value)
count=count+1
While (count<=100)
print
```



SENTINEL-CONTROLLED REPETITION

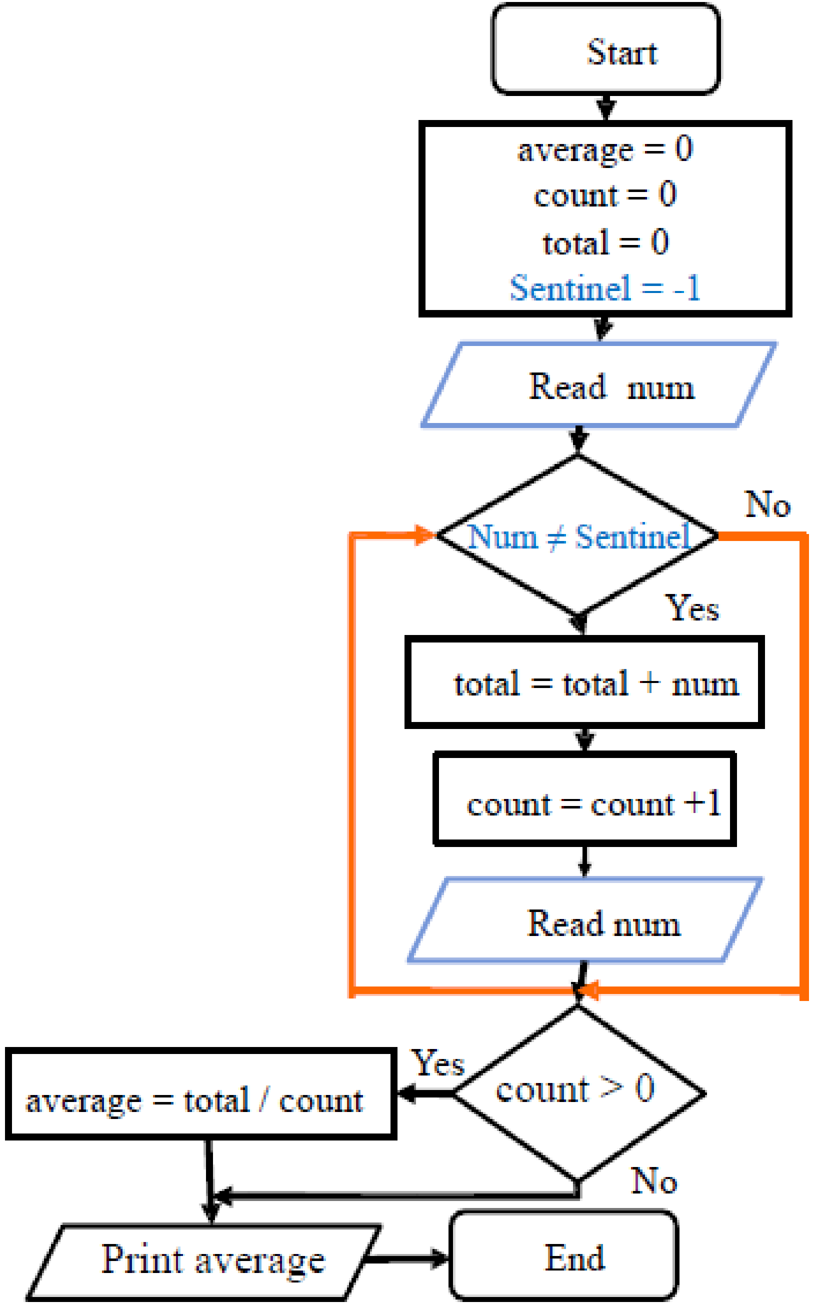
- If we do not know how many times we want to do repetition.
- It is dependent on the data provided to the program.
- “ **Sentinel Value** ”to indicate “end of data entry”
- E.g., enter -1 to end…

Example 12:

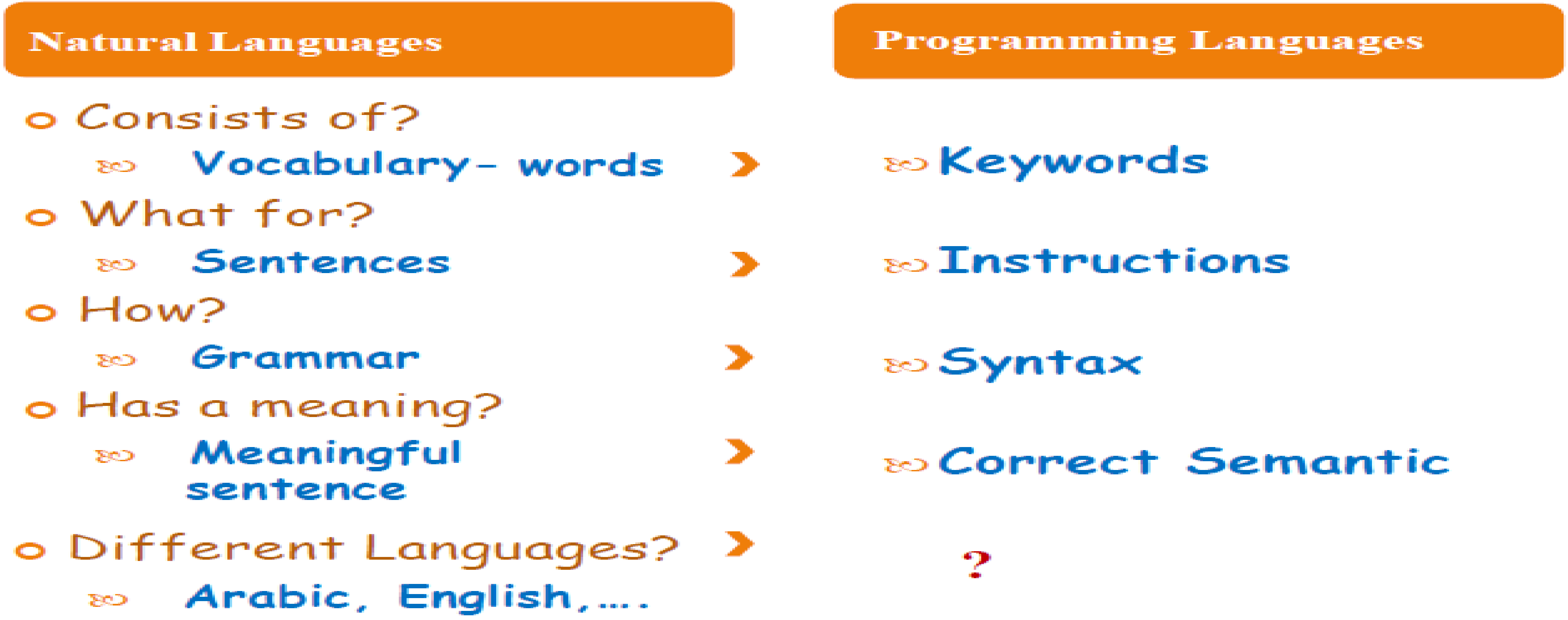
Read and compute the average of a set of numbers.

Pseudocode

```
set average to zero
set count to zero
set total to zero

read number
while ( not end-of-data )
    increment count by 1
    total = total + number
    read number

if ( count > 0 ) then
 average = total / count

display average
```



PROGRAMMING TERMINOLOGIES
Natural



Natural Languages	Programming Languages
Consists of?	
Vocabulary- words ›	Keywords
What for?	
Sentences ›	Instructions
How?	
Grammar ›	Syntax
Has a meaning?	
Meaningful sentence ›	Correct Semantic
Different Languages? ›	?
Arabic, English,….	

PROGRAMMING LANGUAGES

❑ Instructions can be written in various programming languages.

❑ Machine languages:

❑ Any computer can directly understand only its own machine language.

❑ Consist of numbers (0’s and 1’s).

❑ Assembly Languages:

❑ English like ***abbreviations*** to represent elementary operations.

❑ ***Assemblers*** were developed to convert assembly-language programs to machine language.

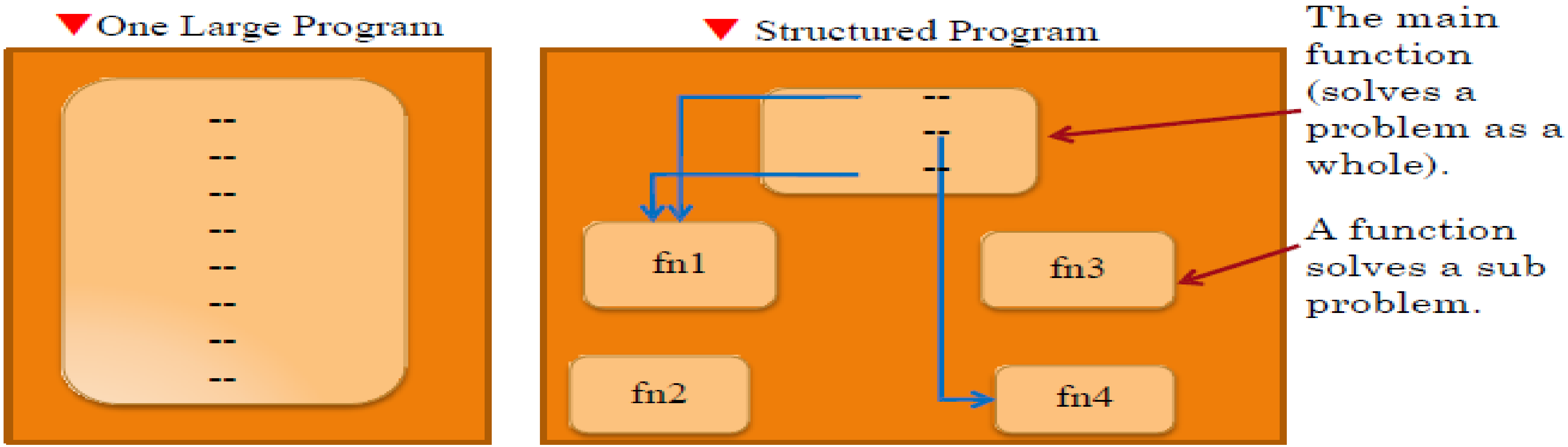
❑ High-Level Languages:

❑ ***Compilers*** convert high

STANDARD C

- ❑Standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C.
- ❑ISO (International standard Organization) in 1990 which was adopted by ANSI and is known as C89 or ANSI/ISO C.
- ❑Standard C.
- ❑What we will study?
- ❑As part of the normal evolution process the standard was updated in 1995 (C95) and 1999 (C99).

STRUCTURED PROGRAMMING



One Large Program

Structured Program

fn1

fn2

fn3

fn4

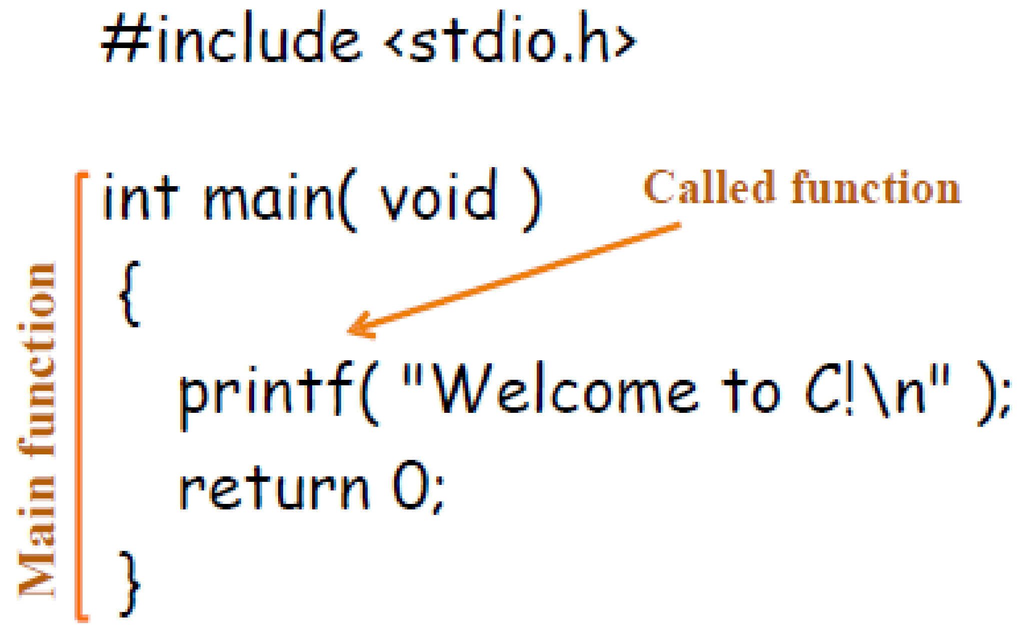
The main function (solves a problem as a whole).

A function solves a sub problem.

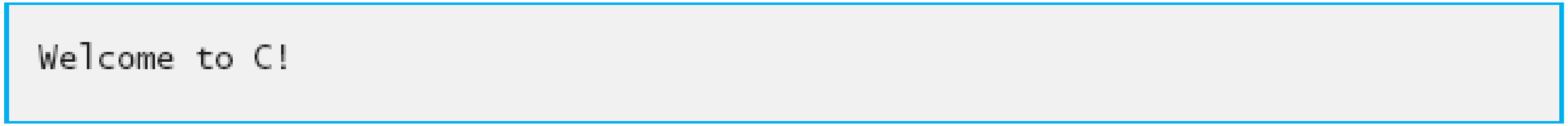
▶ **Easier to test, debug, modify & reuse.**

- Object Oriented Programming (OOP) concept ➔ next course (PL2)

YOUR FIRST PROGRAMINC

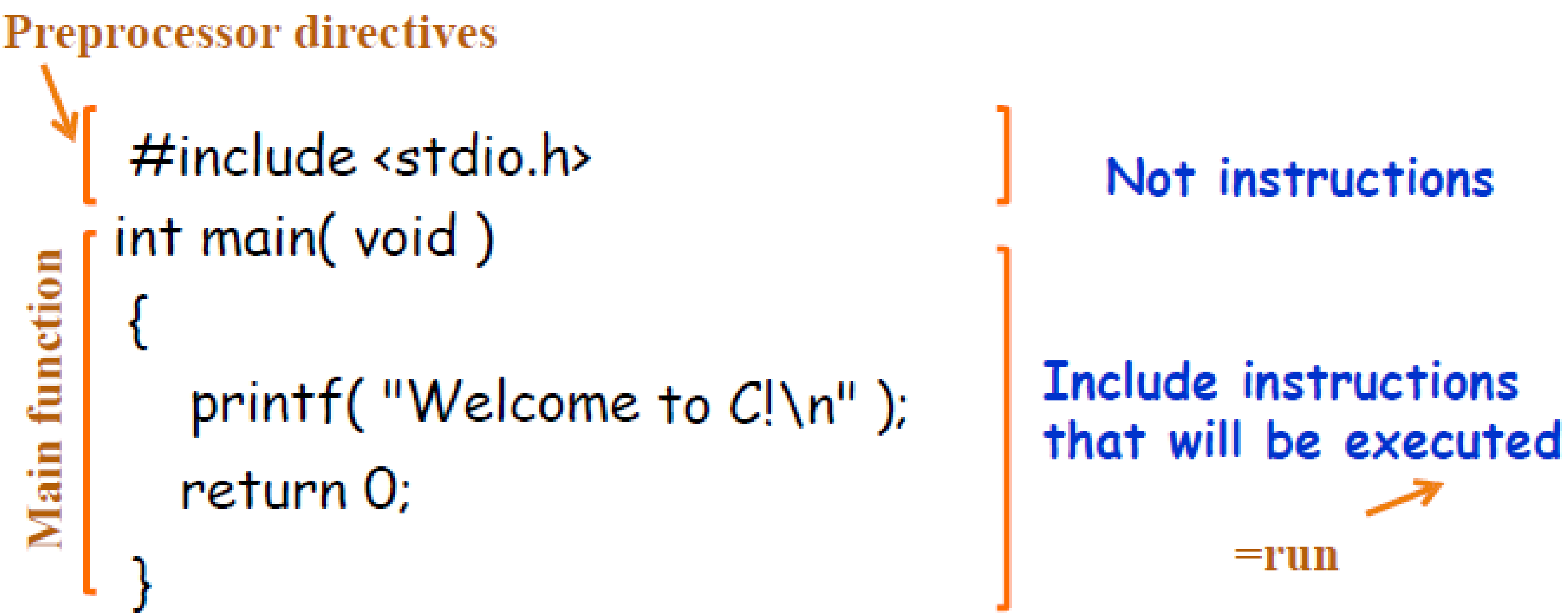


```
#include <stdio.h>

int main( void )
 {
   printf( "Welcome to C!\n" );
   return 0;
 }
```



```
Welcome to C!
```

A First Program in C



Preprocessor directives

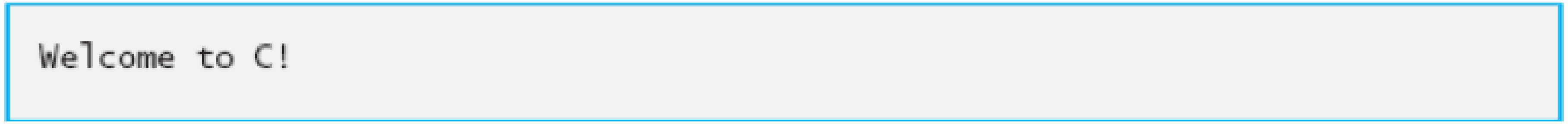
```
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

Not instructions

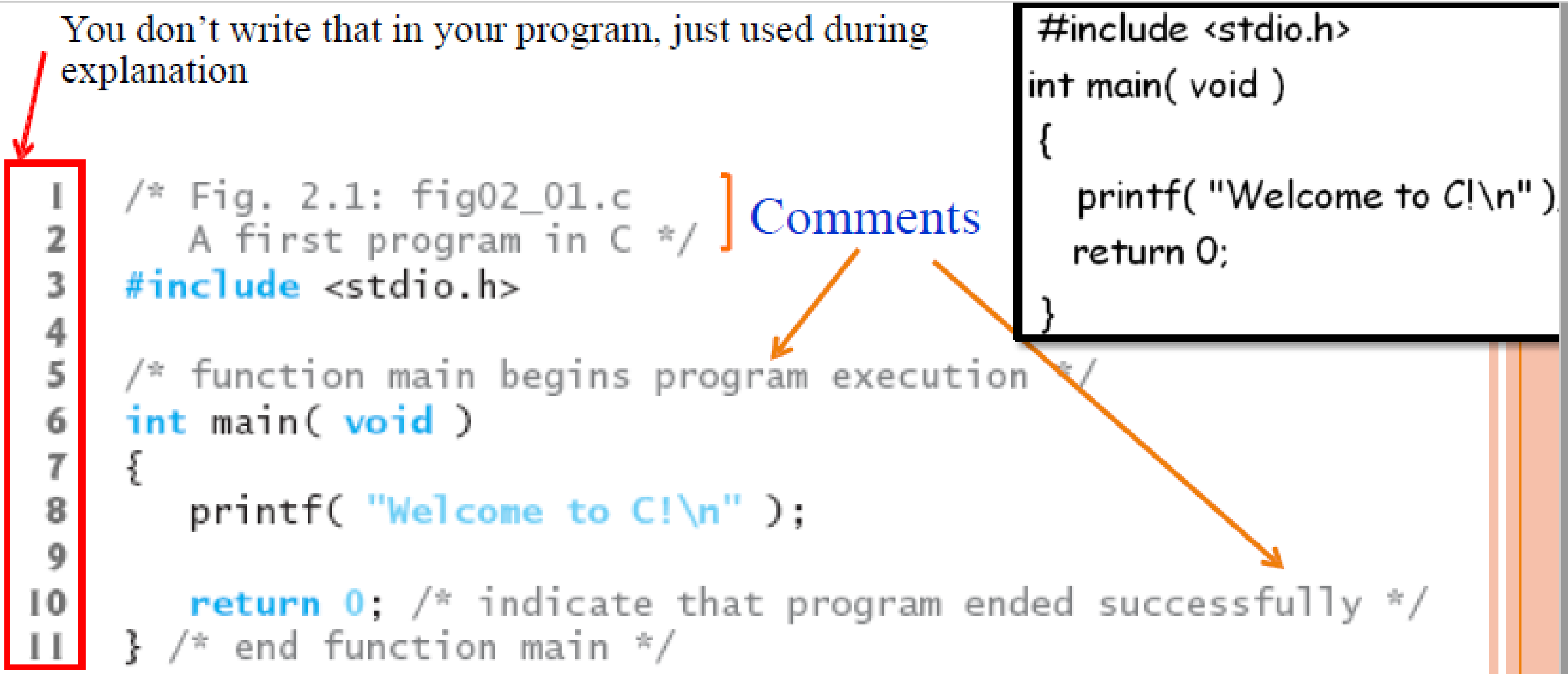
Main function

Include instructions that will be executed

=run



```
Welcome to C!
```



You don't write that in your program, just used during explanation

```c
 1  /* Fig. 2.1: fig02_01.c
 2     A first program in C */
 3  #include <stdio.h>
 4
 5  /* function main begins program execution */
 6  int main( void )
 7  {
 8     printf( "Welcome to C!\n" );
 9
10     return 0; /* indicate that program ended successfully */
11  } /* end function main */
```

Comments

```c
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

Comments:

Do not cause the computer to perform any action when the program is running;

➔ Do nothing

➔ Not instructions

Why Use Comments?

- To:
  - Improve the program's readability.
  - Help others read and understand your program.

- Two forms:

/*.. */ → for multiple-lined comments

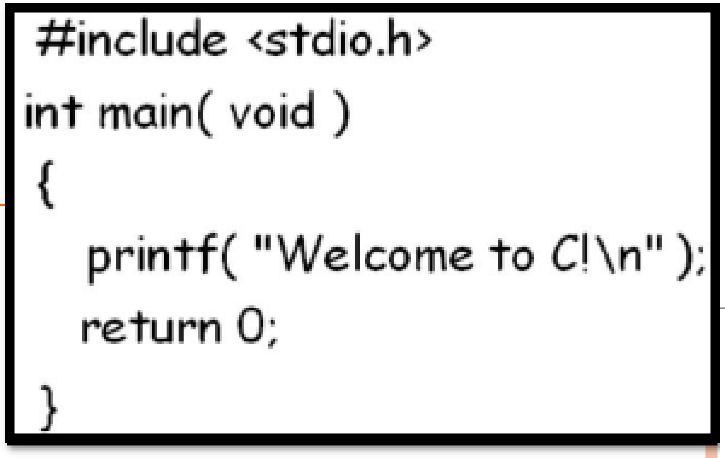
// → for single-line comments

Note: Blank lines, Spaces, tabs are ignored by the compiler

→ Only use them to improve a program's readability.

→ We can write more than an instruction in the same line, but this will affect our readability.

PREPROCESSOR DIRECTIVES



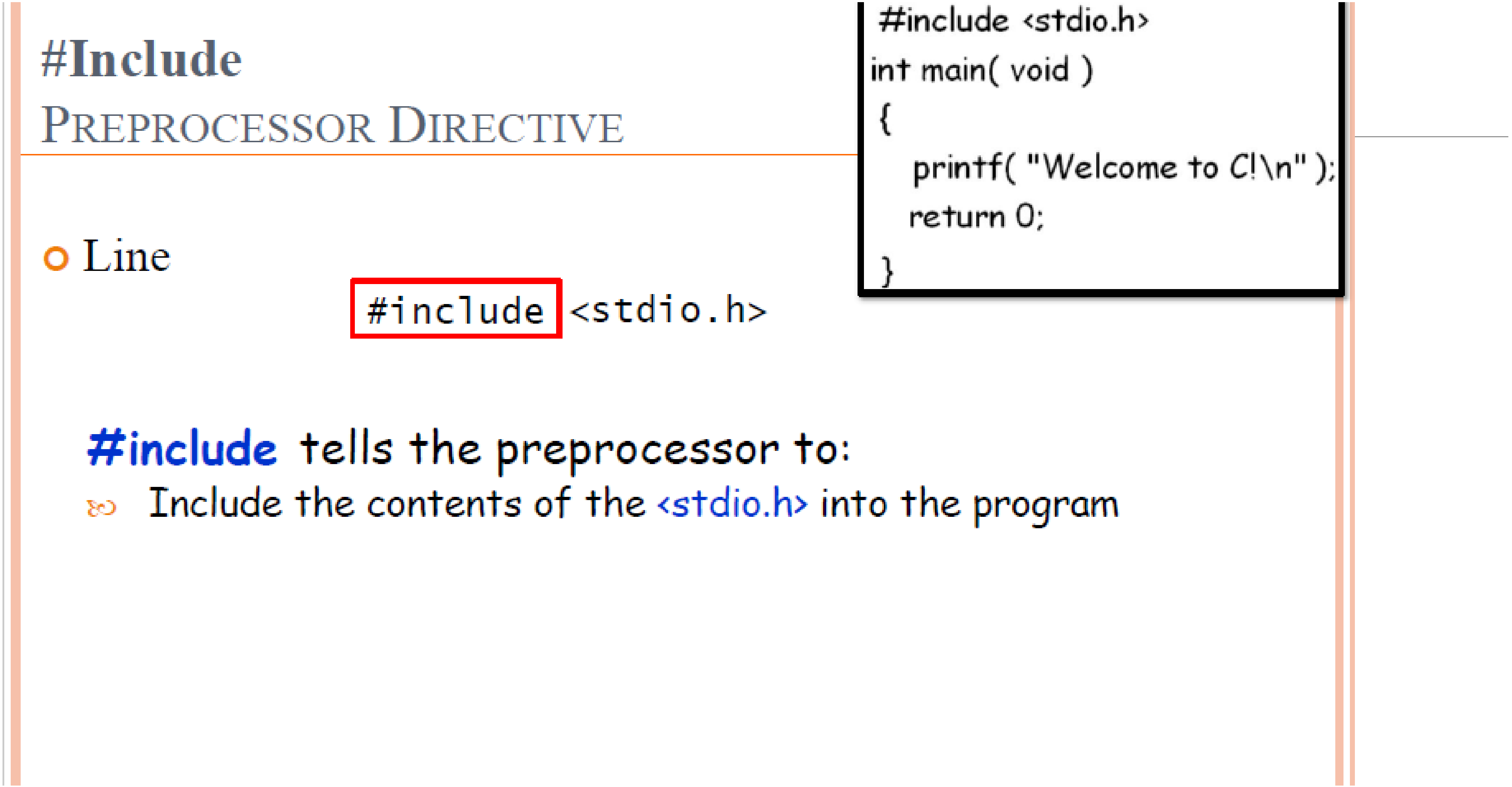
```
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

- Line

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

Lines start with # are:

- Directives to the C preprocessor.
- Processed before the program is compiled.
- Not instructions.



#Include

Preprocessor Directive

```
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

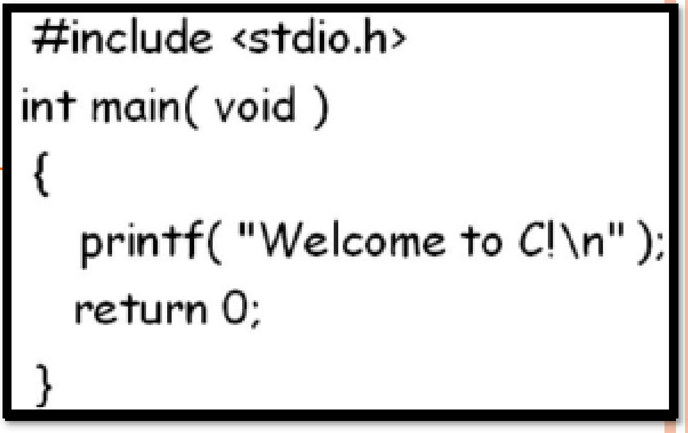
- Line

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

#include tells the preprocessor to:

- Include the contents of the <stdio.h> into the program

HEADER FILE



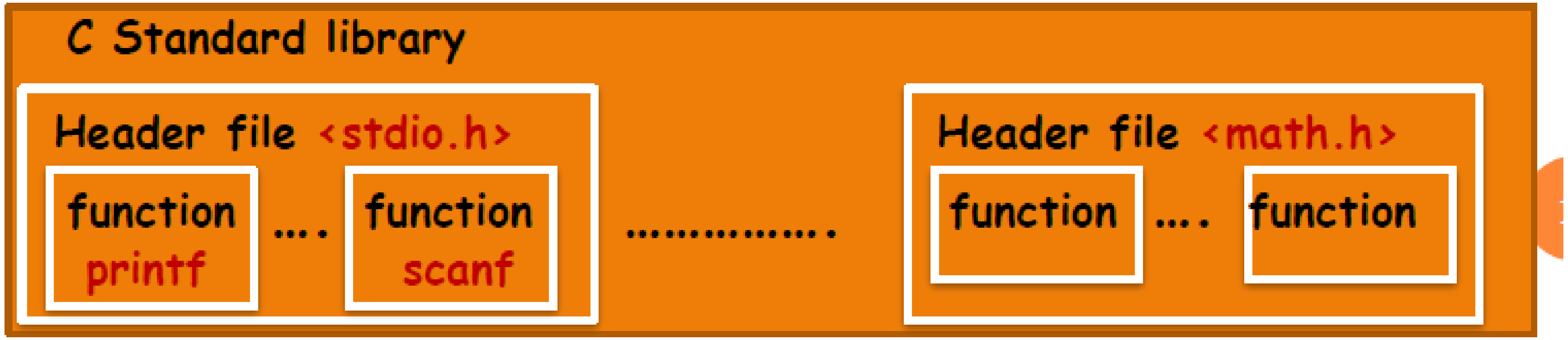
```
#include <stdio.h>
int main( void )
{
    printf( "Welcome to C!\n" );
    return 0;
}
```

- Line

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

<stdio.h> is:

- A header file contains information used by the compiler when compiling calls to library functions such as **printf**.
- Stands for Standard Input/Output header



C Standard library

Header file <stdio.h>

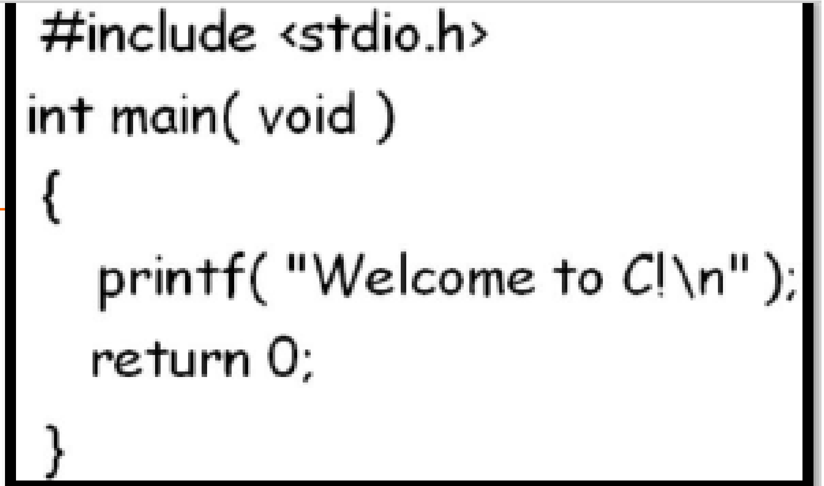
function printf .... function scanf

....

Header file <math.h>

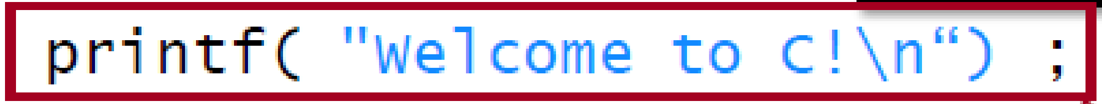
function .... function

A Statement



```
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

- Line



```
printf( "Welcome to c!\n“) ;
```

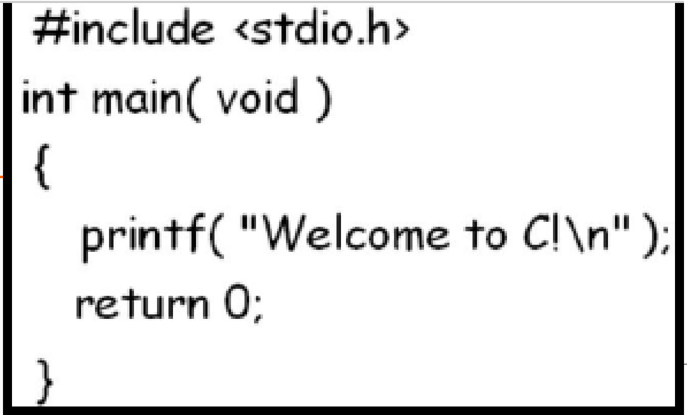
- An instruction = A statement

=

Instructs the computer to perform an action

- Every statement must end with a semicolon (;)

THE PRINTF FUNCTION



```
#include <stdio.h>
int main( void )
 {
    printf( "Welcome to C!\n" );
   return 0;
 }
```

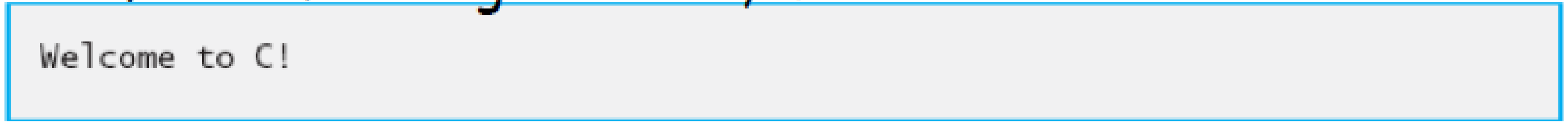
- Line

```
printf(“ Welcome to C!\n " );
```

- Print on the screen the string of characters between the double quotations (“”).

printf Syntax?

- After executing this line, we see:



```
Welcome to C!
```

- The **f** stands for “**formatted**”

Notice:

- The message appears as it's written between the quotations.
- “\n” doesn't appear ➔ formatting ➔ called “*escape sequence*”

Escape Sequences

Escape Sequences	Description
\n	Newline
\t	tab
\a	Alert (sound)
\\	Backslash
\"	Double quotation

- Do something out of the ordinary
- Examples:

```
printf ( "Welcome\n to\n C!\n");
```

```
Welcome
to
C!
```

```
printf ( "Welcome\t to  C!\n" );
```

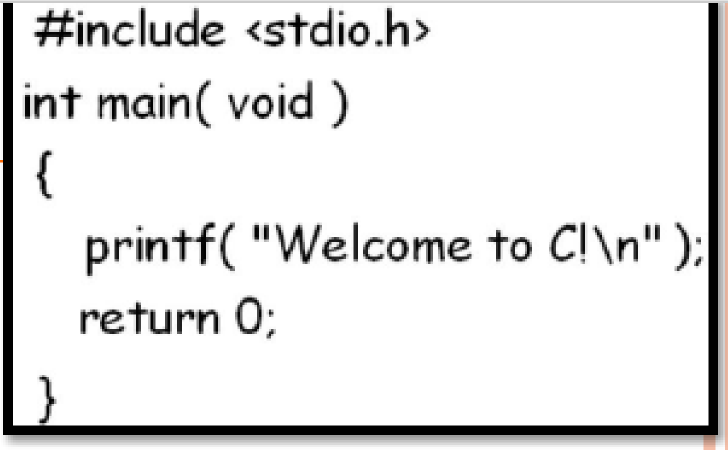
```
Welcome             to  C!
```

```
printf ( " Hello! \n" );
printf( "My name is \"Ahmed \" / Mohamed \\" );
```

```
 Hello!
My name is  "Ahmed" / Mohamed     \
```

We can print in several ways

THE RETURN STATEMENT



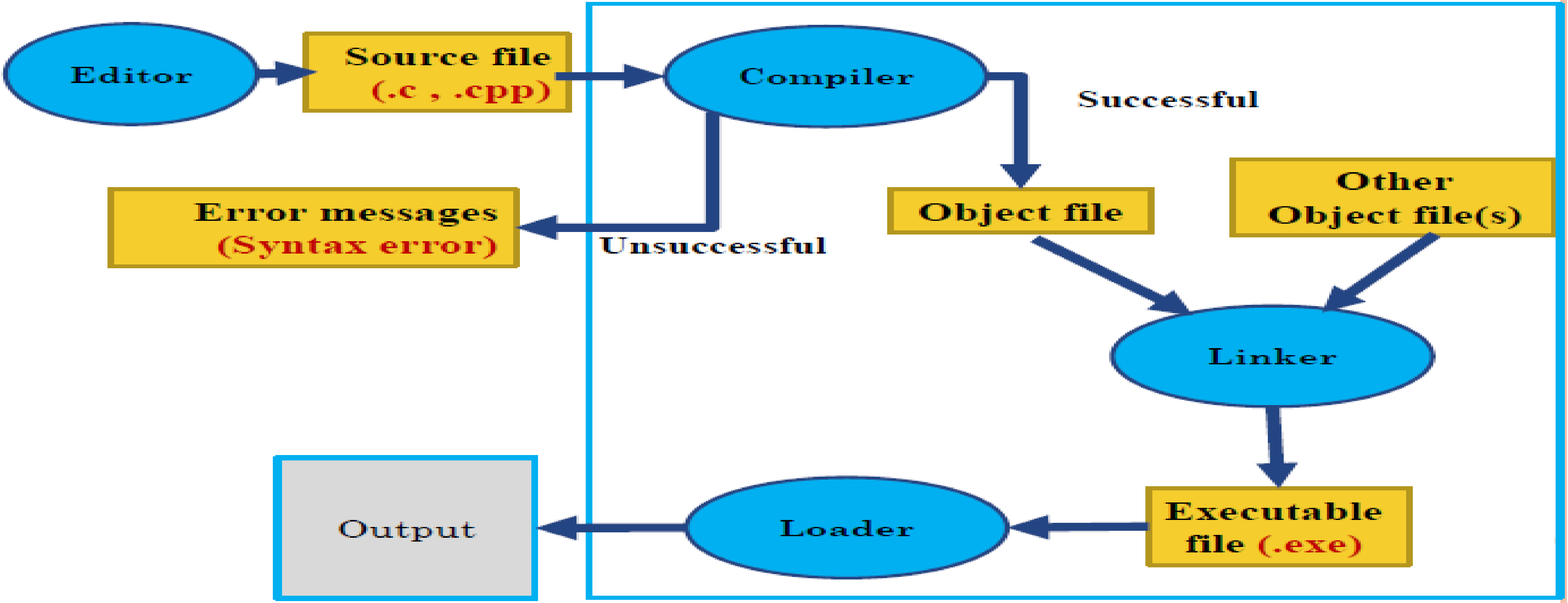
```
#include <stdio.h>
int main( void )
{
   printf( "Welcome to C!\n" );
   return 0;
}
```

- Line

```
        return 0 ;
```

- At the end of every `main` function.
- The keyword `return` is used to exit a function.
- The value `0` indicates that the program has terminated successfully.

TYPICALC PROGRAM DEVELOPMENT ENVIRONMENT



ERROR TYPES

❑**Syntax error** is a violation of the C grammar rules, detected during program compilation

E.g., a missing semicolon, a nun closed comment, spelling mistakes,…etc.

❑**Run-time error** is an attempt to perform an invalid operation, detected during program execution

E.g., Divide by zero,…etc.

❑**Semantic error** is an error caused by following an incorrect algorithm

Not the required output.

ANNOUNCEMENTS

❖ The **Lecture 7 & Lecture 8 Problem Solving** were posted Online Facebook Group last week. Please read them carefully.
❖**Sheet # 5** were posted online this week.
❖**Submissions of Sheet #5** is during next week's Labs
❖**Quiz # 4** will be held in the week after