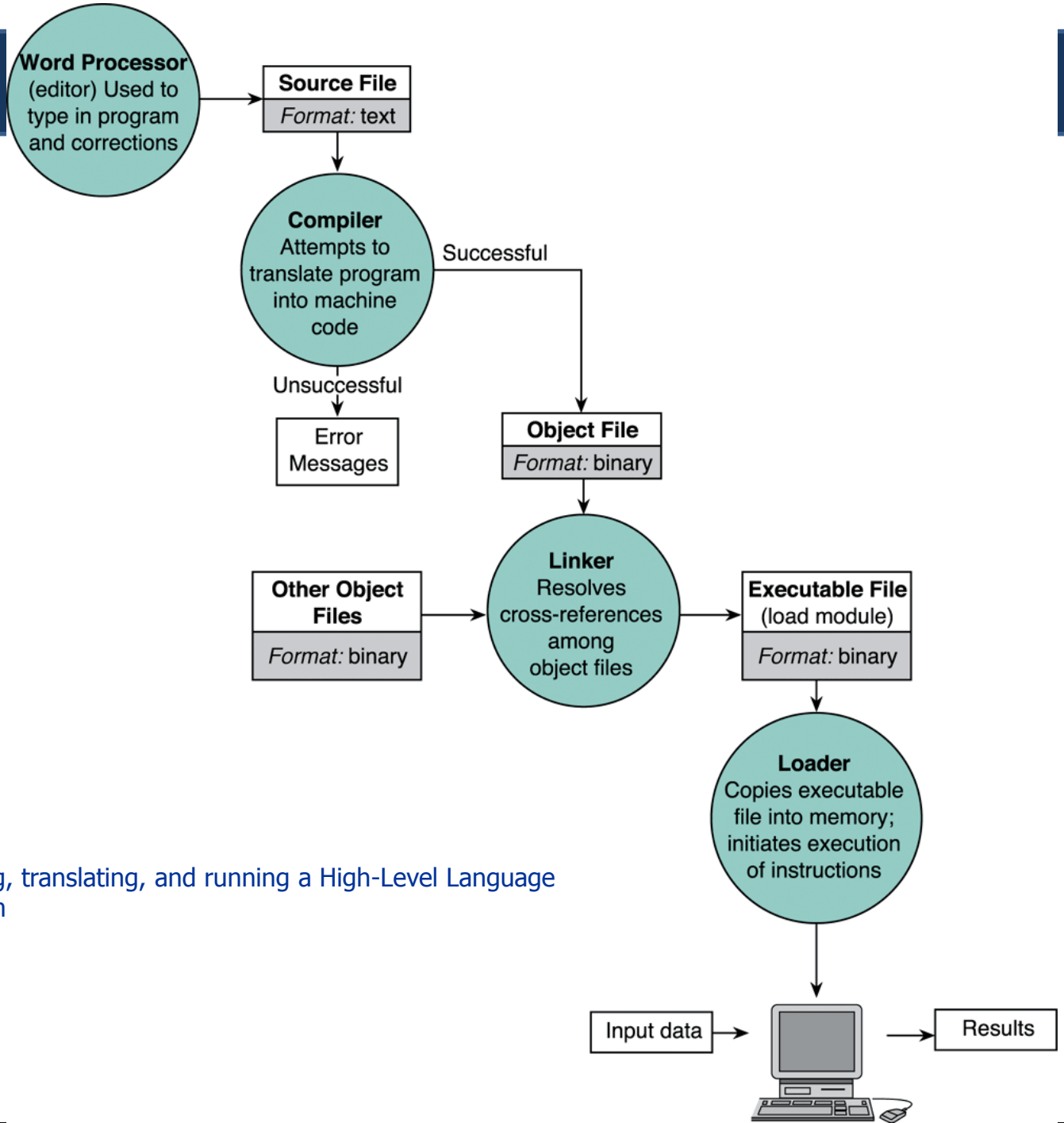
Chapter 3

Fundamental of C Programming Language
and
Basic Input/Output Function

Chapter 3: Fundamental of C and Input/Output

- In this chapter you will learn about:
  - C Development Environment
  - C Program Structure
  - Basic Data Types
  - Input/Output function
  - Common Programming Error



Entering, translating, and running a High-Level Language Program

C Program Structure

- An example of simple program in C

```
#include <stdio.h>

int main(void)
{
  printf("I love programming\n");
  printf("You will love it too once ");
  printf("you know the trick\n");
  return(0);
}
```

The output

- The previous program will produce the following output on your screen

```
I love programming
You will love it too once you know the trick
```

Preprocessor directives

- A C program line begins with # provides an instruction to the C preprocessor
- It is executed **before** the actual compilation is done.
- Two most common directives :
  - #include
  - #define
- In our example (#include<stdio.h>) identifies the ***header*** file for standard input and output needed by the printf().

Function main

- Identify the start of the program
- Every C program has a main ( )
- 'main' is a C ***keyword.*** We **must not** use it for any other variable.
- Using Visual Studio 2005, C program skeleton looks like this:

```
int main(void)
{

    return (0);
}
```

The curly braces { }

- Identify a ***segment / body*** of a program
  - The start and end of a function
  - The start and end of the selection or repetition block.

- Since the opening brace indicates the **start** of a segment with the closing brace indicating the **end** of a segment, **there must be just as many opening braces as closing braces** (this is a common mistake of beginners)

Statement

- A specification of an action to be taken by the computer as the program executes.
- Each statement in C needs to be terminated with semicolon (;)
- Example:

```
#include <stdio.h>

int main(void)
{
  printf("I" love programming\n");
  printf("You will love it too once ");
  printf("you know the trick\n");
  return (0);
}
```

statement

statement

statement

Statement cont…

- Statement has two parts :
  - Declaration
    - The part of the program that tells the compiler the names of memory cells in a program
  - Executable statements
    - Program lines that are converted to machine language instructions and executed by the computer

C program skeleton

- In short, the basic skeleton of a C program looks like this:

```
#include <stdio.h>

int main(void)
{
        statement(s);
        return(0);
}
```

Preprocessor directives

Function main

Start of segment

End of segment

Input/Output Operations

- Input operation
  - an instruction that copies data from an input device into memory
- Output operation
  - an instruction that displays information stored in memory to the output devices (such as the monitor screen)

Input/Output Functions

- A C function that performs an input or output operation
- A few functions that are pre-defined in the header file <stdio.h> such as :
  - printf()
  - scanf()
  - getchar() & putchar()

The printf function

- Used to send data to the standard output (usually the monitor) to be printed according to specific format.
- General format:
  - `printf("string literal");`
    - A sequence of any number of characters surrounded by double quotation marks.
  - `printf("format string", variables);`
    - Format string is a combination of text, conversion specifier and escape sequence.

The printf function cont…

- Example:
  - `printf("Thank you\n");`

```
Thank you
Press any key to continue
```

  - `printf ("Total sum is: %d\n", sum);`

```
Total sum is: 50
Press any key to continue
```

Assuming that the value of sum is 50

    - %d is a placeholder (conversion specifier)
      - marks the display position for a type integer variable
      - Common Conversion Identifier used in printf function.
    - \n is an escape sequence
      - moves the cursor to the new line

	printf
int	%d
float	%f
double	%f
char	%c
string	%s

Escape Sequence

Escape Sequence	Effect
\a	Beep sound
\b	Backspace
\f	Formfeed (for printing)
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\”	“ sign
\o	Octal decimal
\x	Hexadecimal
\O	NULL

Placeholder / Conversion Specifier

No	Conversion Specifier	Output Type	Output Example
1	%d	Signed decimal integer	76
2	%i	Signed decimal integer	76
3	%o	Unsigned octal integer	134
4	%u	Unsigned decimal integer	76
5	%x	Unsigned hexadecimal (small letter)	9c
6	%X	Unsigned hexadecimal (capital letter)	9C
7	%f	Integer including decimal point	76.0000
8	%e	Signed floating point (using e notation)	7.6000e+01
9	%E	Signed floating point (using E notation)	7.6000E+01
10	%g	The shorter between %f and %e	76
11	%G	The shorter between %f and %E	76
12	%c	Character	‘7’
13	%s	String	‘76'

The scanf function

- Read data from the standard input device (usually keyboard) and store it in a variable.
- General format:
  - `scanf("format string", &variable);`
- Notice ampersand (&) operator :
  - C address of operator
  - it passes the address of the variable instead of the variable itself
  - tells the scanf() where to find the variable to store the new value
- Format string is a combination of conversion specifier and escape sequence (if any).

The scanf function cont…

- Common Conversion Identifier used in printf and scanf functions.

	printf	scanf
int	%d	%d
float	%f	%f
double	%f	%lf
char	%c	%c
string	%s	%s

- Example :

```
int age;
printf("Enter your age:");
scanf("%d", &age);
```

The scanf function cont…

- If you want the user to enter more than one value, you serialise the inputs.
- Example:

```
float height, weight;

printf("Please enter your height and weight:");
scanf("%f%f", &height, &weight);
```

getchar() and putchar()

- getchar() - read a character from standard input
- putchar() - write a character to standard output
- Example:

```
Please type a character: h
You have typed this character: h
Press any key to continue
```

```
#include <stdio.h>
int main(void)
{
   char my_char;
   printf("Please type a character:");
   my_char = getchar();
   printf("\nYou have typed this character: ");
   putchar(my_char);
   return (0);
}
```

getchar() and putchar() cont

- Alternatively, you can write the previous code using normal printf / scanf and %c placeholder.
- Example

```
#include <stdio.h>
int main(void)
{
  char my_char;
  printf("Please type a character: ");
  scanf("%c",&my_char);
  printf("\nYou have typed this character: %c", my_char);
  return(0);
}
```

```
Please type a character: h
You have typed this character: h
Press any key to continue
```

Data Files

- Solutions to real problems often involve large amount of data that is not feasible to read from keyboard or print to the screen.
- To work with data file, we first have to define a file pointer to associate the file with

```
FILE *sensor;
```

- To open a file, fopen command is used.

```
sensor = fopen ("sensor.txt", "r");
```

- Options to open a data file:
  - r – read data
  - w – write data
  - a – append data
  - r+ - update
  - w+ - update, overwrite
  - a+ - update by appending

Data files

- To read data from the file, **f**scanf command is used

```
fscanf(sensor, "%lf %lf", &t, &motion);
```

- To print data to the file, **f**printf command is used

```
fprintf(waves, "%.2f %.2f %.2f \n", w1, w2, sum);
```

- Close the file at the end of program using fclose command

```
fclose(sensor);
```

- Other commands that can be used: fputc, fgetc, fputs, fgets

Few notes on C program…

- C is ***case-sensitive***
  - Word, word, WorD, WORD, WOrD, worD, etc are all different variables / expressions

    Eg. sum = 23 + 7
    - What is the value of Sum after this addition ?

- Comments (remember 'Documentation'; Chapter 2)
  - are inserted into the code using /* to start and */ to end a comment
  - Some compiler support comments starting with '//'
  - Provides supplementary information but is ignored by the preprocessor and compiler
    - `/* This is a comment */`
    - `// This program was written by Hanly Koffman`

C Token

- Tokens are a series of continuous characters that compilers treat as separate entities.
- Tokens can be classified into:
  1. Reserved words (also known as keywords)
  2. Identifiers
  3. Constants
  4. String Literal
  5. Punctuators
  6. Operators

1. Reserved Words

- Keywords that identify language entities such as statements, data types, language attributes, etc.
- Have special meaning to the compiler, cannot be used as identifiers (variable, function name) in our program.
- Should be typed in lowercase.
- Example: const, double, int, main, void, printf, while, for, else (etc..)

2. Identifiers

- Words used to represent certain program entities such as variables and function names.
- Example:
  - `int my_name;`
    - my_name is an identifier used as a program variable
  - `void CalculateTotal(int value)`
    - CalculateTotal is an identifier used as a function name

Rules for naming identifiers

Rules	Example
Can contain a mix of characters and numbers. However it **cannot** start with a number	H2o
First character must be a letter or underscore	Number1; _area
Can be of mixed cases including underscore character	XsquAre my_num
Cannot contain any arithmetic operators	R*S+T
… or any other punctuation marks…	#@x%!!
Cannot be a C keyword/reserved word	struct; printf;
Cannot contain a space	My height
… identifiers are **case sensitive**	Tax != tax

Variables

- Variable → a name associated with a memory cell whose value can change
- Variable Declaration: specifies the type of a variable
  - Example: `int num;`
- Variable Definition: *assigning* a value to the declared *variable*
  - Example: `num = 5;`

Basic Data Types

- There are 4 basic *data types* :
  - int
  - float
  - double
  - char
- int
  - used to declare numeric program variables of integer type
  - whole numbers, positive and negative
  - keyword: int

```
int number;
number = 12;
```

Basic Data Types cont…

- float
  - fractional parts, positive and negative
  - keyword: float

```
float height;
height = 1.72;
```

- double
  - used to declare floating point variable of higher precision or higher range of numbers
  - exponential numbers, positive and negative
  - keyword: double

```
double valuebig;
valuebig = 12E-3;
```

Basic Data Types cont…

- char
  - equivalent to ‘letters’ in English language
  - Example of characters:
    - Numeric digits: 0 - 9
    - Lowercase/uppercase letters: a - z and A - Z
    - Space (blank)
    - Special characters: , . ; ? “ / ( ) [ ] { } * & % ^ < > etc
  - single character
  - keyword: char

```
char my_letter;
my_letter = 'U';
```

The declared character must be enclosed within a single quote!

- In addition, there are void, short, long, etc data types.

3. Constants

- Entities that appear in the program code as fixed values.
- Any attempt to modify a CONSTANT will result in error.
- 4 types of constants:
  - Integer constants
    - Positive or negative whole numbers with no fractional part
    - Example:

      const int MAX_NUM = 10;

      const int MIN_NUM = -90;
  - Floating-point constants (float or double)
    - Positive or negative decimal numbers with an integer part, a decimal point and a fractional part
    - Example:

      const double VAL = 0.5877e2; (stands for 0.5877×10^2)

Constants cont…

- Character constants
  - A character enclosed in a single quotation mark
  - Example:
    - `const char letter = 'n';`
    - `const char number = '1';`
    - `printf("%c", 'S');`
      - Output would be: S

Constant example – volume of a cone

```
#include <stdio.h>

int main(void)
{
   const double pi = 3.142;
   double height, radius, base, volume;

   printf("Enter the height and radius of the cone:");
   scanf("%lf %lf", &height, &radius);

   base = pi * radius * radius;
   volume = (1.0/3.0) * base * height;

   printf("\nThe volume of a cone is %f ", volume);
   return (0);
}
```

#define

- You may also associate constant using #define preprocessor directive

```
#include <stdio.h>
#define pi 3.142

int main(void)
{
   double height, radius, base, volume;

   printf("Enter the height and radius of the cone:");
   scanf("%lf %lf", &height, &radius);

   base = pi * radius * radius;
   volume = (1.0/3.0) * base * height;

   printf("\nThe volume of a cone is %f ", volume);
   return (0);
}
```

4. String Literal

- A sequence of any number of characters surrounded by double quotation marks “ ”.
- Example of usage in C program:

```
printf("What a beautiful day.\n");
```

```
What a beautiful day.
Press any key to continue
```

- To have double quotation marks as part of the sentence, precede the quote with backslash

```
printf("He shouted \"stop!\" to the thief\n");
```

```
He shouted "stop!" to the thief.
Press any key to continue
```

5. Punctuators (separators)

- Symbols used to separate different parts of the C program.
- These punctuators include:

  [ ] ( ) { } , ; “ : * #
- Example:

```
#include <stdio.h>

int main (void)
{
    int num = 10;
    printf("%d",num);
    return (0);
}
```

6. Operators

- Tokens that result in some kind of computation or action when applied to variables or other elements in an expression.
- Example of operators:

```
* + = - / < >
```

- Usage example:

```
result = total1 + total2;
```

Common Programming Errors

- ***Debugging*** → Process removing errors from a program
- Three (3) kinds of errors :
  - Syntax Error
    - a violation of the C grammar rules, detected during program translation (compilation).
    - statement cannot be translated and program cannot be executed

Common Programming Errors

- Run-time errors
  - An attempt to perform an invalid operation, detected during program execution.
  - Occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.
  - The computer will stop executing the program, and displays a diagnostic message indicates the line where the error was detected

Common Programming Errors

- Logic Error/Design Error
  - An error caused by following an incorrect algorithm
  - Very difficult to detect - it does not cause run-time error and does not display message errors.
  - The only sign of logic error – incorrect program output
  - Can be detected by testing the program thoroughly, comparing its output to calculated results
  - To prevent – carefully desk checking the algorithm and written program before you actually type it

Summary

- In this chapter, you have learned the following items:
  - environment of C language and C programming
  - C language elements
    - Preprocessor directives, curly braces, main (), semicolon, comments, double quotes
  - 4 basics data type and brief explanation on variable
  - 6 tokens : reserved word, identifier, constant, string literal, punctuators / separators and operators.
  - printf, scanf, getchar and putchar
  - Usage of modifiers : placeholder & escape sequence
  - Common programming errors : syntax error, run-time error and logic error