Introduction to Computers & Programming

Comp 1330/ First Semester 2024/2025

Instructor: Saif Harbia

Faculty of Engineering and Technology
Department of Computer Science

Chapter 02

Overview of C

ABOUT C

1. *High-level programming language developed in 1972.*
2. *Was designed as a language to write the UNIX Operating System*
3. *Originally used primarily for systems programming*
4. *Become popular due to the power and flexibility of C, together with the availability of C-Compilers.*

2.1 C LANGUAGE ELEMENTS

1. Preprocessor Directives

- *Begins with a number symbol (#), i.e. #include, #define.*
- *Each library has a standard header file whose name ends with (.h).*
- *Descriptions of common mathematical functions are found in the header file* ***math.h***
- *Constant macros, i.e. #KMS_PER_MILE*

Figure 2.1 , 47

2. Comments

Give meaning to code, they are ignored by the C preprocessor and compiler

C LANGUAGE ELEMENTS

3. Function Main

Where program execution begins.
Comprised of declarations & executable statements

4. Reserved Words

Identifiers from standard libraries and names for memory cells.
Lower-cased.
Have special meaning in C
Table 2.1, p.50

C LANGUAGE ELEMENTS

5. Standard Identifiers

Have special meanings in C i.e. printf(), scanf()

6. User-Defined Identifiers

To name memory cells that will hold data and program results and to name operations that we define. (Ch03). i.e ***main, KMS_PER_MILE.***

1- Only letters, digits, and underscores.

2- Cannot begin with a digit.

3- Cannot be a C reserved word.

4- C- defined identifier should not be redefined.

VALID IDENTIFIERS

letter_1,
cent,
variable

letter_2,
CENT_PER_INCH,

inches,
Hello,

INVALID IDENTIFIERS

1Letter
int
joe's

double
TWO*FOUR

UPPERCASE AND LOWERCASE LETTERS

- Viewed as different identifiers by compiler,

  i.e. **Student != student != STUDENT**
- Maintain consistency as it improves program reading
- All reserved words in C and the names of all standard library functions use only lowercase letters.
- Uses all uppercase letters in the names of constant macros.

PROGRAM STYLE *Choosing Identifier Names*

1. Pick a meaningful name for a user-defined identifier, so its use is easy to understand, i.e. **salary,**
2. underscore between two words, i.e. **student_name**
3. Choose identifiers long enough to convey your meaning, but avoid excessively long names, i.e. **km_per_hr** instead of **kilometer_per_hour**
4. Do not choose names that are similar to each other, mistyping not detected if you accidentally type another word.

- Avoid using same word in uppercase and also lowercase, i.e. **PC and pc**.
- Also try not to use two names that differ only in the presence or absence of an underscore (**personal_computer and personalcomputer**).

2.2 VARIABLE DECLERATIONS AND DATA TYPES

1. Variable Declarations

Variables: The memory cells used for storing a program's input data and its computational results.

variable declarations:

- communicate to the C compiler the names of all variables used in a program.
- They also tell the compiler what kind of information will be stored in each variable.

double miles; /* input - distance in miles. */
double kms; /* output - distance in kilometers */

identifier

```
int count,
    large;
```

```
double rate, time;
```

2. Data Types

A **data type** is a set of values and a set of operations on those values: **ANSI**

1. **Integers (whole numbers)**: range of **int** must include at least the values -**32767** through **32767**. Store an integer in **int** variable and perform the common arithmetic operations, and compare two integers
2. **Double**: A real number has an **integral part** and **a fractional part** that are separated by a decimal point, perform arithmetic operations and compare.

from 10^{-37} -> 10^{37} (for positive values)

We can use scientific notation to represent real numbers:
Normal Scientific Notation: **1.23 X 10^5 =** 23000.0
C Scientific Notation: **1.23e5 or 1.23E5 (read e: "times 10 to the power")**

2. Data Types

TABLE 2.4 Type double Constants (real numbers)

Valid double Constants	Invalid double Constants
`3.14159`	`150` (no decimal point)
`0.005`	`.12345e` (missing exponent)
`12345.0`	`15e-0.3` (`0.3` is invalid exponent)
`15.0e-04` (value is `0.0015`)	
`2.345e2` (value is `234.5`)	`12.5e.3` (`.3` is invalid exponent)
`1.15e-3` (value is `0.00115`)	`34,500.99` (comma is not allowed)
`12e+5` (value is `1200000.0`)	

Differences between Numeric Types

- Double can be used to represent all numbers, but operations involving integers are faster (less storage).
- Operations with integers are always precise, whereas some loss of accuracy or **round-off error** may occur when dealing with type double numbers.
- This differences result from the way numbers are represented in memory.
- The binary string stored for type *int* value of some number is not the same as the binary string stored for the type *double* for the same number.
- The actual internal representation is computer dependent, and type double numbers usually require more bytes of computer memory than type int.

- real number = mantissa X $2^{exponent}$

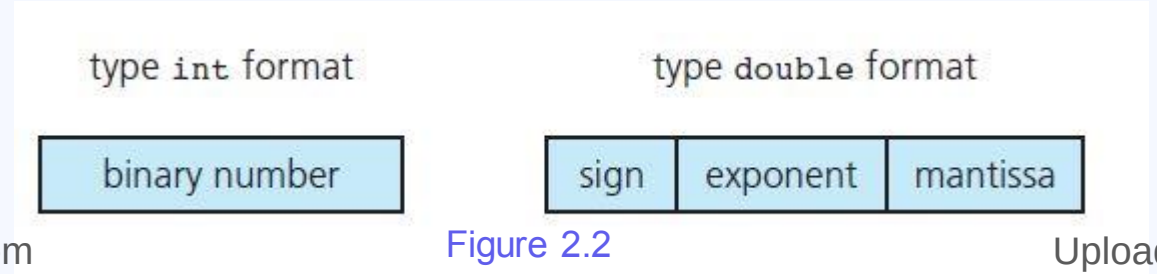


Figure 2.2

TABLE 2.5 Integer Types in C

Type	Range in Typical Microprocessor Implementation
`short`	–32,767 .. 32,767
`unsigned short`	0 .. 65,535
`int`	–2,147,483,647 .. 2,147,483,647
`unsigned`	0 .. 4,294,967,295
`long`	–2,147,483,647 .. 2,147,483,647
`unsigned long`	0 .. 4,294,967,295

Table 2.5

short <= int <= long

TABLE 2.6 Floating-Point Types in C

Type	Approximate Range*	Significant Digits*
`float`	$10^{-37} .. 10^{38}$	6
`double`	$10^{-307} .. 10^{308}$	15
`long double`	$10^{-4931} .. 10^{4932}$	19

*In a typical microprocessor-based C implementation

Table 2.6

Values of type **float must have at least six decimal digits of precision**;
both type **double and long double values must have at least ten decimal digits**

2. Data Types

3. char: Data type char represents an individual character value—a letter, a digit, or a special symbol. Each type char value is enclosed in apostrophes (single quotes) as shown here.

'A' ,'z' ,'2' ,'9' ,'*', ':' ,"" , ' '

- A character is represented in memory as an integer. The value stored is determined by the code used by your C compiler. The **ASCII code (American Standard Code for Information Interchange)** is the most common.

Printable characters vs non-printable characters

TABLE 2.7 ASCII Codes for Characters

Character	ASCII Code
' '	32
'*'	42
'A'	65
'B'	66
'Z'	90
'a'	97
'b'	98
'z'	122
'0'	48
'9'	57

The digit characters '0' through '9' have code values of 48 through 57 (decimal).

```
'0' < '1' < '2' < '3' < '4' < '5' < '6' < '7' < '8' < '9'
```

In ASCII, uppercase letters have the decimal code values 65 through 90.

```
'A' < 'B' < 'C' < ... < 'X' < 'Y' < 'Z'
```

Lowercase letters have the consecutive decimal code values 97 through 122

```
'a' < 'b' < 'c' < ... < 'x' < 'y' < 'z'
```

2.3 EXECUTABLE STATEMENTS

C statements used to write or code the algorithm and its refinements.

Programs in memory

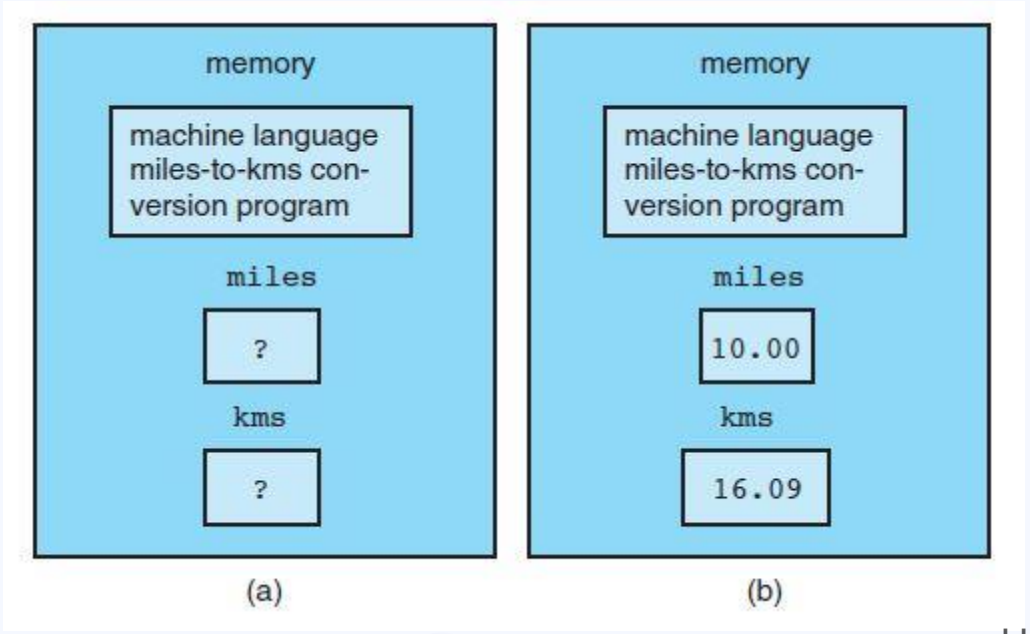


Figure 2.3

Assignment Statements

kms = KMS_PER_MILE * miles;

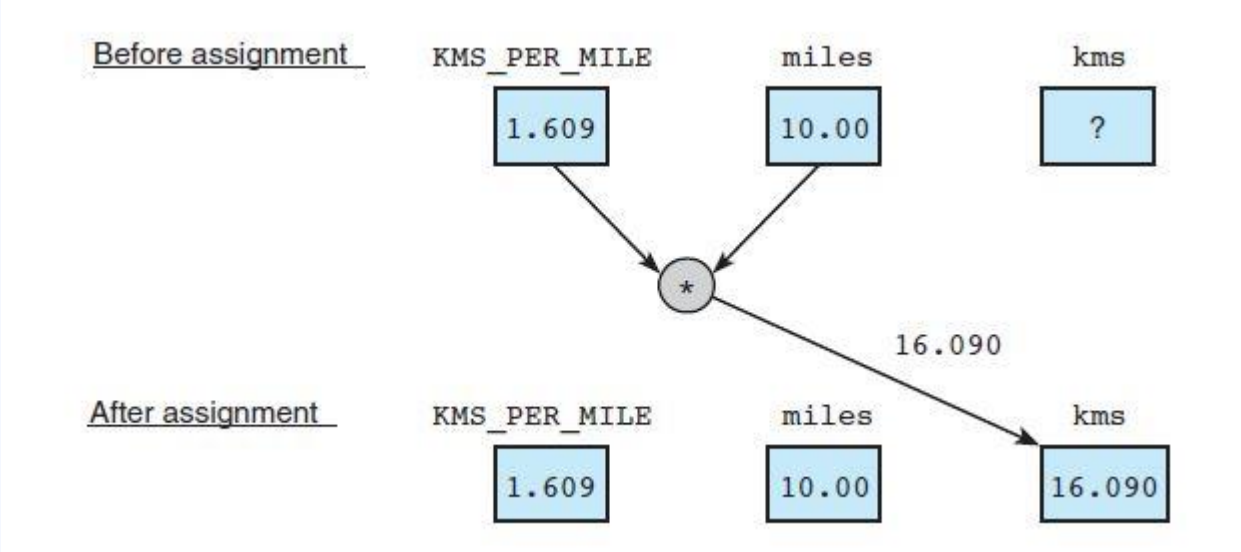


Figure 2.4

FORM: *variable = expression;*

EXAMPLEs:

```
x = y + z + 2.0;
sum = sum + item; (not an algebraic equation)
new_x = x;
new_x = -x;
```

Assignment to char variable:

```
next_letter = 'A';
```

Input/Output Operations and Functions

All input/output operations in C are performed by special program units called **input/output functions .**

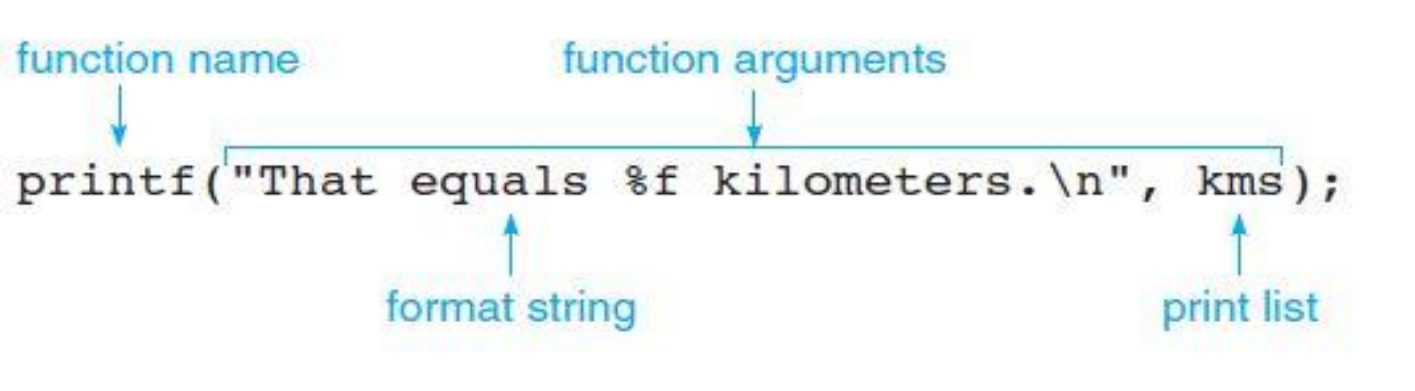
To store data in memory:

1. By assignment to a variable
2. By copying the data from an input device into a variable using a function like **scanf** (**input operation)**

- Program results can be displayed to the program user by an **output operation** .

- Most common **input/output** functions provided by **#include <stdio.h> (input \output library)**

The Printf Function



Result: **That equals 16.090000 kilometers.**
--newline

Placeholders

TABLE 2.8 Placeholders in Format Strings

Placeholder	Variable Type	Function Use
`%c`	`char`	`printf/scanf`
`%d`	`int`	`printf/scanf`
`%f`	`double`	`printf`
`%lf`	`double`	`scanf`

Multiple Placeholders

```
printf("Hi %c%c%c - your age is %d \n", letter_1, letter_2, letter_3, age);
```

Result: **Hi ABC - your age is 35**

--newline

New line character “\n”

```
printf("Here is the first line\n");
printf("\nand this is the second.\n");
```

produce two lines of text with a blank line in between:

```
Here is the first line

and this is the second.
```

```
printf("This sentence appears \non two lines.\n");
```

the characters after the \n appear on a new output line:

```
This sentence appears
on two lines.
```

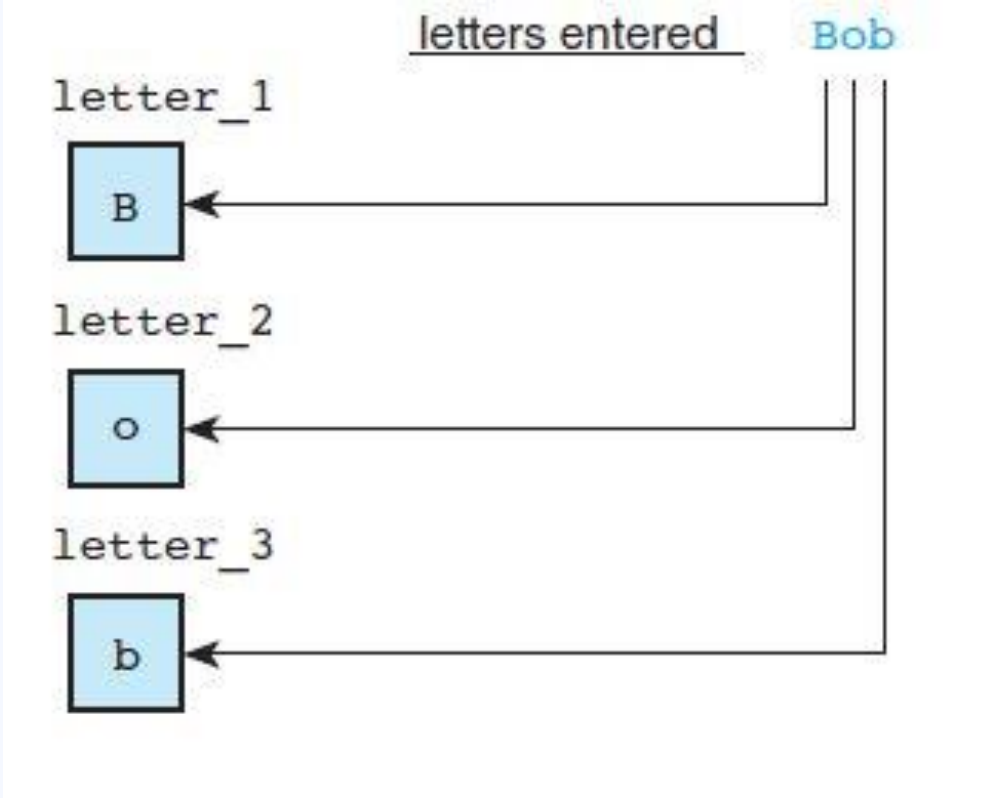
Prompting Message (prompts)

```
printf("Enter the distance in miles> ");
scanf("%lf", &miles);
```

→ No Print List

- The & is the C *address-of* operator: find each variable in which to store a new value.
- When **scanf** executes, the program pauses until the required data are entered and the <return> or <enter> key is pressed.

scanf("%c%c%c", &letter_1, &letter_2, &letter_3);



Scanf() rules

- The number of input characters consumed by the **scanf** function depends on the current format placeholder, which should reflect the type of the variable in which the data will be stored.
- Only one input character is used for a %c (type char variable).
- For a %lf or %d (type double or int variable), the program first skips any spaces and then scans characters until it reaches a character that cannot be part of the number. Usually the program user indicates the end of a number by pressing the space bar or by pressing the <return> or <enter> key.

Scanf() rules

- If you would like scanf to skip spaces before scanning a character, put a blank in the format string before the %c placeholder.
- If you type more data characters on a line than are needed by the current call to scanf , the extra characters will be processed by the next call to scanf .
- You must enter data in the same order as the variables in the input list.
- You should insert one or more blank characters or carriage returns between numeric items.

Return statement

- **return (0);**
- **The return statement transfers control from a function back to the activator of the function.**
- **For function main , control is transferred back to the operating system.**
- **The value in parentheses,0, is considered the result of function main ’s execution.**
- **It indicates that your program executed without error.**

2.4 GENERAL FORM OF A C PROGRAM

preprocessor directives
main function heading
{
declarations
executable statements
}

You can write more than one statement on a line. For example, the line:

printf("Enter distance in miles> "); scanf("%lf", &miles);

Program Style

- **Spaces:** for readability (required between consecutive words in a program line.)
- **Comments:** Comments are part of the **program documentation** because they help others read and understand the program (ignored by the compiler).

```
double miles,  /* input - distance in miles      */
       kms;    /* output - distance in kilometers */
```

Each program should begin with a header section that consists of a series of comments specifying:

- the programmer's name
- the date of the current version
- a brief description of what the program does

2.5 ARITHMETIC EXPRESSIONS

- To manipulate type **int** and **double** data

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
-	subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

Operator /

- 7.0 / 2.0 is 3.5 , but the value of 7 / 2 is 3.
- 299.0 / 100.0 is 2.99 , but the value of 299 / 100 is
- If the / operator is used with a negative and a positive integer, the result may vary from one C implementation to another. For this reason, **you should avoid using division with negative integers**.
- The / operation is undefined when the divisor (the second operand) is 0 .

TABLE 2.10 Results of Integer Division

`3 / 15 = 0`	`18 / 3 = 6`
`15 / 3 = 5`	`16 / -3` varies
`16 / 3 = 5`	`0 / 4 = 0`
`17 / 3 = 5`	`4 / 0` is undefined

Operator %

- The remainder operator (%) returns the integer remainder of the result of dividing its first operand by its second.

TABLE 2.11 Results of % Operation

`3 % 5 = 3`	`5 % 3 = 2`
`4 % 5 = 4`	`5 % 4 = 1`
`5 % 5 = 0`	`15 % 5 = 0`
`6 % 5 = 1`	`15 % 6 = 3`
`7 % 5 = 2`	`15 % -7` varies
`8 % 5 = 3`	`15 % 0` is undefined

Operator %

- The magnitude of *m* % *n* must always be less than the divisor *n,* so if *m* is positive, the value of *m* % 100 must be between 0 and 99 .
- The % operation is undefined when *n* is zero and varies from one implementation to another if *n* is negative.

The formula: *m equals (m / n)* n (m % n)*

***7 equals* (7 / 2) * 2 + (7 % 2)**
***equals* 3 * 2 +** 1

Data Type of an Expression

- The data type of an expression depends on the type(s) of its operands.
- X + y = > integer if both are integers, otherwise it is of type double
- An expression that has operands of both type int and double is a **mixed-type expression**

Mixed-Type Assignment Statement

- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=).
- Either a type double or a type int expression may be assigned to a type double variable,
- m = 3; n = 2; (integers) x ,y (doubles)

1. y = m / n; ⟶ (m / n = 1 => y = 1.0)

2. x = 9 * 0.5; => 4.5
3. n = 9 * 0.5; => 4

Type Conversion through Casts

- n = (int)(9 * 0.5);

TABLE 2.12 Examples of the Use of Type Casts

Application	Example	Explanation

Avoiding integer division	`int num_students; /* number of students who took a test */`
	`int total_score; /* total of all students' test scores */`
	`double average;`
	`average = (double)total_score / (double)num_students;`
		If the assignment statement were written
		`average = (double) (total_score / num_students);`
		integer division would cause the loss of the fractional part of the average.

Rounding a positive number	`double x;`
	`int rounded_x;`
	`/* code to give x a value omitted */`
	`rounded_x = (int)(x + 0.5);`
		Consider cases when x's fractional part is greater than or equal to 0.5, and cases when it is less. On the left we see how 35.51 is rounded to 36; on the right how 35.12 is rounded to 35.
		35.51 +0.50 = 36.01
		35.12 +0.50 = 35.62

Arithmetic operations on characters

- 'A' + 1 = B
- The expression 'A' + 1 adds 1 to the code for 'A' and its value is the next character after 'A‘ which is 'B' in ASCII

Expressions with Multiple Operators

- **Unary operators** take only one operand: x = -y;
- **Binary operators** require two operands: x = y + z;

- What about **multiple operators**?

x + y / z

x / y * z

To understand and write expressions with multiple operators, we must know the C rules for evaluating expressions

Expressions with Multiple Operators

- **Rules for Evaluating Expressions**:

1. Parentheses rule.
2. Operator precedence rule:
   - unary +, - first
   - *, /, % next
   - binary +, - last
3. Associativity rule

- Right-to-Left Associativity (Unary Operators):

```
int x = -+5;
```

- Left-to-Right Associativity (Binary Operators):

```
int result = 5 - 3 + 2;
```

Expressions with Multiple Operators

- **Rules for Evaluating Expressions**:

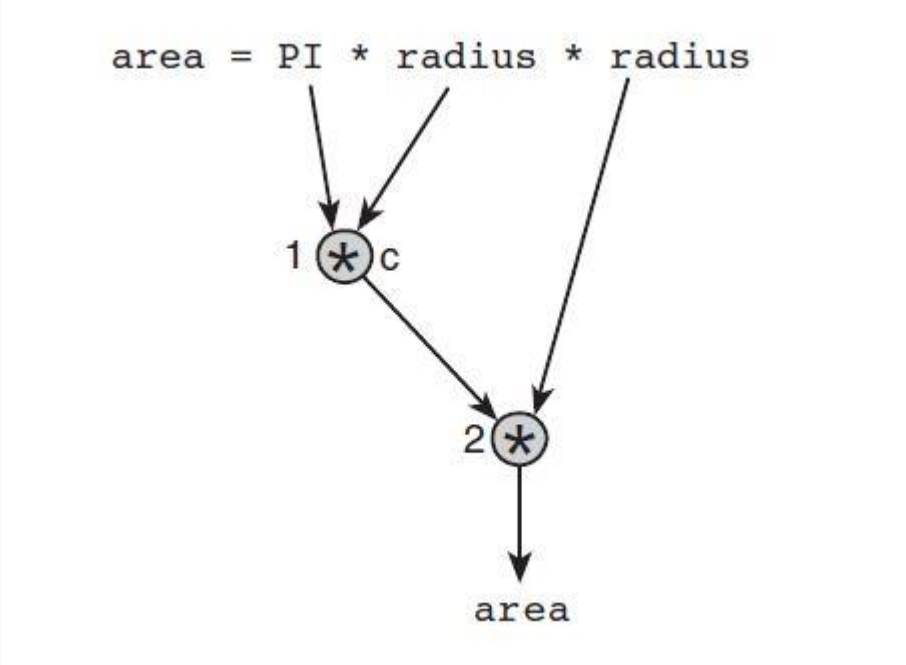
For example, the expression

x * y * z + a / b - c * d

can be written:

(x * y * z) + (a / b) - (c * d)

Example 1



Step-by-step evaluation

```
area     =      PI     *    radius    *    radius
                3.14159          2.0           2.0
                      6.28318
                                12.56636
```

Writing Mathematical Formulas in C

TABLE 2.13 Mathematical Formulas as C Expressions

Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a + b}{c + d}$	`(a + b) / (c + d)`
4. $\frac{1}{1 + x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Numerical Inaccuracies

- **The representational error** (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error:** When you add a large number and a small number, the larger number may “cancel out” the smaller number.

- For example: **1000.0 + 0.0000001234 is equal to 1000.0** on some computers.

Numerical Inaccuracies

- **arithmetic underflow:** If two very small numbers are multiplied, the result may be too small to be represented accurately, so it will be represented as zero
- **Arithmetic overflow:** Similarly, if two very large numbers are multiplied, the result may be too large to be represented.

CASE STUDY (Homework) P.82

2.6 FORMATTING NUMBERS IN PROGRAM OUTPUT

- **Formatting Values of Type int:**

- add a number between the **%** and the **d** of the **%d placeholder** in the **printf** format string.
- This number specifies the **field width** (the number of columns to use for the display of the value.)

printf("Results: %3d meters = %4d ft. %2d in.\n", meters, feet, inches);

If meters is **21** , feet is **68** , and inches is **11** , the program output will be:

Results: 21 meters = 68 ft. 11 in.

1 extra space

2 extra spaces

- **Formatting Values of Type int:**

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
234	%4d	␣234	-234	%4d	-234
234	%5d	␣␣234	-234	%5d	␣-234
234	%6d	␣␣␣234	-234	%6d	␣␣-234
234	%1d	234	-234	%2d	-234

- **Formatting Values of Type double:**

- To describe the format specification for a type *double* value, we must indicate both the total field width needed and the number of decimal places desired.

- The total field width should be large enough to accommodate all digits before and after the decimal point.

- There will be at least one digit before the decimal point because a zero is printed as the whole-number part of fractions that are less than 1.0 and greater than –1.0.

- **Formatting Values of Type double:**

- We should also include a display column for the decimal point and for the minus sign if the number can be negative.

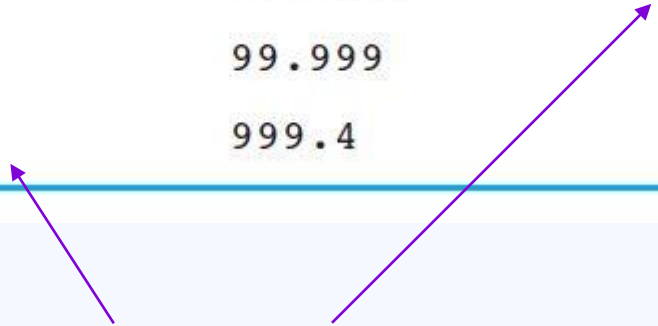
- The form of the format string placeholder is **%n.mf** where n is a number representing the total field width, and m is the desired number of decimal places.

- If x is a type double variable whose value will be between -99.99 and 999.99 , we could use the placeholder **%6.2f** to display the value of x to an accuracy of **two decimal places**

- **Formatting Values of Type double:**

TABLE 2.15 Displaying x Using Format String Placeholder %6.2f

Value of x	Displayed Output	Value of X	Displayed Output
-99.42	-99.42	-25.554	-25.55
.123	▮▮0.12	99.999	100.00
-9.536	▮-9.54	999.4	999.40



- **Formatting Values of Type double:**

TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	%5.2f	▮3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	▮▮3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	▮3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	▮▮-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

Total field width omitted

- **Formatting Values of Type double:**

- **Remarks:**

- a value whose whole-number part requires fewer display columns than are specified by the format field width is displayed with leading blanks.

- To eliminate extra leading blanks, omit the field width from the format string placeholder.

- The simple placeholder %d will cause an integer value to be displayed with no leading blanks.

- A placeholder of the form % .mf has the same effect for values of type double, and this placeholder still allows you to choose the number of decimal places you wish.

2.7 INTERACTIVE MODE, BATCH MODE, AND DATA FILES

- **two basic modes of computer operation:**

1. **Interactive mode:** the program user interacts with the program and types in data while it is running.
2. **batch mode**: the program scans its data from a data file prepared beforehand instead of interacting with its user.

- **Batch Mode:**
  - **Input Redirection:**
  - We assume here that the standard input device is associated with a batch data file instead of with the keyboard. *(figure 2.14 p.92)*
  - **i.e:** myprog <mydata.txt
  - Whenever you convert an interactive program to a batch program make sure you replace each prompt with an echo print that follows the call to *scanf*:

```
printf("Enter the distance in miles> ");

scanf("%lf", &miles);

printf("The distance in miles is %.2f.\n", miles);
```

- **Batch Mode:**

  - **Output Redirection**

  - You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer

  - redirect output from the screen to file myoutput.txt:

myprog >myoutput.txt => myprog <mydata.txt >myoutput.txt

2.8 COMMON PROGRAMMING ERRORS

- Errors are so common that they have their own special name— bugs —and the process of correcting them is called “debugging” a program.
- When the compiler detects an error, the computer displays an error message, which indicates that you have made a mistake and what the likely cause of the error might be.
- Unfortunately, error messages are often difficult to interpret and are sometimes misleading.
- Three kinds of errors—**syntax errors, run-time errors, and logic errors**

1. syntax errors

- occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- *Figure 2.15 p. 95.*
- The compiler attempts to correct errors wherever it can. Look at line 271.
- We see several cases in this listing where one mistake of the programmer leads to the generation of multiple error messages (the missing declaration for variable miles).
- It is often a good idea to concentrate on correcting the errors in the declaration part of a program first. Then recompile the program before you attempt to fix other errors.
- Syntax errors are often caused by the improper use of quotation marks with format strings.

2. Run-Time Errors

- **Detected and displayed by the computer during the execution of a program.**
- **A run-time error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.**
- **When a run-time error occurs, the computer will stop executing your program and will display a diagnostic message that indicates the line where the error was detected.**
- *Figure 2.16 p.96*

3. Logic Errors

- occur when a program follows a faulty algorithm.
- Because logic errors usually do not cause run-time errors and do not display error messages, they are very difficult to detect.
- The only sign of a logic error may be incorrect program output.
- You can detect logic errors by testing the program thoroughly, comparing its output to calculated results.
- You can prevent logic errors by carefully desk checking the algorithm and the program before you type it in.

Undetected Errors

- Errors that may not prevent a C program from running to completion, but they may simply lead to incorrect results.
- Therefore, it is essential that you predict the results your program should produce and verify that the actual output is correct.
- A very common source of incorrect results in C programs is the input of a mixture of character and numeric data.
- Errors can be avoided if the programmer always keeps in mind scanf 's different treatment of the %c placeholder on the one hand and of the %d and %lf placeholders on the other.
- Figure 2.17 p.97.
- Figure 2.18 p.99

Refernces

Problem Solving and Program Design in C, 7th Ed., by Jeri R. Hanly and Elliot B. Koffman