Introduction to Computers and Programming

Dr Bhanu

Computers : Software and Hardware

- A computer is a device that can perform computations and make logical decisions billions of times faster than human beings can.
- Many of today's personal computers can perform several billion additions per second.
- The instructions we write to command computers to perform actions and make decisions is called Software.
- Today's fastest supercomputers can perform thousands of trillions (quadrillions) of instructions per second!
- Computers process data under the control of sets of instructions called computer programs

Computers : Software and Hardware

- A computer consists of various devices referred to as hardware (e.g., the keyboard, screen, mouse, hard disk, memory, DVDs and processing units).
- The programs that run on a computer are referred to as software.

Computer Architecture

- Every computer may be visualized as divided into logical units or sections:
  - Input unit. It obtains information (data and computer programs) from input devices for processing by other units. Information can be entered through keyboard and in many ways, including by speaking to your computer, scanning images and barcodes, reading from secondary storage devices (like hard drives, CD drives, DVD drives and USB drives—also called "thumb drives") and having your computer receive information from the Internet (such as when you download videos from YouTube™, e-books from Amazon and the like).

Computer Architecture

- Output unit. It takes information that the computer has processed and places it on various output devices to make it available for use outside the computer.
- Memory unit. It stores information that has been entered through the input unit, making it immediately available for processing when needed.
    - RAM ROM Hard Disk
- Arithmetic and logic unit (ALU). It performs calculations, such as addition, subtraction, multiplication and division. It also allows the computer to take decisions.
    - Ex: Compare two items from the memory unit and determine whether they are equal.

Computer Architecture

- Central processing unit (CPU). This is the heart of Computer.
- It coordinates and supervises the operation of the other sections.
  - tells the input unit when to read information into the memory unit
  - tells the ALU when information from the memory unit should be used in calculations
  - tells the output unit when to send information from the memory unit to certain output devices.

Computer Architecture

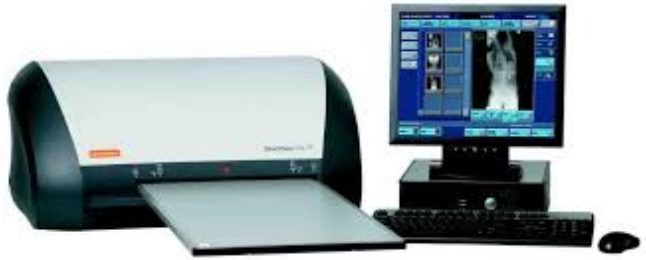
Storage Unit

Secondary Storage

Primary Storage

Data → Input Unit

Output Unit → Information



Control Unit (CU)

Arithmatic And Logical Unit (ALU)

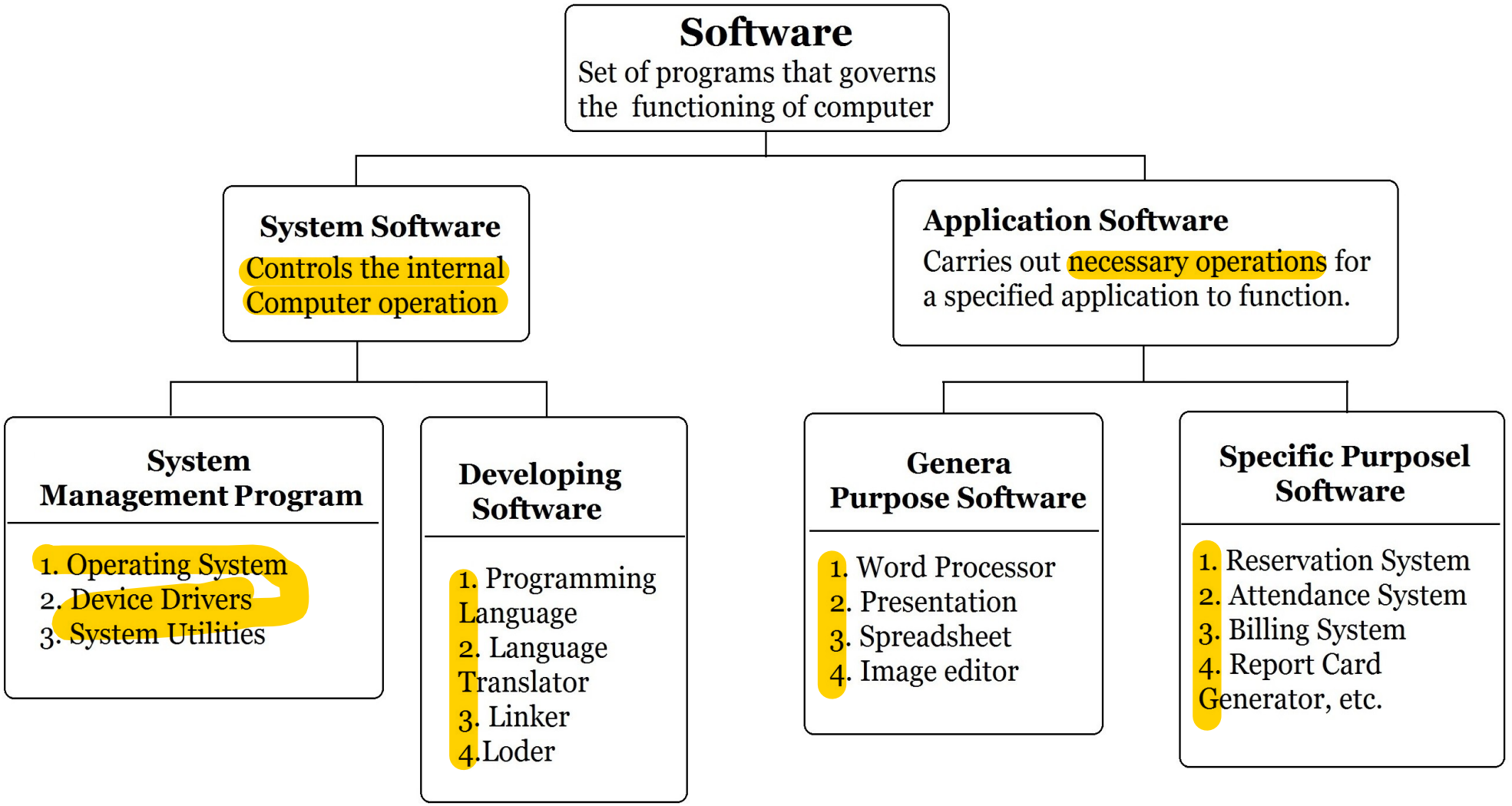
Central Processing Unit (CPU)

→ Data Flow

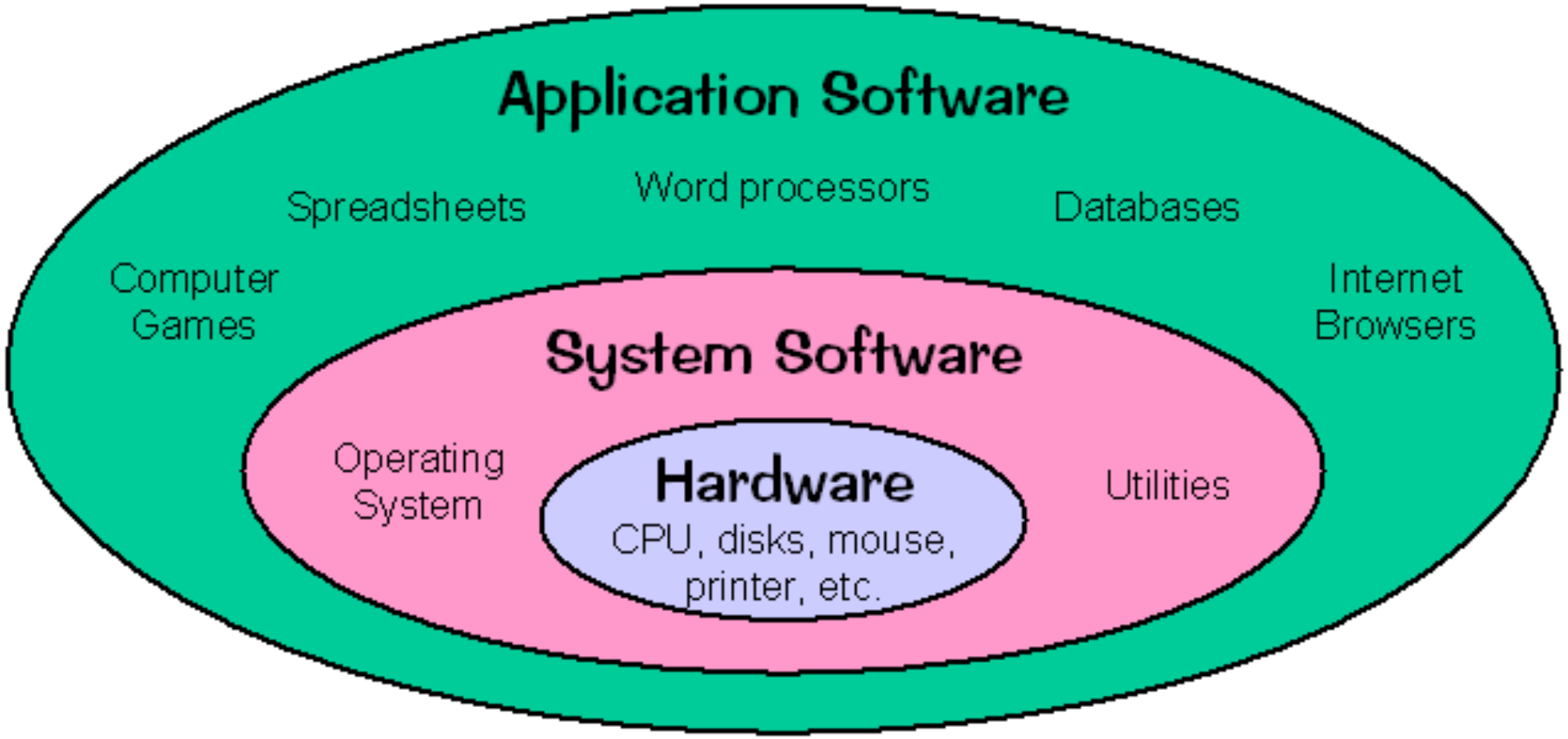
····· Control Flow

Primary and Secondary storage

Primary	Secondary
Very Expensive	Relatively much cheaper
Very fast	Relatively slow
Limited	Unlimited



Hardware, Software and User



Where is the user?

Machine Language, Assembly Language and High-Level Language

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.
- Computer languages may be divided into three general types:
  - Machine languages
  - Assembly languages
  - High-level languages
- Any computer can directly understand only its own machine language.
- Machine language is the "natural language" of a computer and as such is defined by its hardware design.

Machine Language

- Machine language is often referred to as object code.
- Machine language consists of strings of numbers (1s and 0s) that instruct computers to perform their most elementary operations one at a time.
    - 10000001, 00110101
- Machine languages are machine dependent (i.e., a particular machine language can be used on only one type of computer). They are processor specific.
- Translator programs called assemblers convert assembly-language programs to machine language.

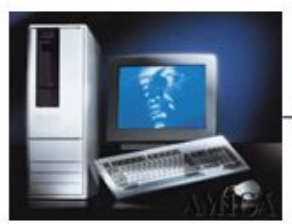
Assembly Language

- Written using Mnemonics
  - Words resemble their meaning
  - Add, Sub, Load, Store, Mul, Div….
- Easy for humans to understand
- Computers can’t understand AL
  - Assembler translates
- Requires many instructions to accomplish even the simplest tasks.
- To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks.

High-level languages

- High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations.
- Translator programs called compilers convert high-level language programs into machine language.
- C, C++, Microsoft's .NET languages (e.g., Visual Basic, Visual C++ and Visual C#) and Java are among the most widely used high-level programming languages.

High Vs. Low Level Languages

Human languages	E.g., English, French, Spanish, Chinese, German, Arabic etc.
High level programming language	E.g., Python, Java, C++ for (i = 1; i <= 10; i++)
Low level programming language	Assembly MOV #10, R0
Machine language	Binary 10100000 1010 00
	Computer hardware

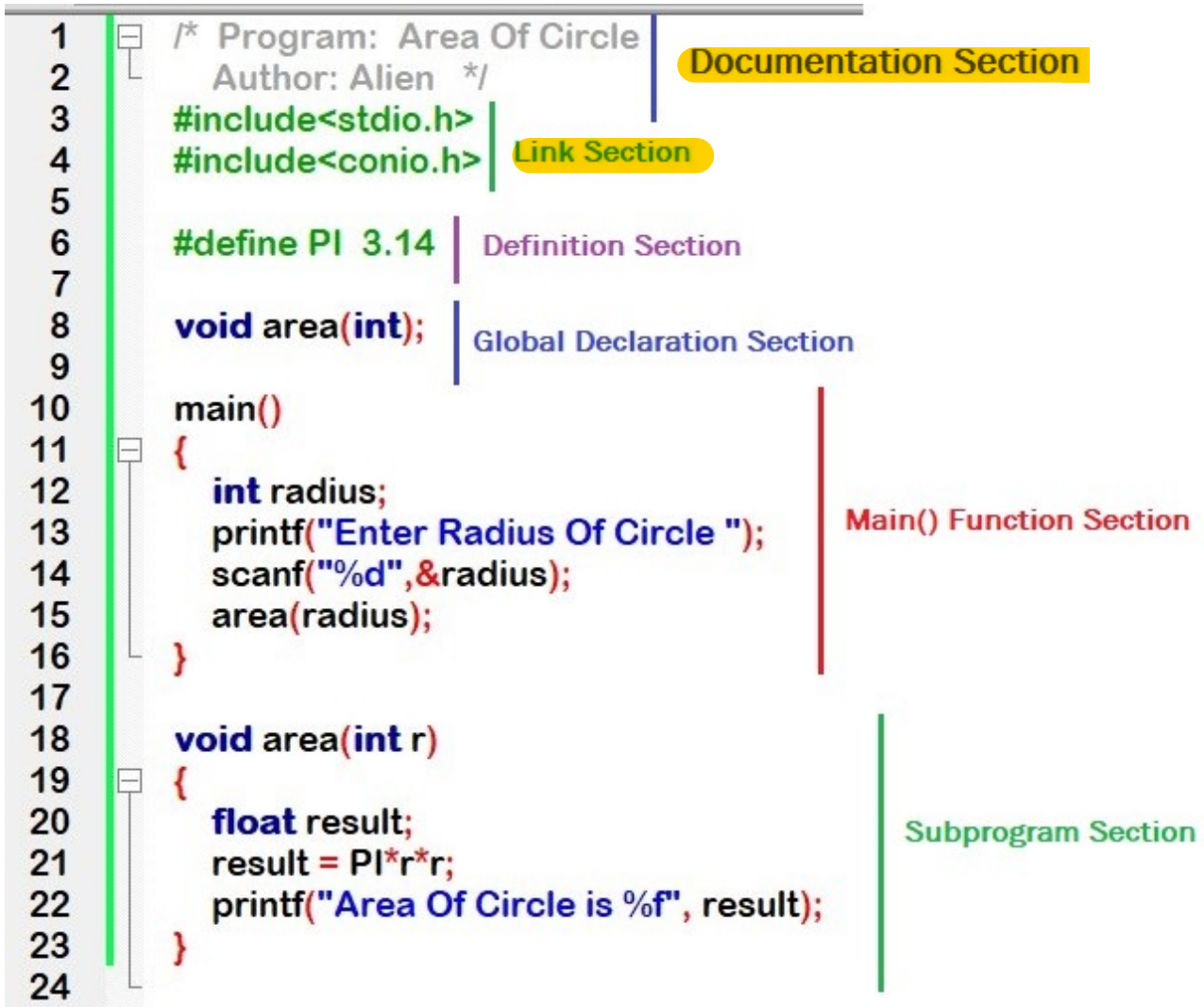
High level

Low level

C Programming Language

- The C language was evolved from B by Dennis Ritchie at Bell Laboratories.
- C initially became widely known as the development language of the UNIX operating system.
- Today, virtually all new major operating systems are written in C and/or C++.
- C is available for most computers.
- C is mostly hardware independent (Portable).

Structure of C Program



A Simple C Program

```
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 */
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

←Documentation

← Preprocessor directive

← Main function

← Body {……..}

Comments

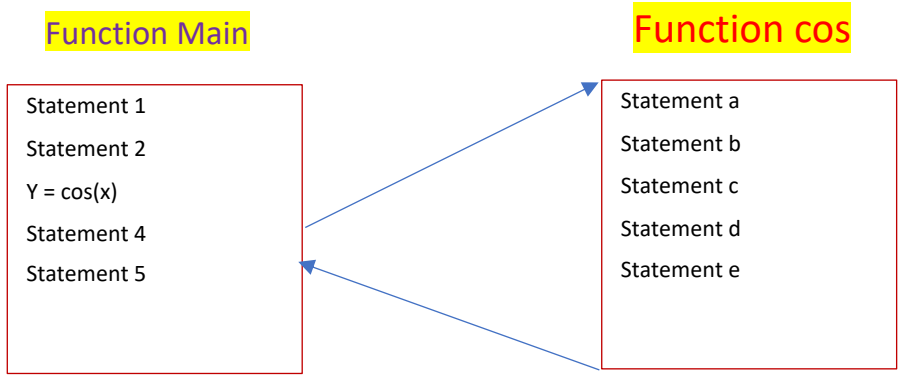
- Block (Multiple line) comments begin with /* and end with */.
- Single-line comments in which everything from // to the end of the line is a comment.
- Insert comments to document programs and improve program readability.
- Comments do not cause the computer to perform any action when the program is run.
- Comments are ignored by the C compiler and do not cause any machine-language object code to be generated.
- Comments also help other people read and understand your program.

Preprocessor Commands

- **`#include <stdio.h>`**
  - is a preprocessor directive to the C compiler.
- It tells the compiler to include the contents of the standard input/output header (<stdio.h>) file in the program
- All preprocessor commands start with a pound sign (#) and appear at the beginning of the program
- They can appear anywhere in the program. Traditionally included at the beginning.

Main Function

- Every C program has a primary function that must be named **main**.
- **main**() **function** is the entry point of any **C** program.
- It is the point at which execution of program starts. When a **C** program is executed, the control goes directly to the **main**() **function**.
- A **return** statement ends the execution of a function and returns control to the calling function. Execution resumes in the calling function at the point immediately following the call.



Identifier

- "Identifiers" are the names we give for variables, types, functions, and labels in our program.
- Identifier names must differ in spelling and case from any keywords.
- Keywords are reserved for special use.
- We create an identifier by specifying it in the declaration of a variable, type, or function.

Rules for naming identifiers

- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- The first letter of an identifier should be either a letter or an underscore.
- You cannot use keywords like int, while etc. as identifiers.
- There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

C Keywords

- Keywords are predefined, reserved words used in programming that have special meanings to the compiler.
- Keywords are part of the syntax and they cannot be used as an identifier.
- 32 keywords in C

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

Identifiers

Valid identifiers: total TOTAL sum Sum average aveRage

_x y_ mark_1 x1

Invalid identifiers **:** 1x char marks% sum-salary total marks

Note: Underscore character is usually used as a link between two words in long identifiers.

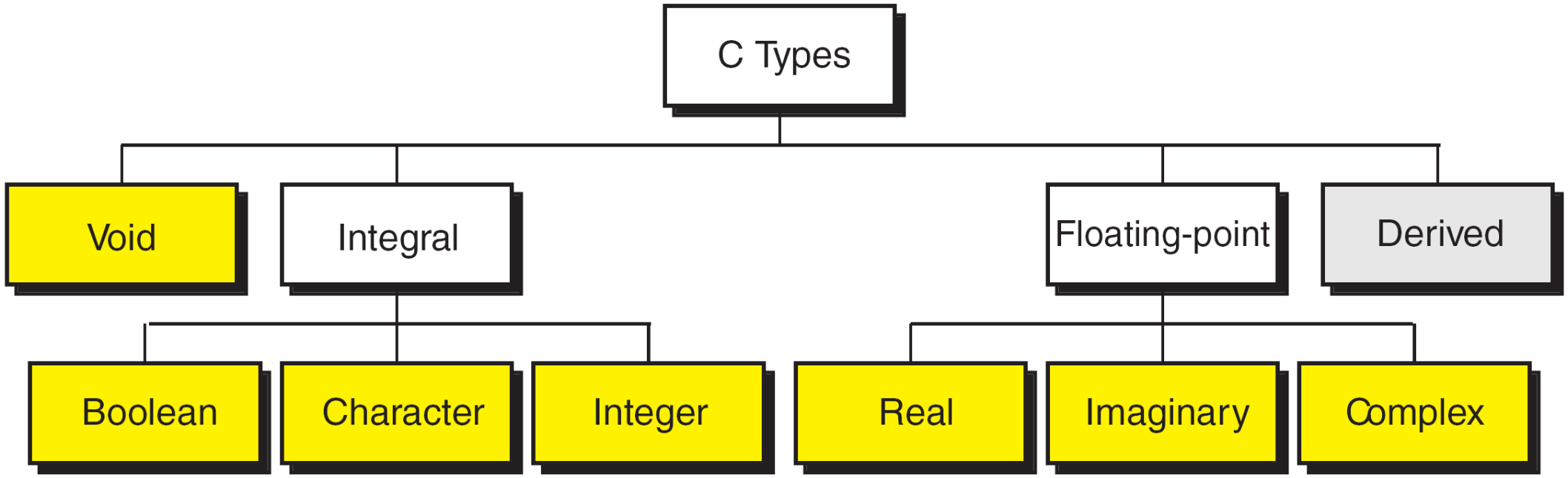
Ex : totalmarks → total_marks → TotalMarks

C is a case-sensitive language

Keywords Vs Identifiers

Keyword	Identifier
Predefined word	User defined word
Must be written in lowercase only	Can written in lowercase and uppercase
Has fixed meaning	User's choice (must be meaningful)
Compiler knows it's meaning	Compiler doesn't know…must be declared
Combination of alphabetic characters	Combination of alphanumeric characters and underscore
Used only for intended purpose only	Used as per user's requirement

C Types

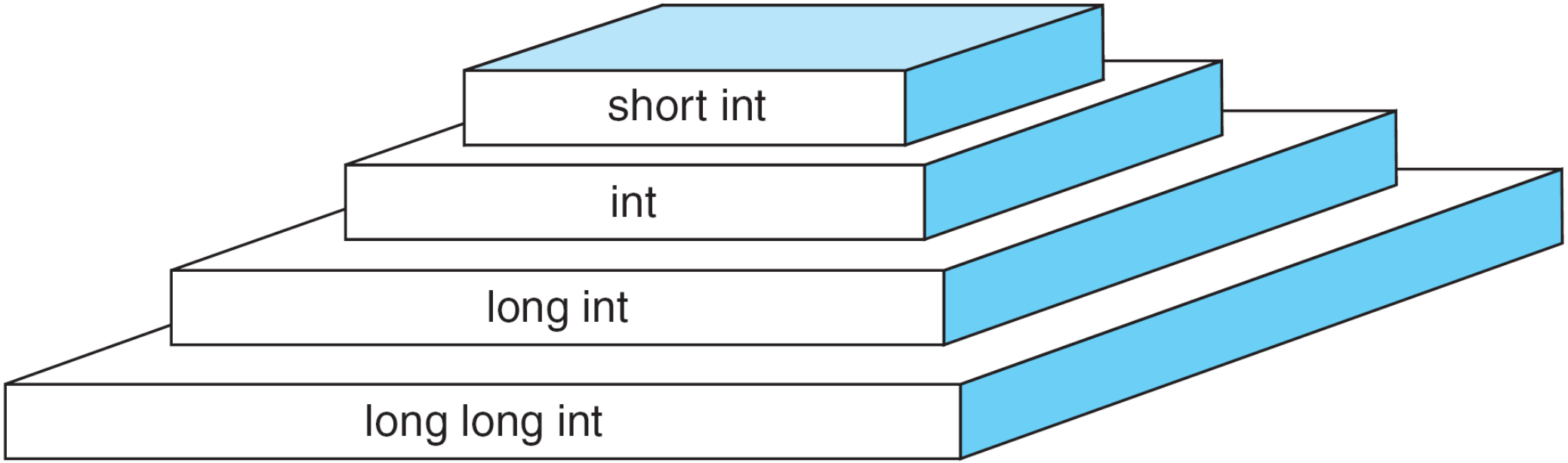


A **type** defines a set of values and a set of operations that can be applied on those values.

C Types

- Void has no values and no operations. Useful in the operation of functions
- Integral types can’t have fraction….whole numbers only
- Boolean type (bool) is stored as 0 (false) or 1 (true)
- Character (char) type represents all the keyboard characters..ASCII
- Integer type is a number without fraction, signed or unsigned

C Integer Types



Note

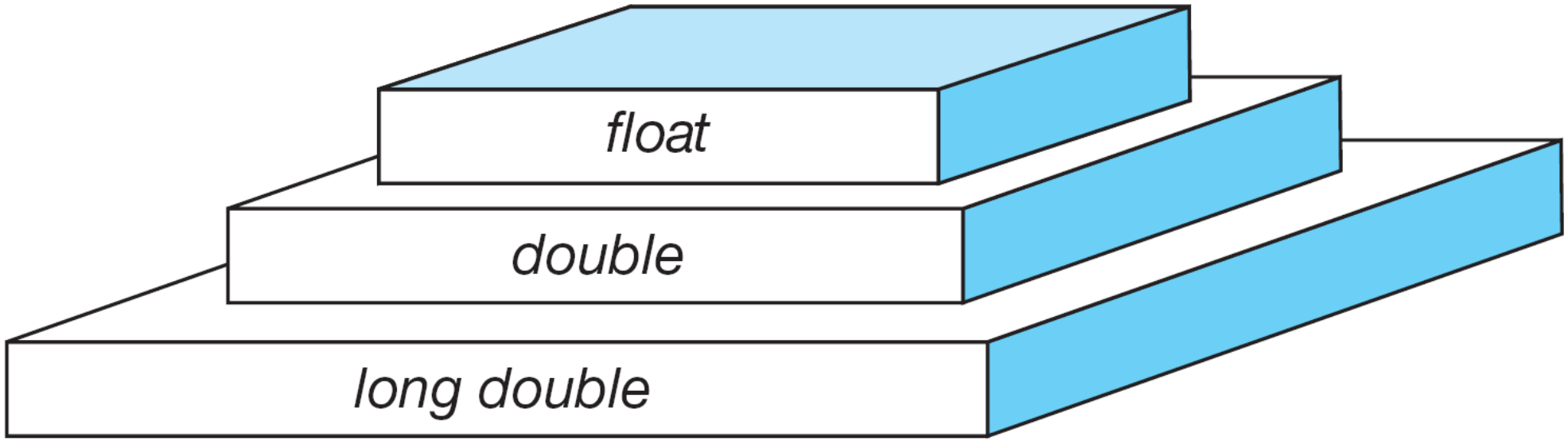
sizeof (short int) ≤ sizeof (int) ≤ sizeof (long int) ≤ sizeof (long long int)

Integer Sizes and Values

- 0 and 1 are called bits
- Byte is group of 8 bits. (Ex. 01000111)
- Byte size 2 means ….16 bits……..2^{16} = 65536

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

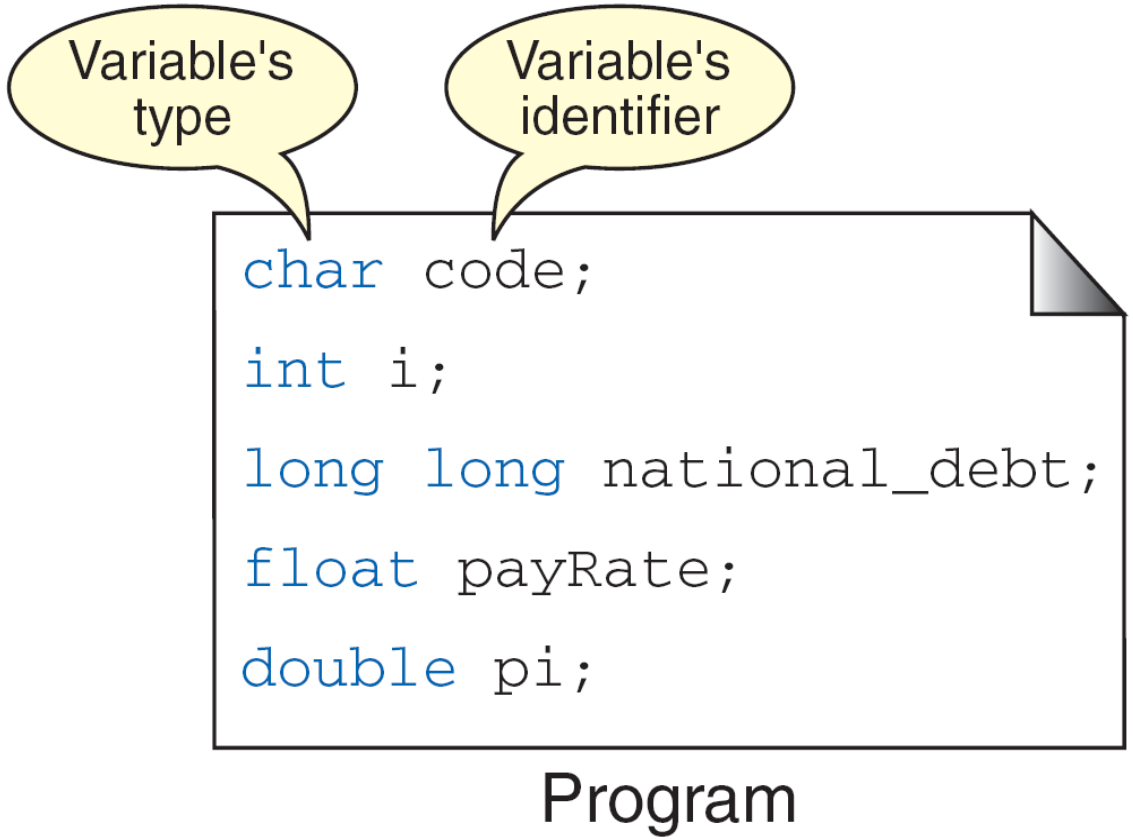
C Floating Point Types



sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Variables

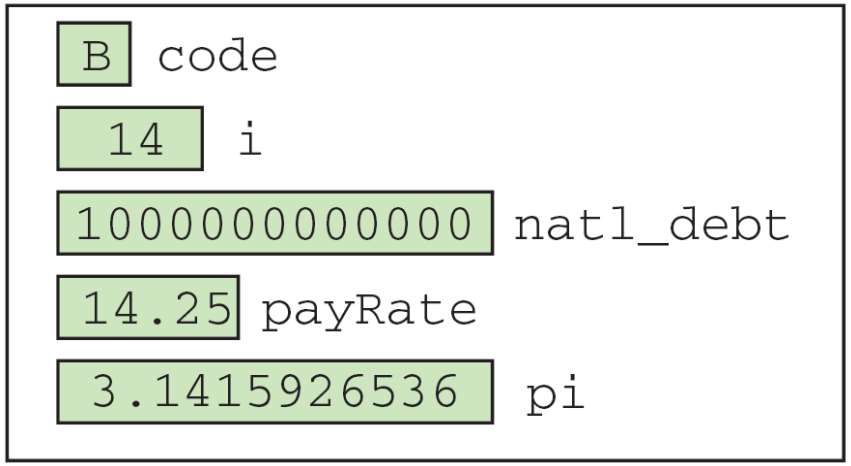
- *Variables are named memory locations that have a type, such as integer or character, which is inherited from their type.*
- *The type determines the values that a variable may contain and the operations that may be used with its values.*



Variable Initialization

```
char code = 'b';
int  i    = 14;
long long natl_debt = 1000000000000;
float     payRate   = 14.25;
double    pi        = 3.1415926536;
```

Program



Memory

Note

When a variable is defined, it is not initialized.
We must initialize any variable requiring
prescribed data when the function starts.

PROGRAM Print Sum of Three Numbers

```
1  /* This program calculates and prints the sum of
2     three numbers input by the user at the keyboard.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    int a;
12    int b;
13    int c;
14    int sum;
15
```

PROGRAM Print Sum of Three Numbers (continued)

```
16  // Statements
17     printf("\nWelcome. This program adds\n");
18     printf("three numbers. Enter three numbers\n");
19     printf("in the form: nnn nnn nnn <return>\n");
20     scanf("%d %d %d", &a, &b, &c);
21
22     // Numbers are now in a, b, and c. Add them.
23     sum = a + b + c;
24
25     printf("The total is: %d\n\n", sum);
26
27     printf("Thank you. Have a good day.\n");
28     return 0;
29  }  //  main
```

PROGRAM Print Sum of Three Numbers (continued)

```
Results:
Welcome. This program adds
three numbers. Enter three numbers
in the form: nnn nnn nnn <return>
11 22 33

The total is: 66

Thank you. Have a good day.
```

Arithmetic in C

- The C arithmetic operators are summarized in Fig. 2.9.
- The asterisk (`*`) indicates multiplication and the percent sign (`%`) denotes the remainder operator
- In algebra, if we want to multiply *a times b, we can simply place these single-letter variable names side by side as in ab.*
- In C, however, if we were to do this, `ab` would be interpreted as a single, two-letter name (or identifier).
- Arithmetic expressions in C must be written in straight-line form.

Arithmetic in C

- The arithmetic operators are all binary operators.
- Integer division yields an integer result.
- For example, the expression `7 / 4` evaluates to `1` and the expression `17 / 5` evaluates to `3`.
- C provides the remainder operator, `%`, which yields the remainder after integer division.
- The remainder operator is an integer operator that can be used only with integer operands.
- The expression `x % y` yields the remainder after `x` is divided by `y`.
- Thus, `7 % 4` yields `3` and `17 % 5` yields `2`.

Arithmetic Operators in C

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

Operator Precedence

- The rules of operator precedence are generally the same as those in algebra:
  - Operators in expressions contained within pairs of parentheses are evaluated first.
  - *Parentheses are said to be at the "highest level of precedence."*
  - *In cases of nested, or embedded, parentheses, such as*

```
( ( a + b ) + c )
```

the operators in the innermost pair of parentheses are applied first.

Operator Precedence

- Multiplication, division and remainder operations are applied first.
- If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right.
- Multiplication, division and remainder are said to be on the same level of precedence.
- Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right.
- Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Fig. 2.10 | Precedence of arithmetic operators.

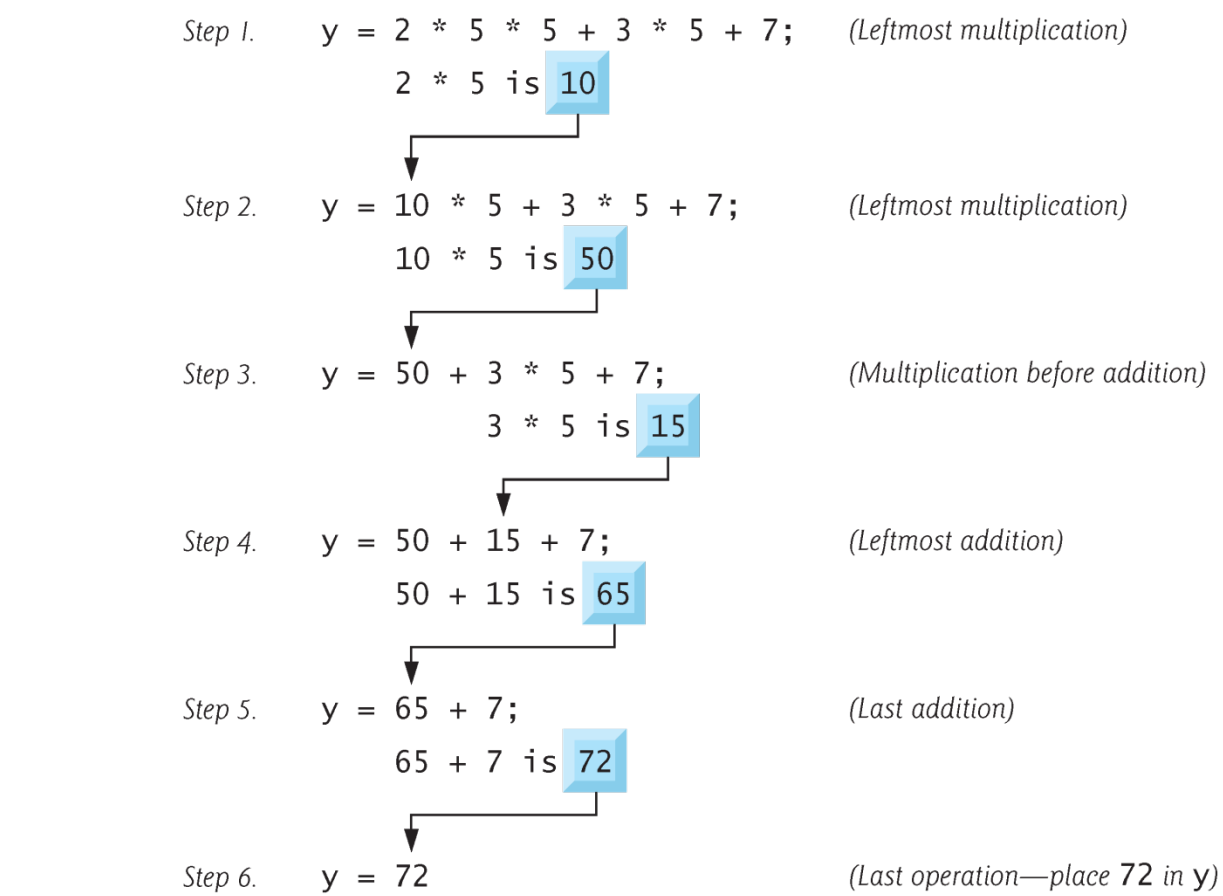


Fig. 2.11 | Order in which a second-degree polynomial is evaluated.