Introduction to C++ Programming

OBJECTIVES

In this lecture you will learn:

1. To write simple computer programs in C++.
2. To write simple input and output statements.
3. To use fundamental types.
4. Basic computer memory concepts.

Introduction

C++ Programming

1. Facilitates disciplined approach to computer program design
2. Programs process information and display results

Examples Demonstrate

3. How to display messages
4. How to obtain information from the user

First Program in C++: Printing a Line of Text

Simple Program

- Prints a line of text
- Illustrates several important features of C++

Outline

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream>
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n";
9
10    return 0; // indicate that program ended successfully
11
12 } // end function main
```

```
Welcome to C++!
```

Single-line comments

Function **main** returns an

directive to

Left brace **{** begins function body

Statements end with a semicolon **;**

exactly once in every C++ program

Corresponding right brace **}** ends fu

Name

Stream insertion operator

namespace **std**

Keyword **return** is one of several means to exit a function; value **0** indicates that the program terminated successfully

2_01.cpp
f 1)

fig02_01.cpp
output (1 of 1)

Good Programming Practice

Every program should begin with a comment that describes the purpose of the program, author, date and time.

Common Programming Error

Forgetting to include the `<iostream>` header file in a program that inputs data from the keyboard or outputs data to the screen causes the compiler to issue an error message, because the compiler cannot recognize references to the stream components (e.g. `cout`).

Common Programming Error

Syntax errors are also called compiler errors, compile-time errors or compilation errors, because the compiler detects them during the compilation phase. **You will be unable to execute your program until you correct all the syntax errors in it.** As you will see, some compilation errors are not syntax errors.

Basics of C Programming

Steps in Learning English Language:

Alphabets → Words → Sentences → Paragraph

Steps in Learning C Language:

Alphabets
Numbers
Special Symbols

→

Constants
Variables
Keywords

→ Instructions → C Program

Character Set in C++

- The character set are set of words, digits, symbols and operators that are valid in C++.
- There are four types of Character Set:-

Character Set in C++		
1.	Letters	Uppercase A-Z Lowercase a-z
2.	Digits	All digits 0-9
3.	Special Characters	All Symbols: , . : ; ? ' " ! | \ / ~ _ $ % # & ^ * - + < > ( ) { } [ ]
4.	White Spaces	Blank space, Horizontal tab, Carriage return, New line, Form feed

Tokens: The smallest individual units of a program are called tokens.

1. Constants
2. Variables
3. Keywords
4. Data Types

A C++ program is written using these tokens, white spaces , and the syntax of the language.

Constants , Identifiers and Keywords

The alphabets , numbers and special symbols when properly combined form constants , identifiers and keywords.

Constant: a constant is a quantity that does not change. This can be stored at a location in memory of computer.

Variable(identifiers) : is considered as a name given to the location in memory where this constant is stored. Naturally the contents of the variable can change. There are fundamental requirement of any language. Each language has its own rules for naming these identifiers.

Following are the rules for naming identifiers:

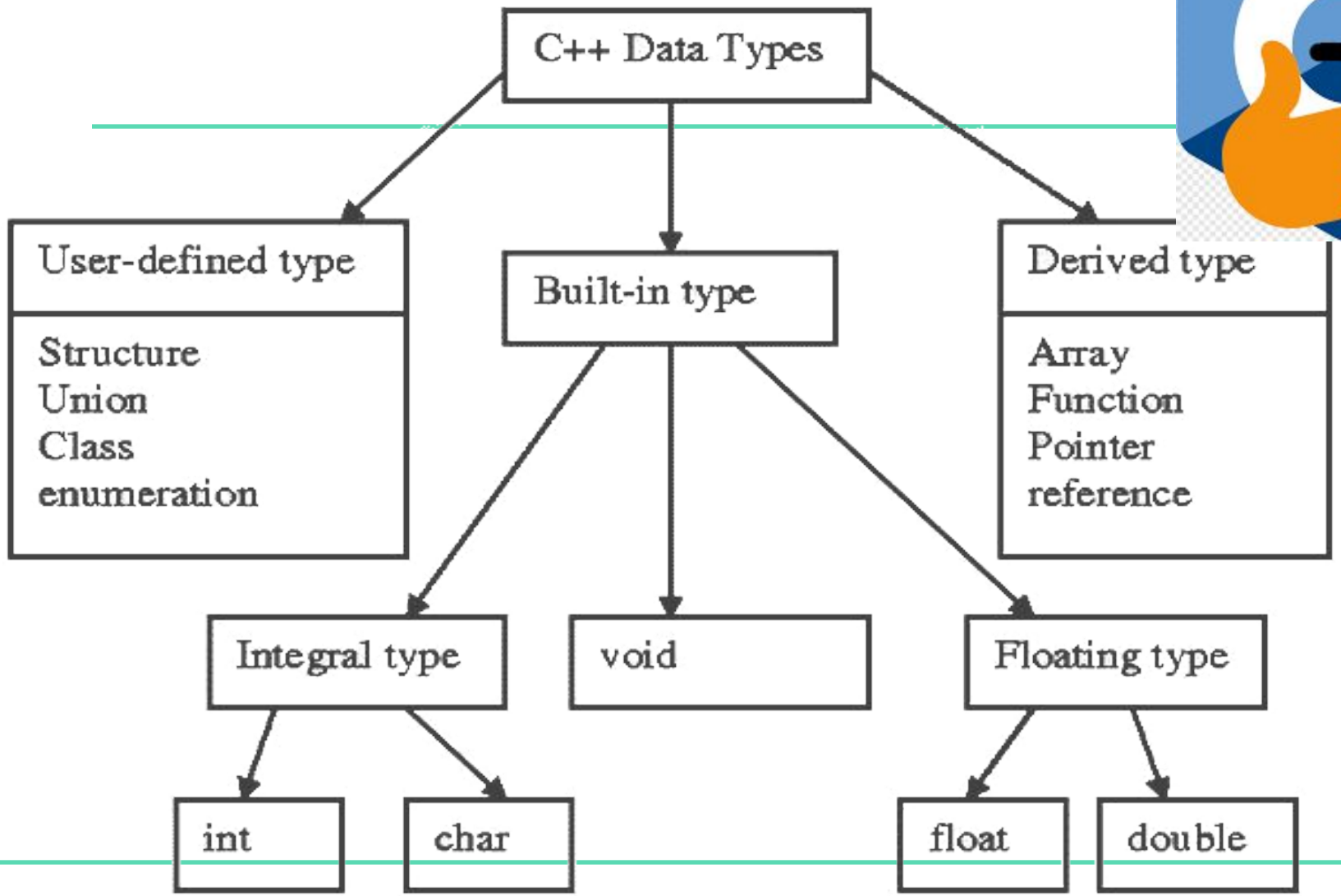
1. Only alphabetic characters digits and underscores are permitted.
2. The name cannot start with a digit.
3. Uppercase and Lowercase letters are distinct
4. A declared keyword cannot be used as a variable name.

For Example:

3X + Y = 20

Keywords

Keywords implement specific C++ language features. They are explicitly reserved identifiers and cannot be used as names for the program variables or other user defined program elements.



Data Type

Data Type : type of data to be stored in a variable

Primitive data type (built-in data type): provided as an integral part of the language

Integer type

Real type

```
int val;
```

Primitive Data Type

Primitive Type

Data type		Memory size (byte)	range
INTEGER	char	1	-128 ~ +127
	short	2 or more	-32768 ~ +32767
	int	4	-2147483648 ~ +2147483647
	long	4	-2147483648 ~ +2147483647
REAL	float	4 or more	$3.4*10^{-37} \sim 3.4*10^{+38}$
	double	8	$1.7*10^{-307} \sim 1.7*10^{+308}$
	long double	8	big

Primitive Data Type

Why do we need to define the type of data?

1. Efficient use of memory space

2. Data loss can happen when store big data into small memory space

Primitive Data Type

sizeof operator

Return memory size of operand in byte

Need () when the operand is data type

Otherwise () is optional

```
#include <iostream>
using namespace std;
int main(void)
{
	int val=10;
	cout << sizeof val << endl;// print memory size
	 cout << sizeof(int) << endl;// print int data type

	return 0;
}
```

Primitive Data Type

Criteria of selection of data type

Real type data

Accuracy

'double' is common

Data type	Accuracy
float	6th below decimal point
double	15th below decimal point
long double	More accurate than double

Primitive Data Type

Example

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(void)
{
	double radius;
	double area;
	cout << "Input radius of circle" << endl;
	cin >> radius;
	area = radius * radius * 3.1415;
		cout <<  area;
		return 0;
}
```

Primitive Data Type

unsigned: the range of data is changed

Positive integer only

Can not be used in real data type

Data type	Byte	Range
char(signed char)	1	-128 ~ +127
unsigned char	1	0 ~ (127 + 128)
short(signed short)	2	-32768 ~ +32767
unsigned short	2	0 ~ (32767 + 32768)
int(signed int)	4	-2147483648 ~ +2147483647
unsigned int	4	0 ~ (2147483647 + 2147483648)
long(signed long)	4	-2147483648 ~ +2147483647
unsigned long	4	0 ~ (2147483647 + 2147483648)

Primitive Data Type

How to express letter (including characters, notations, …) inside computer?

ASCII (American Standard Code for Information Interchange) code was born for expressing letters.

Defined by ANSI (American National Standard Institute)

The standard of letter expression by computer

Ex) letter ‘A’ □ 65, letter ‘B’ □ 66

Primitive Data Type

Range of ASCII code

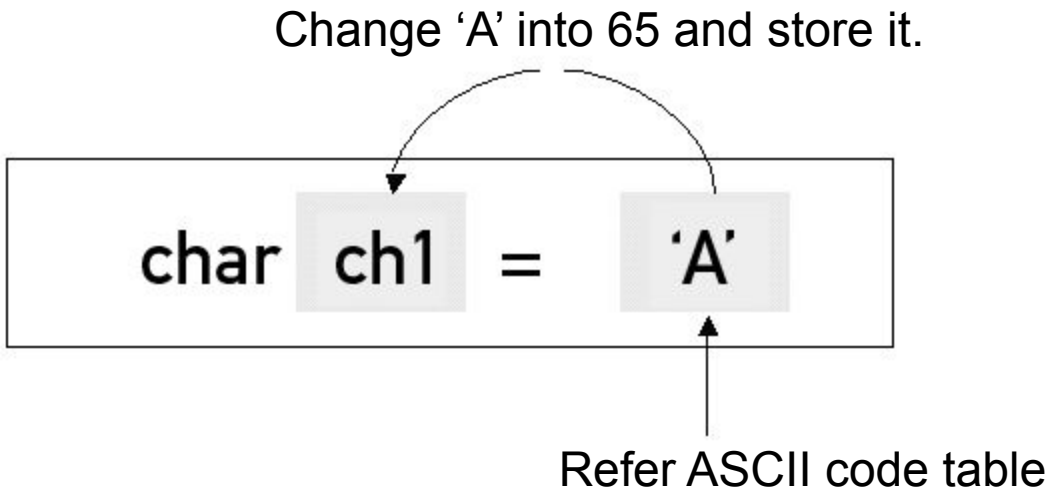
0 ~ 127, □ possible using ‘char’ type

Declare ‘char’

Expression of letters

‘’ (quotation mark)

Declare “store letter A into variable ch1”



Primitive Data Type

Example

```
#include <iostream>
using namespace std;
int main(void)
{
	char ch1='A';
	char ch2=65;

	cout << ch1 <<endl << ch2 << endl;
	cout << (int)ch1 << endl << (int)ch2 <<endl;

	return 0;
}
```

Primitive Data Type

ASCII code

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Symbolic Constant

Make ‘variable’ to ‘constant’

```
#include <iostream>
using namespace std;
int main(void)
{
	const int MAX = 100;
	const double PI = 3.1415;
	return 0;
}
```

Another C++ Program: Adding Integers

- **Variables**
  - **Location in memory where value can be stored**
  - **Common data types (fundamental, primitive or built-in)**
    - `int` **– integer numbers**
    - `char` **– characters**
    - `double` **– floating point numbers**
  - **Declare variables with name and data type before use**
    - `int integer1;`
    - `int integer2;`
    - `int sum;`

Another C++ Program: Adding Integers (Cont.)

- **Variables (Cont.)**
  - **Can declare several variables of same type in one declaration**
    - **Comma-separated list**
    - `int integer1, integer2, sum;`
  - **Variable names**
    - **Valid identifier**
      - **Series of characters (letters, digits, underscores)**
      - **Cannot begin with digit**
      - **Case sensitive (upper and lower case letter)**
    - **Keywords**

```cpp
// Fig. 2.5: fig02_05.cpp
// Addition program that displays the sum of two numbers.
#include <iostream> // allows program to perform input and output

// function main begins program execution
int main()
{
   // variable declarations
   int number1; // first integer to add
   int number2; // second integ
   int sum; // sum of number1 a

   std::cout << "Enter first int
   std::cin >> number1; // read first integer from user into number1

   std::cout << "Enter second integer: "; // prompt user for data
   std::cin >> number2; // read second integer from user into number2

   sum = number1 + number2; // add the numbers; store result in sum

   std::cout << "Sum is " << sum << std::endl; // display sum; end line

   return 0; // indicate that program ended successfully

} // end function main
```

Declare integer variables

Use stream extraction operator with standard input stream to obtain user input

Stream manipulator **`std::endl`** outputs a newline, then "flushes output buffer"

Concatenating, chaining or cascading stream insertion operations

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

`fig02_05.cpp`
`output` (1 of 1)

Good Programming Practice

Place a space after each comma (,) to make programs more readable.

Good Programming Practice

Some programmers prefer to declare each variable on a separate line. This format allows for easy insertion of a descriptive comment next to each declaration.

Portability Tip

C++ allows identifiers of any length, but your C++ implementation may impose some restrictions on the length of identifiers. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice

Choosing meaningful identifiers helps make a program self-documenting—a person can understand the program simply by reading it rather than having to refer to manuals or comments.

Good Programming Practice

Always place a blank line between a declaration and adjacent executable statements. This makes the declarations stand out in the program and contributes to program clarity.

Another C++ Program: Adding Integers (Cont.)

- Input stream object
  - `std::cin` from `<iostream>`
    - Usually connected to keyboard
    - Stream extraction operator `>>`
      - Waits for user to input value, press *Enter* (Return) key
      - Stores value in variable to right of operator
        - Converts value to variable data type
    - Example
      - `std::cin >> number1;`
        - Reads an integer typed at the keyboard
        - Stores the integer in variable `number1`

Another C++ Program: Adding Integers (Cont.)

- Assignment operator =
  - Assigns value on left to variable on right
  - Binary operator (two operands)
  - Example:
    - `sum = variable1 + variable2;`
      - Add the values of `variable1` and `variable2`
      - Store result in `sum`
- Stream manipulator `std::endl`
  - Outputs a newline
  - Flushes the output buffer

Another C++ Program: Adding Integers (Cont.)

- Concatenating stream insertion operations
  - Use multiple stream insertion operators in a single statement
    - Stream insertion operation knows how to output each type of data
  - Also called chaining or cascading
  - Example
    - `std::cout << "Sum is " << number1 + number2 << std::endl;`
      - Outputs `"Sum is "`
      - Then, outputs sum of `number1` and `number2`
      - Then, outputs newline and flushes output buffer

Memory Concept

- Variable names
  - Correspond to actual locations in computer's memory
    - Every variable has name, type, size and value
  - When new value placed into variable, overwrites old value
    - Writing to memory is destructive
  - Reading variables from memory nondestructive
  - Example
    - `sum = number1 + number2;`
      - Value of `sum` is overwritten
      - Values of `number1` and `number2` remain intact

Fig. 2.6 | Memory location showing the name and value of variable number1.

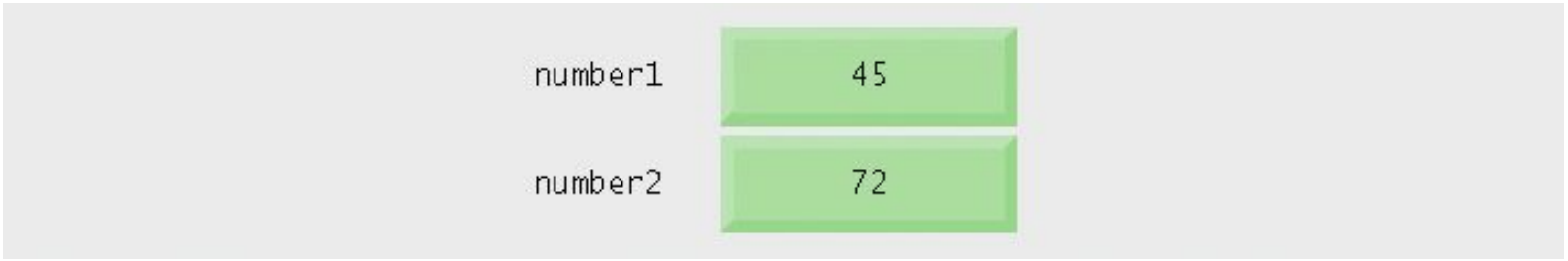


Fig. 2.7 | Memory locations after storing values for number1 and number2.

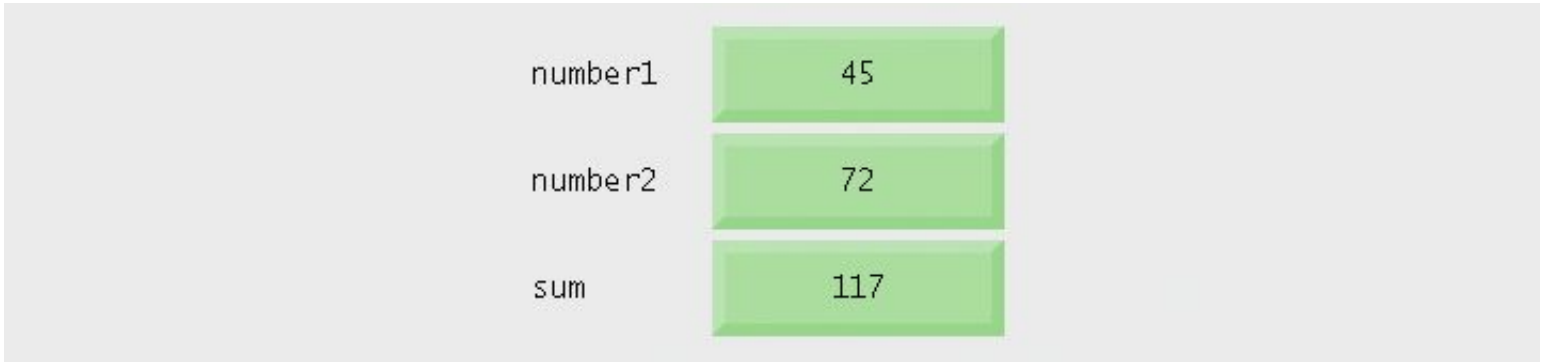


Fig. 2.8 | Memory locations after calculating and storing the sum of number1 and number2.

Arithmetic

- Arithmetic operators
  - `*`
    - Multiplication
  - `/`
    - Division
    - Integer division truncates remainder
      - `7 / 5` evaluates to 1
  - `%`
    - Modulus operator returns remainder
      - `7 % 5` evaluates to 2

Common Programming Error

Attempting to use the modulus operator (%) with non integer operands is a compilation error.

Arithmetic (Cont.)

- Straight-line form
  - Required for arithmetic expressions in C++
  - All constants, variables and operators appear in a straight line
- Grouping subexpressions
  - Parentheses are used in C++ expressions to group subexpressions
    - Same manner as in algebraic expressions
  - Example
    - `a * ( b + c )`
      - Multiple `a` times the quantity `b + c`

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

Fig. 2.9 | Arithmetic operators.

2.6 Arithmetic (Cont.)

- **Rules of operator precedence**
  - **Operators in parentheses evaluated first**
    - **Nested/embedded parentheses**
      - **Operators in innermost pair first**
  - **Multiplication, division, modulus applied next**
    - **Operators applied from left to right**
  - **Addition, subtraction applied last**
    - **Operators applied from left to right**

Common Programming Error 2.4

Some programming languages use operators ** or ^ to represent exponentiation. C++ does not support these exponentiation operators; using them for exponentiation results in errors.

Use pow(A, B) = A^B function in C++

```
Step 1.  y = 2 * 5 * 5 + 3 * 5 + 7;   (Leftmost multiplication)
         2 * 5 is 10

Step 2.  y = 10 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
         10 * 5 is 50

Step 3.  y = 50 + 3 * 5 + 7;          (Multiplication before addition)
         3 * 5 is 15

Step 4.  y = 50 + 15 + 7;             (Leftmost addition)
         50 + 15 is 65

Step 5.  y = 65 + 7;                  (Last addition)
         65 + 7 is 72

Step 6.  y = 72                       (Last operation—place 72 in y)
```

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

Decision Making: Equality and Relational Operators

- **Condition**
  - Expression can be either `true` or `false`
  - Can be formed using equality or relational operators
- **`if` statement**
  - If condition is `true`, body of the `if` statement executes
  - If condition is `false`, body of the `if` statement does not execute

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Fig. 2.12 | Equality and relational operators.

Common Programming Error 2.5

A syntax error will occur if any of the operators ==, !=, >= and <= appears with spaces between its pair of symbols.

Common Programming Error

Reversing the order of the pair of symbols in any of the operators !=, >= and <= (by writing them as =!, => and =<, respectively) is normally a syntax error. In some cases, writing != as =! will not be a syntax error, but almost certainly will be a logic error that has an effect at execution time. (cont…)

Outline

fig02_13.cpp

(1 of 2)

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // allows program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin;  // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program e
11 int main()
12 {
13    int number1; // first i
14    int number2; // second
15
16    cout << "Enter two integers to compare: "; //
17    cin >> number1 >> number2; // read two integ
18
19    if ( number1 == number2 )
20       cout << number1 << " == " << number2 << e
21
22    if ( number1 != number2 )
23       cout << number1 << " != " << number2 << endl;
24
25    if ( number1 < number2 )
26       cout << number1 << " < " << number2 << en
27
28    if ( number1 > number2 )
29       cout << number1 << " > " << number2 << endl;
30
```

using declarations eliminate need for **std::** prefix

Declare variables

Can write **cout** and **cin** without **std::** prefix

if statement compare
of **number1** and **nu**

If condition is **true** (i.e., values are equal), execute this

if statement compares values of **number1** and **nu** test for inequality

If condition is **true** (i.e., values are not equal), execute this statement

Compares two numbers using relational operator **<** and **>**

```
31    if ( number1 <= number2 )
32       cout << number1 << " <= " << number2 << endl
33
34    if ( number1 >= number2 )
35       cout << number1 << " >= " << number2 << endl;
36
37    return 0; // indicate that program ended successfully
38
39 } // end function main
```

Compares two numbers using relational operators **<=** and **>=**

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

Outline

fig02_13.cpp

(2 of 2)

fig02_13.cpp output (1 of 3)

(2 of 3)

(3 of 3)

Common Programming Error

It is a syntax error to split an identifier by inserting white-space characters (e.g., writing `main` as `ma in`).