Introduction to programming-1

Beginning Concepts for Problem Solving in C++:

variables, data types, operators, and formatted IO

Algorithms and flowcharts

Learning Objectives

List and explain the basic kinds of data

- Distinguish between variables and constants
- Declare integer, floating point, and character variables
- List the basic data types used in C++
- Identify and explain commonly used operators in C++
- Use C++ operators correctly according to their placement in the hierarchy chart
- Set up and evaluate expressions and equations using variables, constants, operators, and hierarchy of operations

Character Set

- Set of characters used in the language
- Alphabets
- A to Z and a to z
- Digits 0 to 9
- Special Ch , : . “ ! + - * / ? # $ etc

ASCII Table

Non-Printing Characters					Printing Characters								
Name	Ctrl char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
null	ctrl-@	0	00	NUL	32	20	Space	64	40	@	96	60	`
start of heading	ctrl-A	1	01	SOH	33	21	!	65	41	A	97	61	a
start of text	ctrl-B	2	02	STX	34	22	"	66	42	B	98	62	b
end of text	ctrl-C	3	03	ETX	35	23	#	67	43	C	99	63	c
end of xmit	ctrl-D	4	04	EOT	36	24	$	68	44	D	100	64	d
enquiry	ctrl-E	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	ctrl-F	6	06	ACK	38	26	&	70	46	F	102	66	f
bell	ctrl-G	7	07	BEL	39	27	'	71	47	G	103	67	g
backspace	ctrl-H	8	08	BS	40	28	(	72	48	H	104	68	h
horizontal tab	ctrl-I	9	09	HT	41	29	)	73	49	I	105	69	i
line feed	ctrl-J	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
vertical tab	ctrl-K	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
form feed	ctrl-L	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
carriage feed	ctrl-M	13	0D	CR	45	2D	-	77	4D	M	109	6D	m
shift out	ctrl-N	14	0E	SO	46	2E	.	78	4E	N	110	6E	n
shift in	ctrl-O	15	0F	SI	47	2F	/	79	4F	O	111	6F	o
data line escape	ctrl-P	16	10	DLE	48	30	0	80	50	P	112	70	p
device control 1	ctrl-Q	17	11	DC1	49	31	1	81	51	Q	113	71	q
device control 2	ctrl-R	18	12	DC2	50	32	2	82	52	R	114	72	r
device control 3	ctrl-S	19	13	DC3	51	33	3	83	53	S	115	73	s
device control 4	ctrl-T	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	ctrl-U	21	15	NAK	53	35	5	85	55	U	117	75	u
synchronous idel	ctrl-V	22	16	SYN	54	36	6	86	56	V	118	76	v
end of xmit block	ctrl-W	23	17	ETB	55	37	7	87	57	W	119	77	w
cancel	ctrl-X	24	18	CAN	56	38	8	88	58	X	120	78	x
end of medium	ctrl-Y	25	19	EM	57	39	9	89	59	Y	121	79	y
substitute	ctrl-Z	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
escape	ctrl-[	27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
file separator	ctrl-\	28	1C	FS	60	3C	<	92	5C	\	124	7C	|
group separator	ctrl-]	29	1D	GS	61	3D	=	93	5D	]	125	7D	}
record separator	ctrl-^	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
unit separator	ctrl-_	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

"http://www.physics.udel.edu/~watson/scen103/ascii.html"
Last updated Feb. 4, 1998

Back

Escape Sequences (partial list)

Escape Sequence	Represents
\a	Bell (alert)
\b	Backspace
\f	Formfeed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\“	Double quotation mark
****	Backslash
\?	Literal question mark

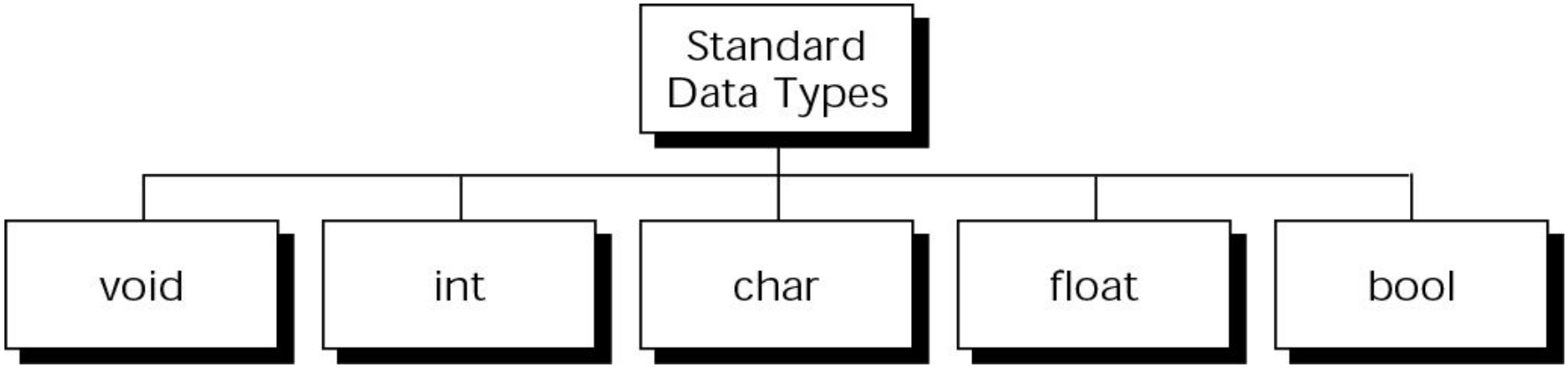
Types of Data (simplified view) ?

- The basic kinds of data that we will mostly use:
  - **Numeric**
    - Integers:
    - Real (floating point) numbers:
  - **Character** (are enclosed by single quotes in C)
    - All letters, numbers, and special symbols
      - Ex. ‘A’, ‘+’, ‘5’
  - **String** (are enclosed by double quotes in C)
    - Are combinations of *more than one* character
      - Ex. “programming”, “ME 30”
  - **Logical** (also called ‘**Boolean**’ named after George Boole an English mathematician from the early 1800’s)
    - True or False (1 or 0)

Write the Type of Data for each value
I-Integer, F-floating point, S-String, C-character

	Data	Type
1.	17	
2.	“George”	
3.	2.35	
4.	0.0023	
5.	-25	
6.	‘m’	
7.	4.32E-6	
8.	“185.3”	
9.	0	
10.	1	

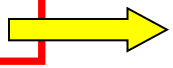
Figure 2-6 Standard data types



Constants and Variables

- Constant
  - A data element that never changes in your program
    - 5, 62.37, 4.219E-6, “record_string”, ‘$’
    - i = j + 7; /* which one is the constant? */
    - first_letter = ‘a’; /* which one is the constant? */
- Variable
  - A data element that can take on different values
    - Its name represents a location (address) in memory ?
      - i = j + 7; /* which are variables? */
      - second_letter = ‘b’;/* which is the variable? */
    - Values are ‘assigned’ using a single equal sign (=)
      - Read the statement: i = j + 7;

NOTE!! Variables in C++ must be ‘*declared*’ before they can be used!



Declaring Variables and Data Types

- Variables must be ***declared*** before they can be used
  - Gives info to the compiler about:
    - How many bytes of memory to set aside ?
    - How to interpret the bytes (data)
  - Example:

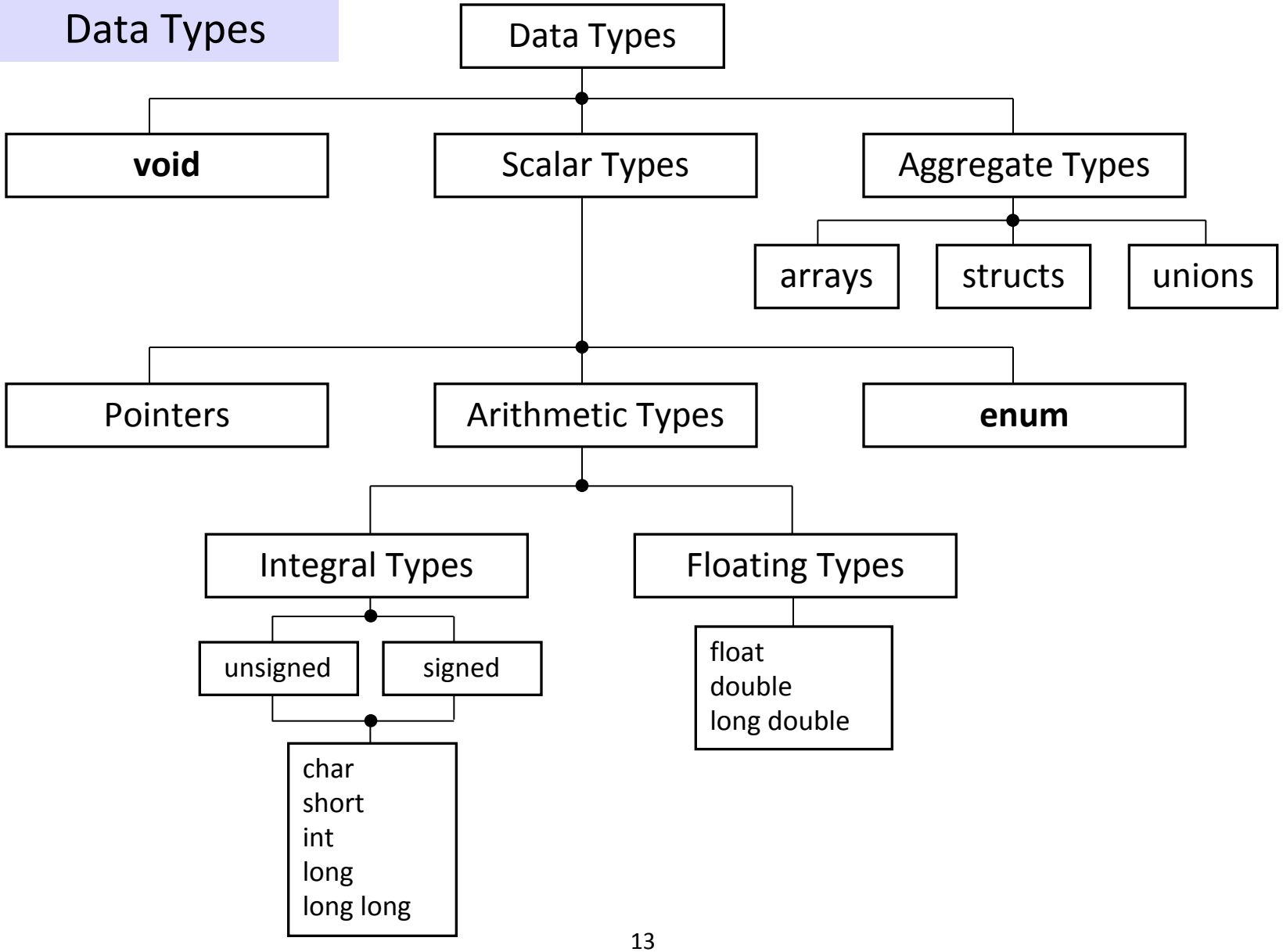
```
int my_var; /* an integer variable my_var */
float sensor1; /* a float variable sensor1 */
char Letter01_a; /* a char variable Letter01_a */
```

    - Two parts are needed in the declaration:
      - *Data type designator* (e.g., **int**), (note, a 'reserved word')
      - *Identifier* (i.e., a name: **my_var**), must be a valid identifier

Data Types and Memory

- Recall that declaring a variable
  - sets aside memory and
  - indicates the type of data that will be stored
- Considerations in declarations
  - What *kind* of data will you be working with?
    - Just whole numbers? Fractions? Characters?
  - Is *speed* important?
    - Integer operations are fastest
  - How much memory is available?
    - Important for embedded systems and microcontrollers
  - What is the *range* of the data?
    - From 0 to n, or from –n to +n, etc.
  - What kind of *precision* do you need?
    - Precision number of decimal places

Hierarchy of Data Types



Memory Allocation for Arithmetic Data Types

		Size (bytes)		Range of Values	
Name (alternate)	**Description**	AVR GCC	MSVC++	AVR GCC	MSVC++
char	Character or small integer	1	1	Signed: -128 to 127 ($-2^{(8-1)}$ to $2^{(8-1)}-1$) Unsigned: 0 to 255 (0 to 2^8-1)	Signed: -128 to 127 Unsigned: 0 to 255
short int (short)	Short integer	2	2	Signed: -32768 to 32767 Unsigned: 0 to 65535	Signed: -32,768 to 32,767 Unsigned: 0 to 65535
int	Integer	2	4	Signed: -32768 to 32767 Unsigned: 0 to 65535	Signed: -2,147,483,648 to 2,147,483,647 Unsigned: 0 to 4,294,967,295
long int (long)	Long integer	4	4	Signed: -2,147,483,648 to 2,147,483,647 Unsigned: 0 to 4,294,967,295	Signed: -2,147,483,648 to 2,147,483,647 Unsigned: 0 to 4,294,967,295
long long int (long long)	Really long integer	8	8	Signed: ≈ -9.2E+18 to ≈ 9.2E+18 Unsigned: 0 to ≈ 1.8E+19	Signed: ≈ -9.2E+18 to ≈ 9.2E+18 Unsigned: 0 to ≈ 1.8E+19
float	'Single-precision' floating point number	4	4	≈ ±1E ± 38 (7 decimal digits of precision)	≈ ±1E ± 38 (7 decimal digits of precision)
double	'Double-precision' floating point number	4	8	≈ ±1E ± 38 (7 decimal digits of precision)	≈ ±1E ±308 (15 decimal digits of precision)
long double	Long double-precision floating point number		8		≈ ±1E ±308 (15 decimal digits of precision)

Constants

Note:

A character constant is enclosed in single quotes.

Note:

Use single quotes for character constants.

Use double quotes for string constants.

Note:

The only bool types constants are true, printed as 1, and false, printed as 0.

Figure 2-10 Some strings

```
""                              // A null string
"h"
"Hello World!\n"
"HOW ARE YOU?"
"Good Morning!"
"'Good' Morning!"               // 'Good' Morning
"\"Good\" Morning!"             // "Good" Morning
```

Figure 2-11 Null characters and null strings

'\0' → Null character

"" → Empty string

Operators

Arithmetic Operations

Name	Symbol	Example	Result
Exponentiation	^	4^2	16
Multiplication	*	16*2	32
Division	/	16/2	8
Addition	+	16+2	18
Subtraction	-	16-2	14

Operator Hierarchy

1. Parenthesis
2. Exponentiation
3. Multiplication/Division
4. Addition/Subtraction

Hierarchy of Operations Example

3 * (6 + 2) / 12 – (7 – 5) ^ 2 * 3 ( ) first

= 3 * 8 / 12 – 2 ^ 2 * 3 ^ next

= 3 * 8 / 12 – 4 * 3 Mult/Div (L to R)

= 24 / 12 – 4 * 3 Mult/Div (L to R)

= 2 – 12 Add/Subtr

= -10

Operators

- Operator: *a symbol (or combination of symbols) used to combine variables and constants to produce a value*
  (Overland, B. (1995) C in plain English, MIS Press, New York.)
  - The variables and constants that get combined are called '*operands*'
  - How the combining is carried out depends on the *precedence* and *associativity* of the operator
  - Example – identify the operator(s), operands, and results on each line:

```
int i, j = 3;
i = j + 7;
```

Operator Precedence and Associativity

- All operators have the properties of *precedence* and *associativity*.
- Precedence has to do with which operations take priority in groupings of operands around adjacent operators
- Associativity has to do with which operations take priority when the operators in an expression have the same precedence
- Use parentheses () to specify a particular grouping order and to make expressions more readable

C Operator Precedence and Associativity

Operator	*Description*	*Associativity*
() [] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (change *type*) Dereference Address Determine size in bytes	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

Note 1: Parentheses are also used to group expressions to force a different order of evaluation; such parenthetical expressions can be nested and are evaluated from inner to outer.

Source: http://www.eece.maine.edu/~eason/ece177/precedence.pdf

C Operator Precedence and Associativity

Operator	*Description*	*Associativity*

`()`
`[]`
`.`
`->`
`++ --`
	Parentheses (function call) (see Note 1)
	Brackets (array subscript)
	Member selection via object name
	Member selection via pointer
	Postfix increment/decrement
		left-to-right

`++ --`
`+ -`
`! ~`
`(type)`
`*`
`&`
`sizeof`
	Prefix increment/decrement
	Unary plus/minus
	Logical negation/bitwise complement
	Cast (change *type*)
	Dereference
	Address
	Determine size in bytes
		right-to-left

`* / %`	Multiplication/division/modulus	left-to-right
`+ -`	Addition/subtraction	left-to-right
`<< >>`	Bitwise shift left, Bitwise shift right	left-to-right
`< <=` `> >=`	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
`== !=`	Relational is equal to/is not equal to	left-to-right
`&`	Bitwise AND	left-to-right
`^`	Bitwise exclusive OR	left-to-right
`|`	Bitwise inclusive OR	left-to-right
`&&`	Logical AND	left-to-right
`||`	Logical OR	left-to-right
`?:`	Ternary conditional	right-to-left

`=`
`+= -=`
`*= /=`
`%= &=`
`^= |=`
`<<= >>=`
	Assignment
	Addition/subtraction assignment
	Multiplication/division assignment
	Modulus/bitwise AND assignment
	Bitwise exclusive/inclusive OR assignment
	Bitwise shift left/right assignment
		right-to-left

`,`	Comma (separate expressions)	left-to-right

Note 1: Parentheses are also used to group expressions to force a different order of evaluation; such parenthetical expressions can be nested and are evaluated from inner to outer.

Source: http://www.eece.maine.edu/~eason/ece177/precedence.pdf

Operator Practice

- Identify the operators, operands, and results on each line

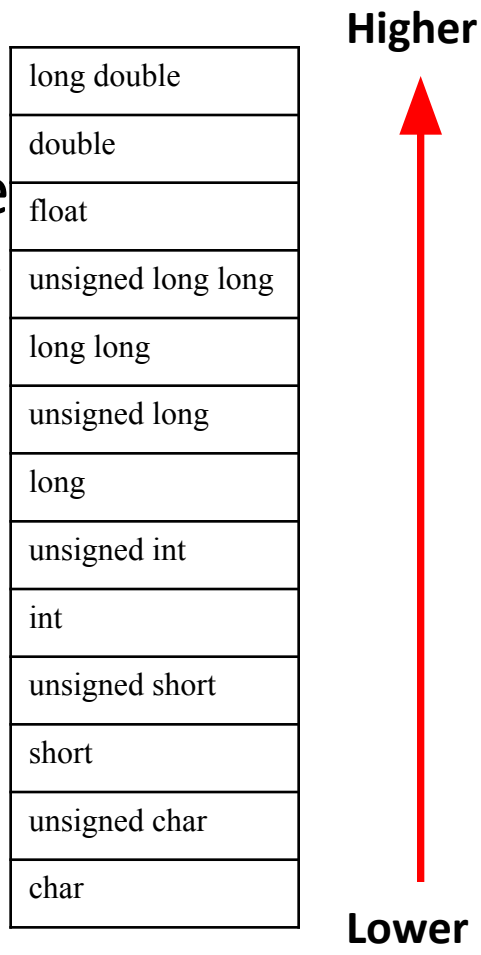
```
int i, count, mult, mult2=1, total_sum;
float total_percentage;
 i = 1;
 mult = total_sum = i;
 count = i + 1;
 count = count + 1;
 total_percentage = total_sum / 100; /* a little tricky! */
 mult1 = count * 17.23;
 count += 1;
 mult2 *= 2;
```

Arithmetic with Mixed Data Types

- Fundamentally, the computer is not able to arithmetically combine data of different types
  - Arithmetic with integers → integer results
    - `int div_int = 8 / 5;  /* what is div_int? */`
  - Arithmetic with floating point data → floating point results
    - `float div_flt = 8.0 / 5.0;  /* what is div_flt? */`
  - Arithmetic with integers and floating point values in the same expressions → ??
    - `int div_int = 8.0 / 5;  /* what is div_int? */`
    - `float div_flt = 8.0 / 5;  /* what is div_flt? */`

Implicit Type Casting

- In binary operations (e.g., +, -, /, etc.) with operands of mixed data types, the *resultant* data type will take the *higher order* data type of the two operands[1]



[1]Cheng, Harry H. (2010). C for Engineers and Scientists: An Interpretive Approach, McGraw-Hill, New York.

Expressions and Statements

- Expression
  - Any combination of operators, numbers, and names that *evaluates to a single value*
    - 5
    - j = 17
    - j = i = 10
    - i + j
- Statement
  - A chunk of code that does something (executes)
    - Terminating an expression with a semicolon (;) turns it into a statement
  - We will cover other statements later

Practice - operator precedence

- On paper, determine output of:

```
int i, j = 3;
i = j + 7;
i = j / 12 + 5;
int k = (j + i * 3);
printf("%d, %d, %d", i, j, k);
```

  - **Output:**

Practice - operator precedence solution

- Program Trace:

```
int i, j = 3;     /* j = 3 */
i = j + 7;        /* i = 3+7 = 10 */
i = j / 12 + 5;   /* i = (3/12)+5 = 9 */
int k = (j + i * 3); /* k=3+27=30 */
printf("%d, %d, %d", i, j, k);
```

  - **Output: 9, 3, 30**

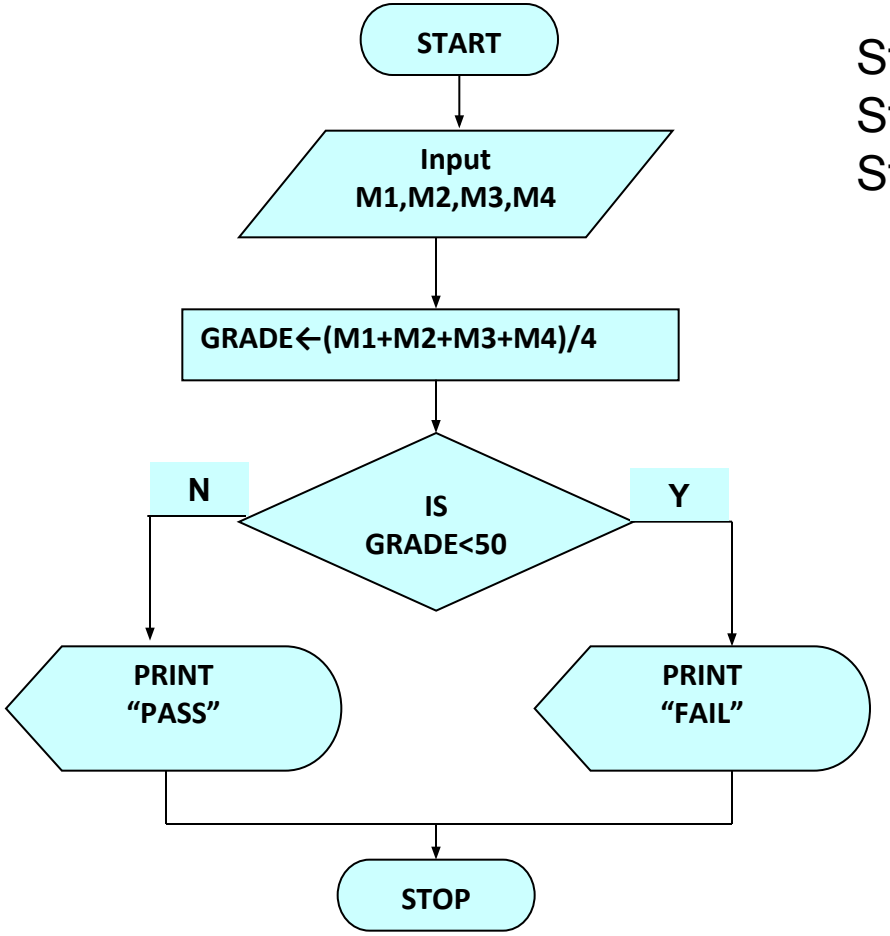
Reserved Words in C++

You may not use these as variable names

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

Back

Example



Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE <50) then
Print “FAIL”
else
Print “PASS”
endif

Example 2

- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

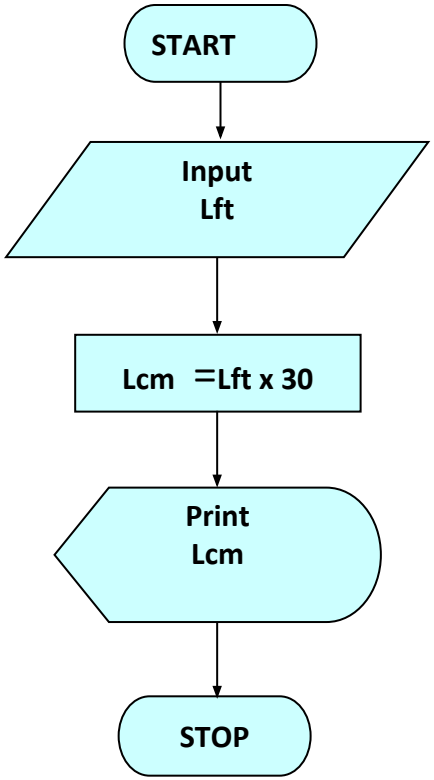
- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

Example 2

Algorithm

- Step 1: Input Lft
- Step 2: Lcm = Lft x 30
- Step 3: Print Lcm

Flowchart



Example 3

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

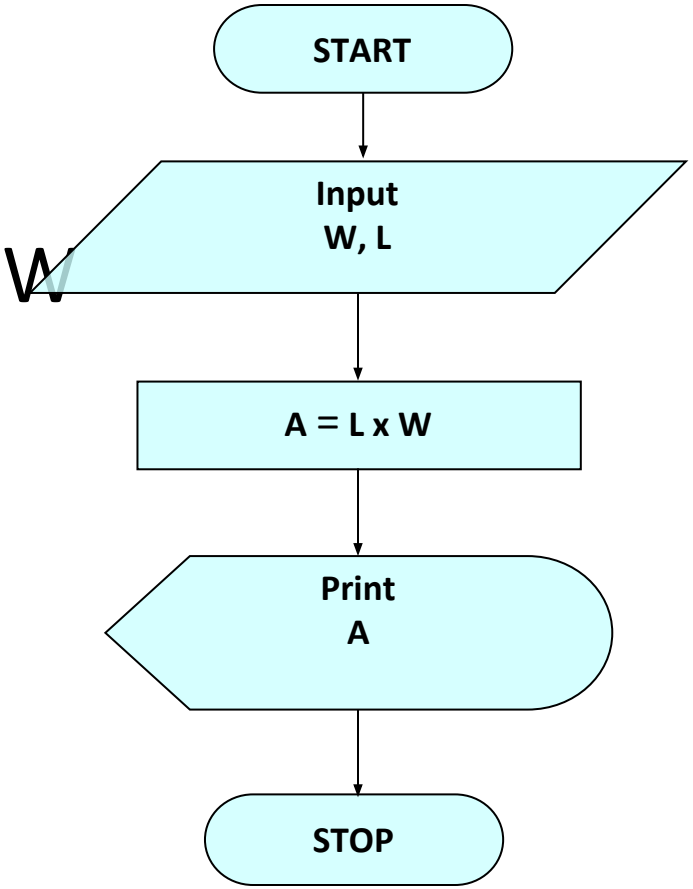
Pseudocode

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Example 3

Algorithm

- Step 1: Input W,Length
- Step 2: Area = Length x W
- Step 3: Print Area



Example 4

- Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation

  $ax^2 + bx + c = 0$

- Hint: **d** = sqrt ($b^2 - 4ac$) and the roots are: ***x*1** = (–*b* + *d*)/2*a* and ***x*2** = (–*b* – *d*)/2*a*

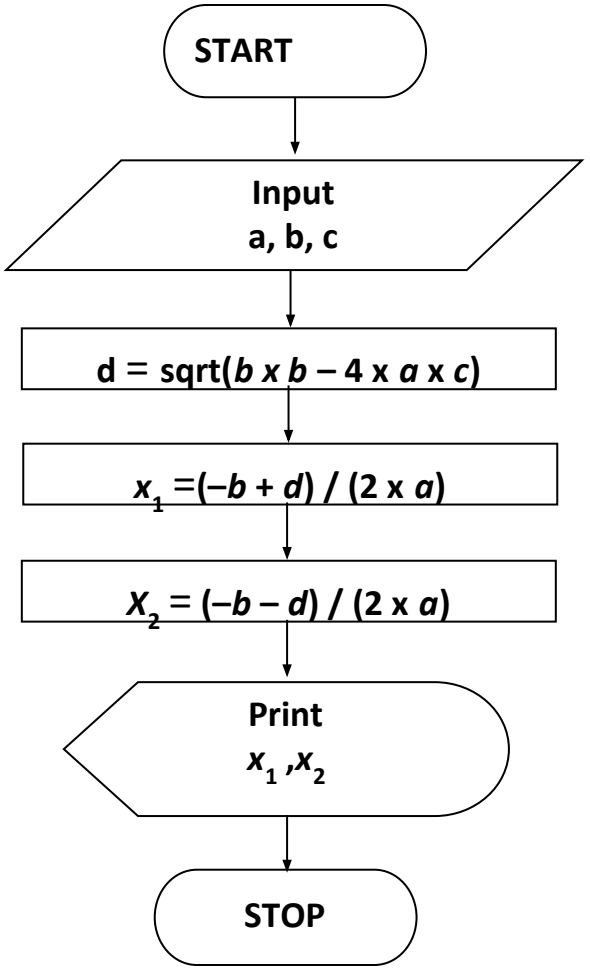
Example 4

Pseudocode:

- *Input the coefficients (a, b, c) of the quadratic equation*
- *Calculate* **d**
- *Calculate* ***x*1**
- *Calculate* **x2**
- *Print* x*1 and* x2

Example 4

- **Algorithm**:
- Step 1: Input a, b, c
- Step 2: *d* = sqrt ($b \times b - 4 \times a \times c$)
- Step 3: *x*1 = (–*b* + *d*) / (2 x *a*)
- Step 4: *x*2 = (–*b* – *d*) / (2 x *a*)
- Step 5: Print *x*1, *x*2



Outline

```
1   // Fig. 1.2: fig01_02.cpp
2   // A first program in C++
3   #include <iostream.h>
4
5   int main()
6   {
7      cout << "Welcome to C++!\n";
8
9      return 0;      // indicate tha
10  }
```

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

Comments

Written between **/*** and ***/** or following a **//**.

Improve program readability and do not cause the computer to perform any action.

preprocessor directive

Message to the C++ preprocessor.

Lines beginning with **#** are preprocessor directives.

#include <iostream.h> tells the

C++ programs contain one or more functions, one of which must be **main**

Parenthesis are used to indicate a function

integer value.

Prints the *string* of characters contained between the

ncluding **cout**, the **<<** *operator*, the **e to C++!\n"** and the *semicolon* *atement*.

All statements must end with a semicolon.

very function

return is a way to exit a function from a function.

return 0, in this case, means that the program terminated normally.