E1123 Computer Programming (a)

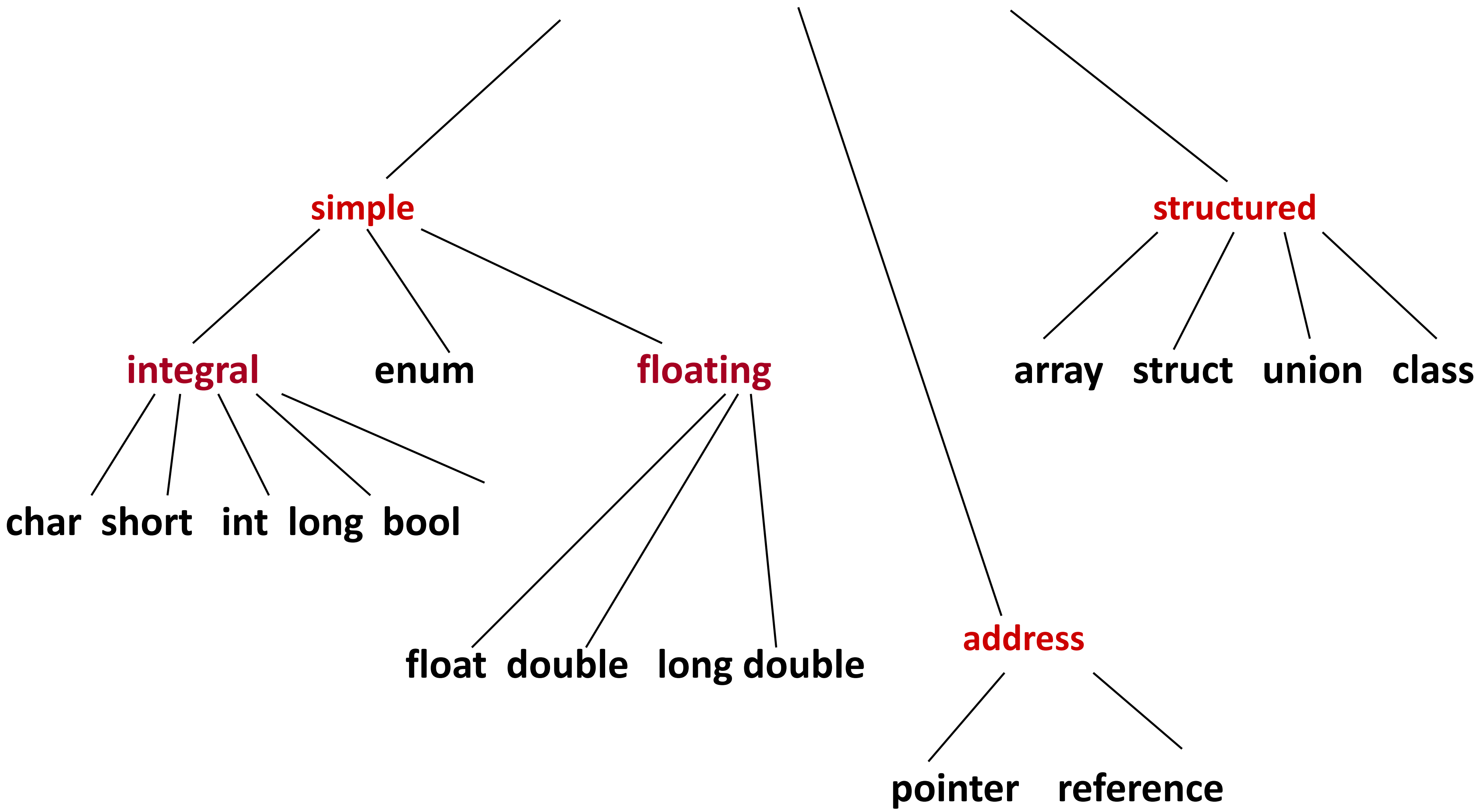
(Fall 2020)

<u>Variables and Data Types</u>

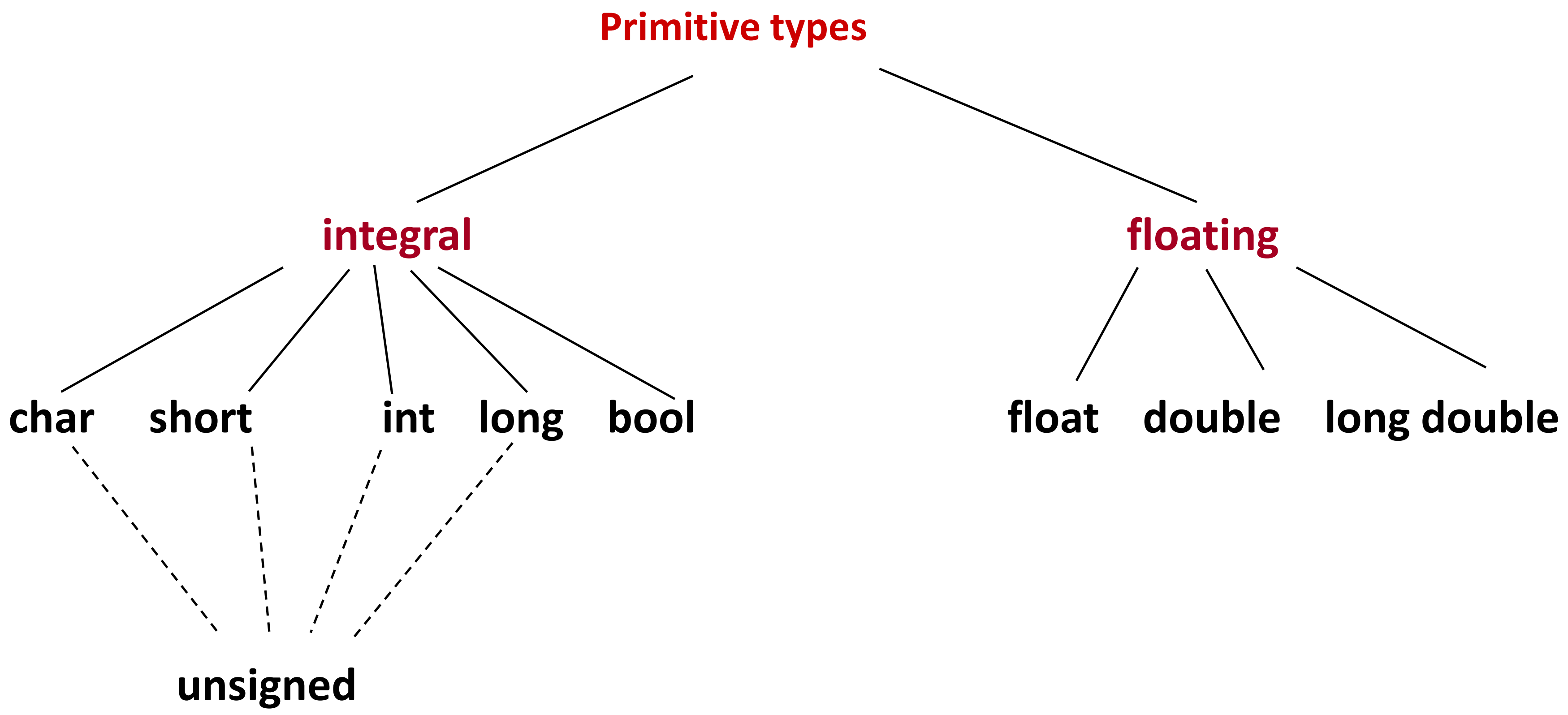
INSTRUCTOR

DR / AYMAN SOLIMAN

C++ Data Types


simple
structured
integral
enum
floating
array
struct
union
class
char
short
int
long
bool
float
double
long double
address
pointer
reference

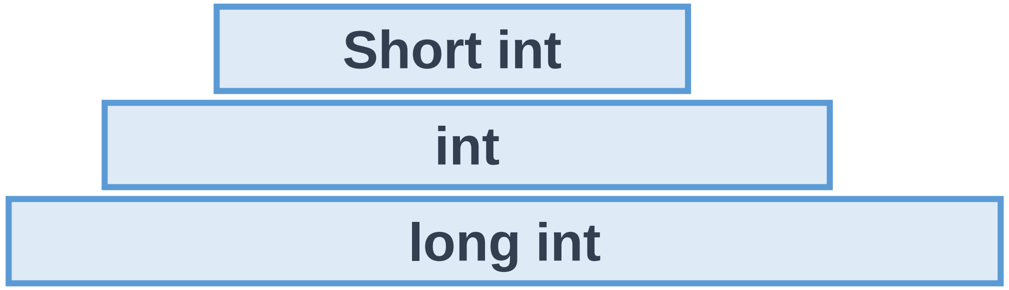
➢ C++ Primitive Data Types



➢ Integer Type

An integer type is a number without a fractional part. It is also known as an integral number. C++ supports three different sizes of the integer data type: short int, int and long int.

sizeof(short int)<= sizeof(int)<= sizeof(long int)



➢ Integer Type

The type also defines the size of the field in which data can be stored. In C++, even though the size is machine dependent, most PCs use the integer sizes shown below.

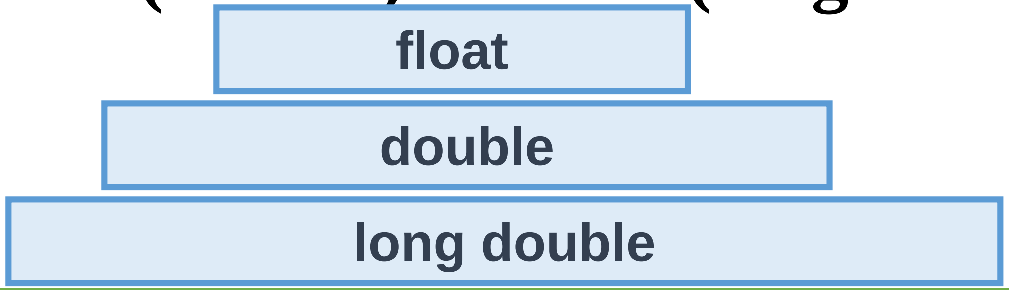
Type	Sign	Byte Size	Number of Bits	Min Value	Max Value
short int	Signed unsigned	2	16	-32768 0	32767 65535
int	Signed unsigned	4	32	-2,147,483,648 0	2,147,483,647 4,294,967,295
long int	Signed unsigned	4	32	-2,147,483,648 0	2,147,483,647 4,294,967,295

➢ Floating Point

A floating-point type is a number with a fractional part, such as 43.32. The C++ language supports three different sizes of floating-point:

- ❑ float
- ❑ double
- ❑ and long double.

sizeof(float)<= sizeof(double)<= sizeof(long double)



➤ Floating Point

Although the physical size of floating-point types is machine dependent, many computers support the sizes shown below.

Type	Byte size	Number of Bits
float	4	32
double	8	64
long double	10	80

➢ bool Data Type

- bool type
  - Two values: true and false
  - Manipulate logical (Boolean) expressions
- true and false are called logical values
- bool, true, and false are reserved words

➢ Character object type

- Char is for specifying character data
- char variable may hold only a single lower-case letter, a single upper-case letter, a single digit, or a single special character like a $, 7, *, etc.
- case sensitive, i.e. a and A are not same.
- ASCII is the dominant encoding scheme
  - Examples

' ' encoded as 32	**'+'** encoded as 43
'A' encoded as 65	**'Z'** encoded as 90
'a' encoded as 97	**'z'** encoded as 122

➢ C++ Data Type String

- A string is a sequence of characters enclosed in double quotes

  - string sample values

    “Hello”   “Year 2000”   “1234”

- The empty string (null string) contains no displayed characters and is written as   “”

➢ C++ Data Type String (cont.)

- **string is not a built-in (standard) type**
  - it is a programmer-defined data type
  - it is provided in the C++ standard library
- Need to include the following two lines:

```
#include <string>
using namespace std;
```

- **string operations** include
  - comparing 2 string values
  - searching a string for a particular character
  - joining one string to another (concatenation)
  - etc...

➢ Assignment Statements

- = is the assignment operator not an equal sign
- Used to assign a value to a variable
- General Form:

  identifier = expression;

  - Assignment statements end with a semi-colon

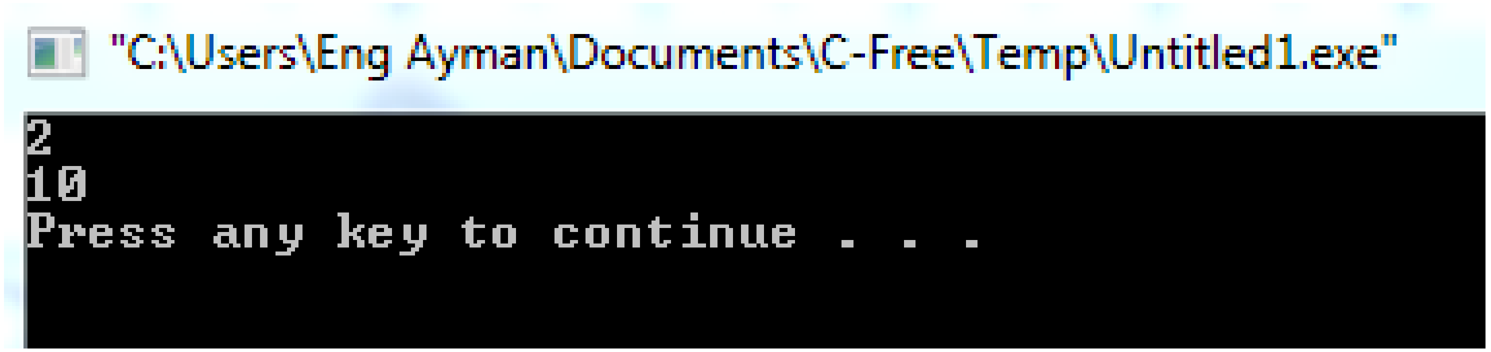
  - The single variable to be changed is always on the left of the assignment operator '=‘

  - On the right of the assignment operator can be
    - Constants -- age = 21;
    - Variables -- my_cost = your_cost;
    - Expressions -- circumference = diameter * 3.14159;

Assignment Conversions

- if double expression is assigned to an integer variable, its fractional part is dropped
- if int expression is assigned to a double variable, the expression is converted to double with zero fractional part
- consider

```
double y = 2.7;
int i = 15;
int j = 10;
i = y;          // i is now 2
cout << i << endl;
y = j;          // y is now 10.0
cout << y << endl;
```


"C:\Users\Eng Ayman\Documents\C-Free\Temp\Untitled1.exe"

```
2
10
Press any key to continue . . .
```

➢ Effect of Several Assignments

- What are the values of a and b after all statements are executed?

```
        int a = 1;

        int b;

        b = a;

        a = 2;
```

- **a = 2, b = 1**

➢ Type Compatibility

- as a rule, you cannot store a value of one type in a variable of another type
- trying to do it leads to *type mismatch*

```
int intvar;
intvar = 2.99;
```

- compiler prints this:

```
warning: assignment to ’int' from ’double'
```

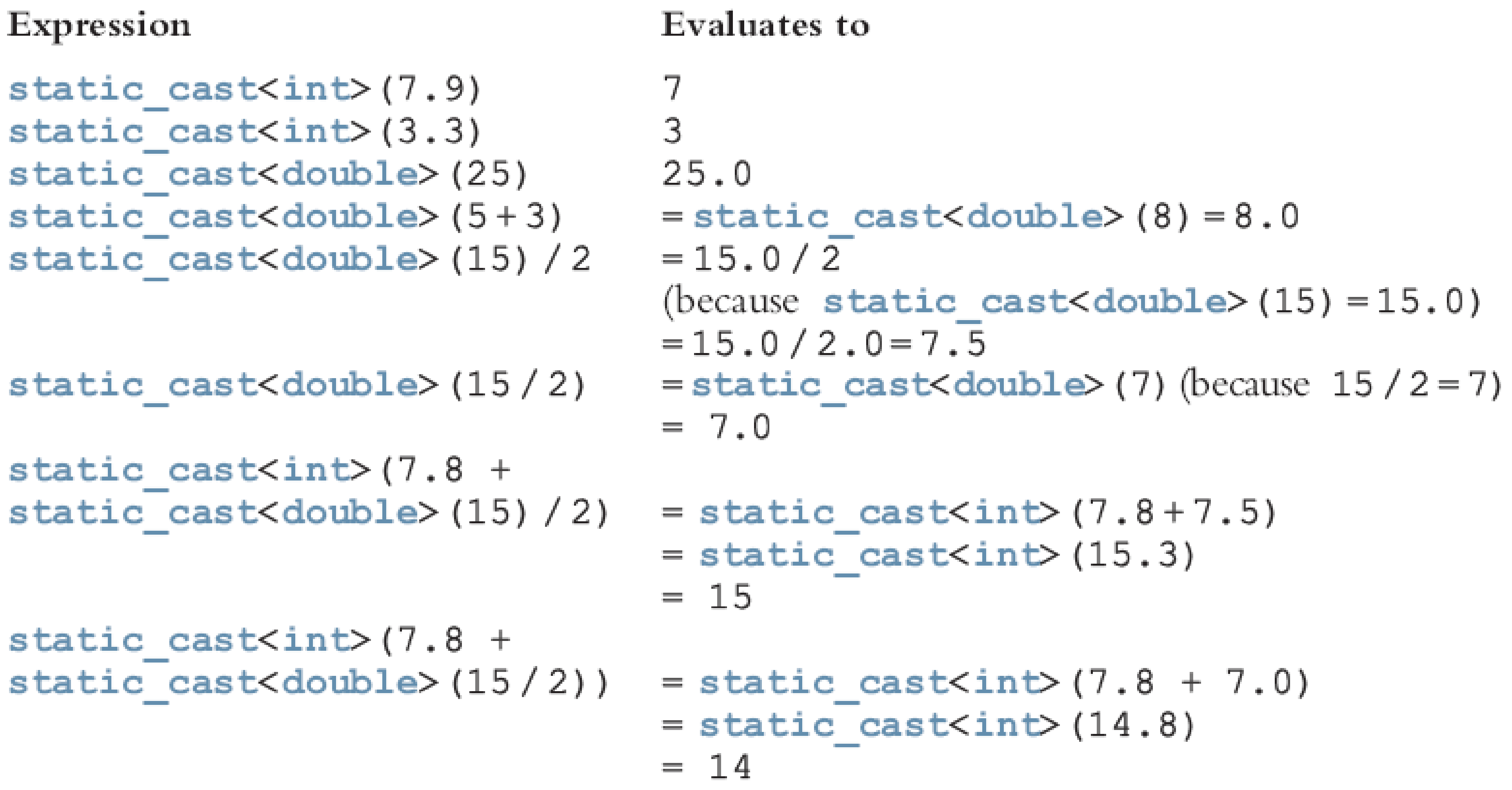
- but still compiles the program discarding the fractional part
- it is usually a bad idea (but some programmers do it) to store char values in variables of type int, bools can also be in int. Even though you compiler allows it, it obscures the meaning of the variables and should be avoided

➢ Type Conversion (Casting)

- Implicit type coercion: when value of one type is automatically changed to another type
- Cast operator: provides explicit type conversion

```
static_cast<dataTypeName>(expression)
```

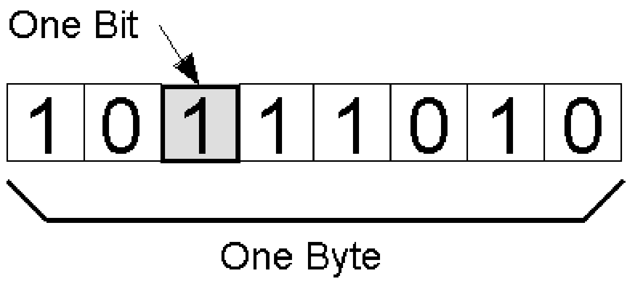
➢ Type Conversion (Casting)

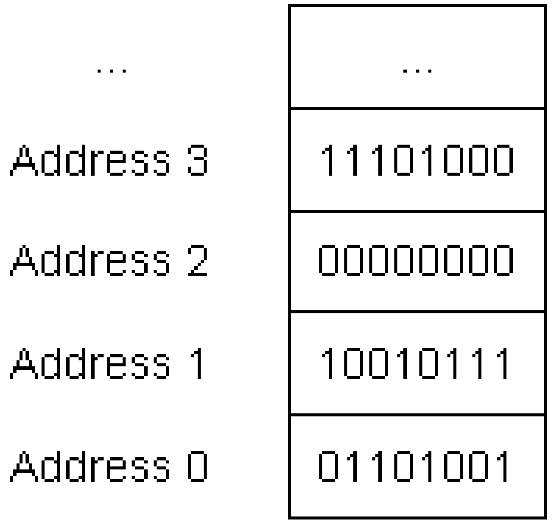


Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5+3)`	`= static_cast<double>(8) = 8.0`
`static_cast<double>(15) / 2`	`= 15.0 / 2`
	(because `static_cast<double>(15) = 15.0)`
	`= 15.0 / 2.0 = 7.5`
`static_cast<double>(15 / 2)`	`= static_cast<double>(7)` (because `15 / 2 = 7)`
	`= 7.0`
`static_cast<int>(7.8 +`	
`static_cast<double>(15) / 2)`	`= static_cast<int>(7.8+7.5)`
	`= static_cast<int>(15.3)`
	`= 15`
`static_cast<int>(7.8 +`	
`static_cast<double>(15 / 2))`	`= static_cast<int>(7.8 + 7.0)`
	`= static_cast<int>(14.8)`
	`= 14`

➢ Variables

- ❑ **Variables** are names for a piece of memory that can be used to store or manipulate information.
- ❑ Each **variable** in C++ has a specific **type**, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- ❑ Typically, we refer to a variable's "data type" as its "type"





Arithmetic

- Arithmetic is performed with operators.
- Arithmetic operators are listed in following table

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	`x / y`
Modulus	`%`	$r \bmod s$	`r % s`

division

7 % 5 evaluates to 2

- Integer division truncates remainder

7 / 5 evaluates to 1

- there is no remainder operator with floating-point operands

Results of Arithmetic operators

- Arithmetic operators can be used with any numeric type.
- An **operand** is a number or variable used by the operator e.g.
  - integer1 + integer2
    - + is operator
    - integer1 and integer2 are operands
- Result of an operator depends on the types of operands
  - If both operands are int, the result is int
  - If <u>one or both</u> operands are double, the result is double
  - If operator has both types of operands
    - Integer is changed to floating-point
    - Operator is evaluated
    - Result is floating-point

Examples comparing mathematical and C++ expressions

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

Summarizing increment and decrement operators

Operator	Called	Sample expression	Explanation
++	preincrement	**++a**	Increment **a** by 1, then use the new value of **a** in the expression in which **a** resides.
++	postincrement	**a++**	Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1.
--	predecrement	**--b**	Decrement **b** by 1, then use the new value of **b** in the expression in which **b** resides.
--	postdecrement	**b--**	Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1.

The associativity of these unary operators is from right to left

Increment and Decrement Examples

```
int k;

 k=6;

int i; i= k++;                        // i is 6, k is 7

cout << i << " " << k << endl;

int j; j= ++k;                        // j is 8, k is 8

cout << j << " " << k << endl;
```

Rules of operator precedence

Some arithmetic operators act before others (e.g., multiplication before addition)

opertors	opeertions	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 are evaluated left to right.
***, /,** or **%**	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or **-**	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Operator Precedence

An example to understand operator precedence.

```
20 -  4 / 5  * 2   +   3 * 5  % 4
```

```
            (4 / 5)
           ((4 / 5) * 2)
           ((4 / 5) * 2)     (3 * 5)
           ((4 / 5) * 2)    ((3 * 5) % 4)
      (20 -((4 / 5) * 2))   ((3 * 5) % 4)
      (20 -((4 / 5) * 2)) + ((3 * 5) % 4)
```

Assignment expression abbreviations

- Program can be written and compiled a bit faster by the use of abbreviated assignment operators
- C++ provides several assignment operators for abbreviating assignment expressions.
- **Addition assignment operator**

  c = c + 3; abbreviated to
  c += 3;
- Statements of the form

  variable = variable operator expression;

  can be rewritten as

  variable operator= expression;

Assignment operator	Sample expression	Explanation	Assigns
Assume: **int c = 3, d = 5, e = 4, f = 6, g = 12;**			
+=	**c += 7**	**c = c + 7**	**10** to **c**
-=	**d -= 4**	**d = d - 4**	**1** to **d**
***=**	**e *= 5**	**e = e * 5**	**20** to **e**
/=	**f /= 3**	**f = f / 3**	**2** to **f**
%=	**g %= 9**	**g = g % 9**	**3** to **g**

Programming Style

In order to improve the readability of your program, use the following conventions:

- Start the program with a header that tells what the program does.
- Use meaningful variable names.
- Document each variable declaration with a comment telling what the variable is used for.
- Place each executable statement on a single line.
- A segment of code is a sequence of executable statements that belong together.
  - Use blank lines to separate different segments of code.
  - Document each segment of code with a comment telling what the segment does.

Thank you