C++ Programming Essentials

Unit 1: Sequential Programming

CHAPTER 3: NUMERIC TYPES, EXPRESSIONS

DR. ERIC CHOU

IEEE SENIOR MEMBER

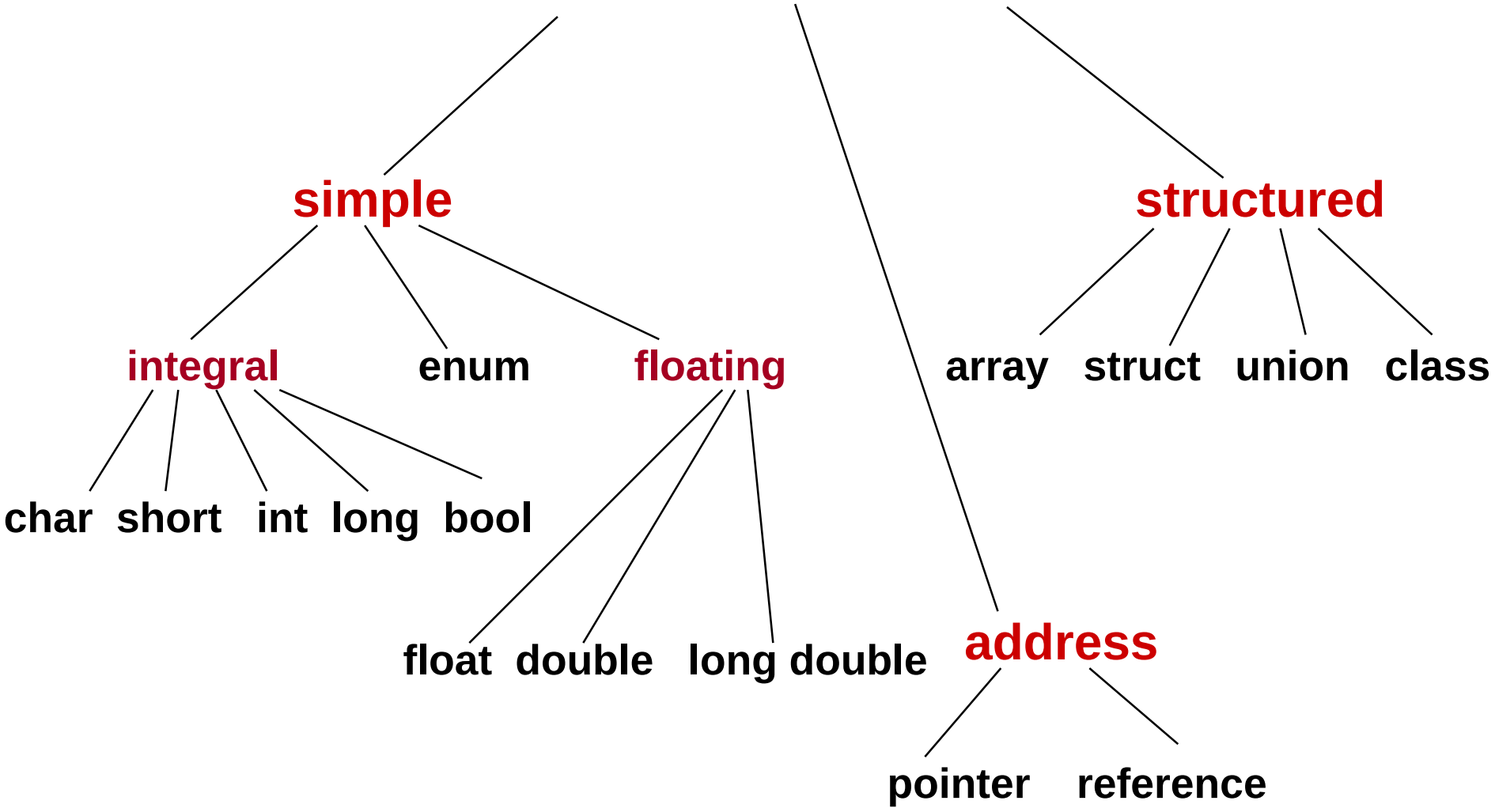
LECTURE 1

C++ Numerical Data Types

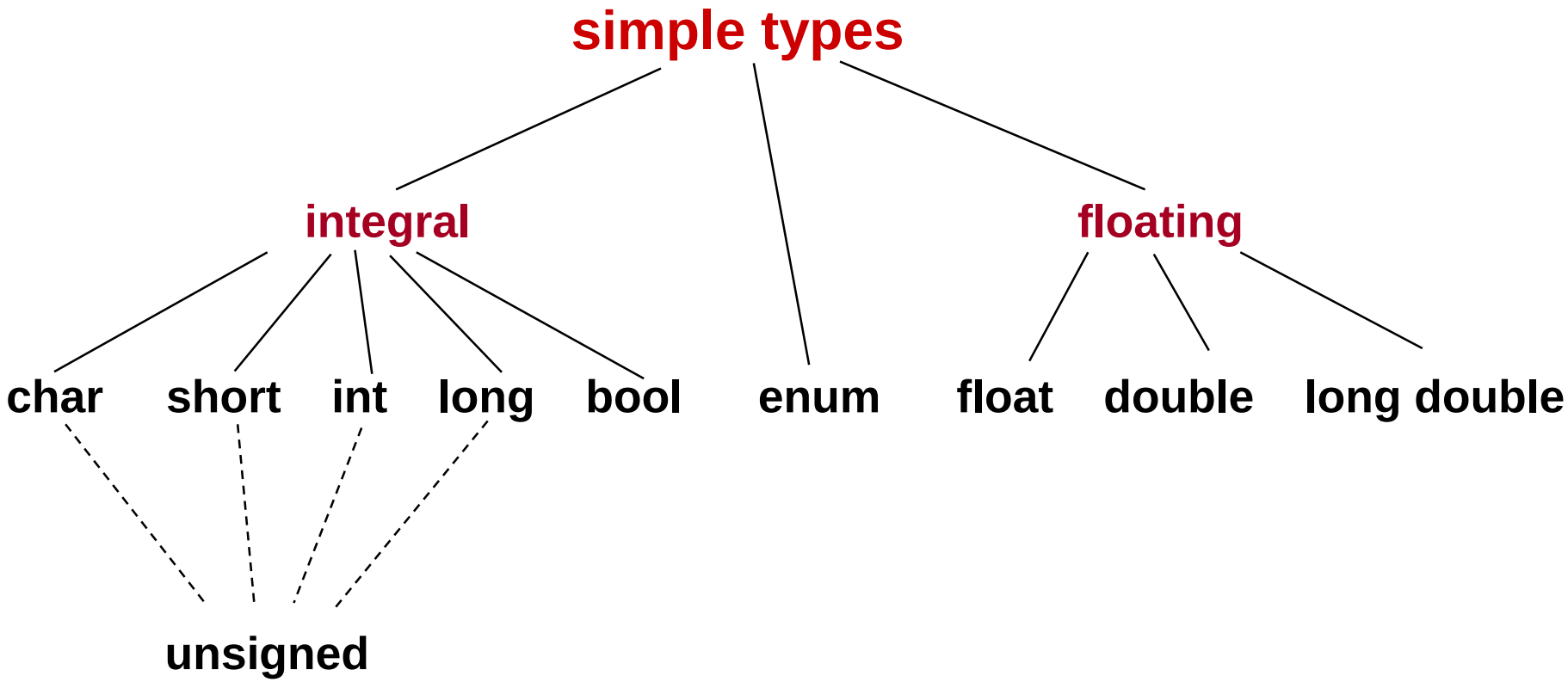
Chapter 3 Topics

- Constants of Type int and float
- Evaluating Arithmetic Expressions
- Implicit Type Coercion and Explicit Type Conversion
- Additional C++ Operators
- Operator Precedence
- Type Coercion in Arithmetic and Relational Precedence
- Calling a Value-Returning Function

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++ Simple Data Types



Standard Data Types in C++

- Integral Types
  - represent whole numbers and their negatives
  - declared as int, short, or long
- Floating Types
  - represent real numbers with a decimal point
  - declared as float, or double
- Character Type
  - represents single characters
  - declared as char

Samples of C++ Data Values

- **int** sample values

  4578 -4578 0

- **float** sample values

  95.274 95. .265

  9521E-3 -95E-1 95.213E2

- **char** sample values

  ‘B’ ‘d’ ‘4’ ‘?’ ‘*’

Scientific Notation

2.7E4 means $2.7 \times 10^{4} =$

2.7000 = 27000.0

2.7E-4 means $2.7 \times 10^{-4} =$

0002.7 = 0.00027

More About Floating Point Values

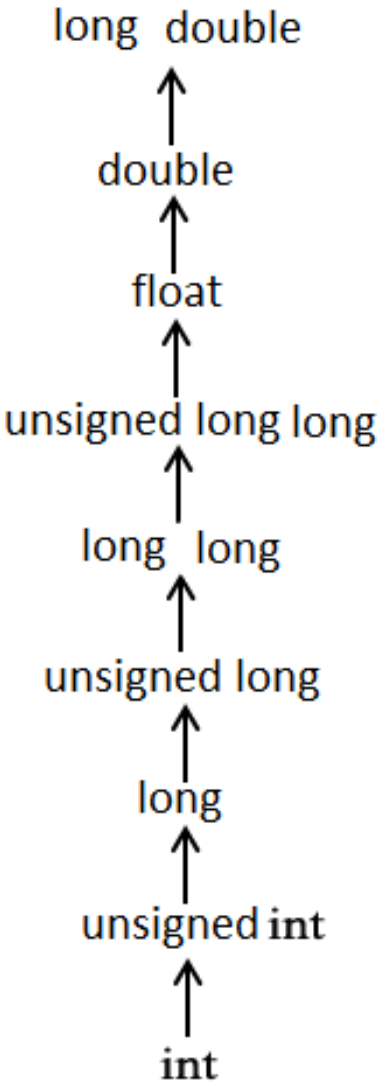
- floating point numbers have an integer part and a fractional part, with a decimal point in between. Either the integer part or the fractional part, but not both, may be missing
- EXAMPLES 18.4 500. .8 -127.358

- alternatively, floating point values can have an exponent, as in scientific notation--the number preceding the letter E doesn't need to include a decimal point
- EXAMPLES 1.84E1 5E2 8E-1 -.127358E3

Data type	Size(bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65535
int	4	-2147483648 to +2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to +2147483647
Unsinged long	4	0 to 4294967295
float	4	-3.4e-38 to +3.4e-38
double	8	1.7 e-308 to 1.7 e+308
long double	8	1.7 e-308 to 1.7 e+308
bool	1 bit	
void	-	-
wchar_t	2 or 4	1 wide character

Usual Arithmetic Conversion

- The **usual arithmetic conversions** are implicitly performed to cast their values to a common type.
- The compiler first performs *integer promotion*; if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy –



Demo Program:

implicit.cpp

Go Dev C++!!!

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char** argv) {
    int  i = 17;
    char c = 'c'; /* ascii value is 99 */
    float sum;

    sum = i + c;
    cout << "Value of sum : " << sum << endl;
    return 0;
}
```

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch3\implicit\implicit.exe
Value of sum : 116
```

LECTURE 2

Arithmetic Operators and Function's Return Value

Division Operator

- The result of the division operator depends on the type of its operands
- If one or both operands has a floating point type, the result is a floating point type. Otherwise, the result is an integer type

Examples

11 / 4	has value	2
11.0 / 4.0	has value	2.75
11 / 4.0	has value	2.75

Modulus Operator

- the modulus operator % can only be used with integer type operands and always has an integer type result
- its result is the integer type remainder of an integer division

EXAMPLE

11 % 4 has value 3 because

$$4 \overline{)\ 11} \qquad R = ?$$

Integer Division and Modulus

Integers **a**, **b**, **q**, **r**.

```
int a, b, q, r;
```

Assuming **a = b * q + r;** // **q** is quotient, **r** is remainder, **b** is divisor.

In C ++ language,

```
q = a / b;     // q is the quotient
r = a % b;     // r is the remainder
```

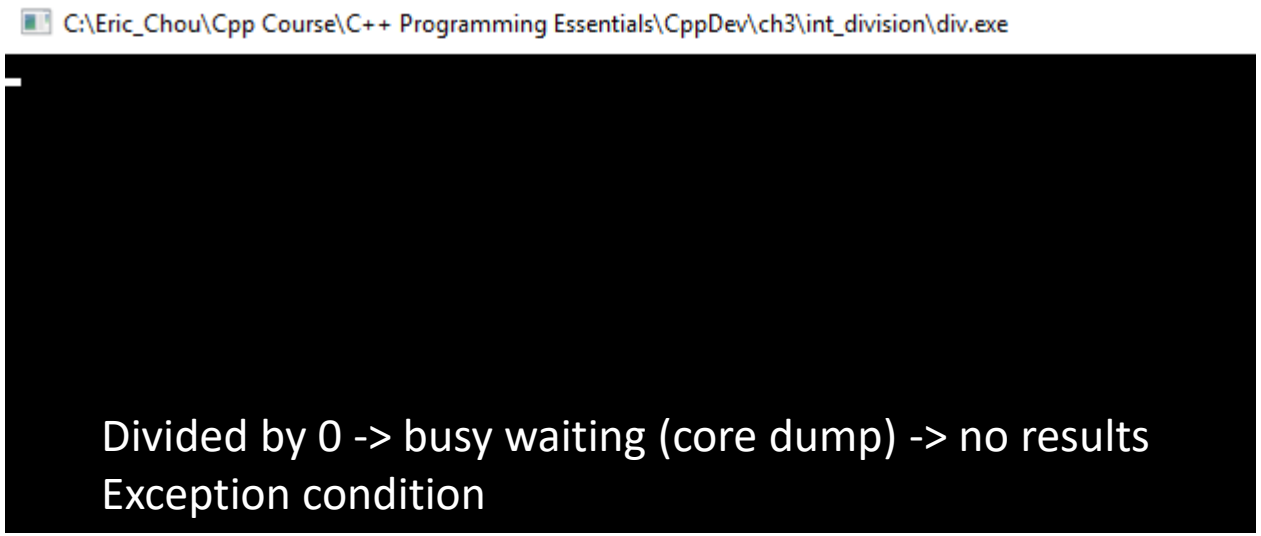
Note: quotient is the result of **a** divided by **b** using integer division rule.

Demo Program:

intdiv.cpp

Go Dev C++!!!

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main(int argc, char** argv) {
6       int a= 10;
7       int b= 0;
8
9       cout << (a/b) << endl;
10      return 0;
11  }
```



Demo Program:

doublediv.cpp

Go Dev C++!!!

```cpp
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    double a= 10;
    double b= 0;

    cout << (a/b) << endl;
    return 0;
}
```

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch3\double_division\DoubleDivision.exe
inf

Program finished with Inf (infinity) as result.
```

Main returns an int value to the operating system

```
//*************************************************************************
//   FreezeBoil  program
//   This program computes the midpoint between
//   the freezing and boiling points of water
//*************************************************************************
#include  < iostream >
using  namespace  std;

const   float   FREEZE_PT  =   32.0 ;       // Freezing point of water
const   float   BOIL_PT      =  212.0 ;       // Boiling point of water

int  main (  )
{
       float avgTemp ;              // Holds the result of averaging
                                    //  FREEZE_PT and BOIL_PT
```

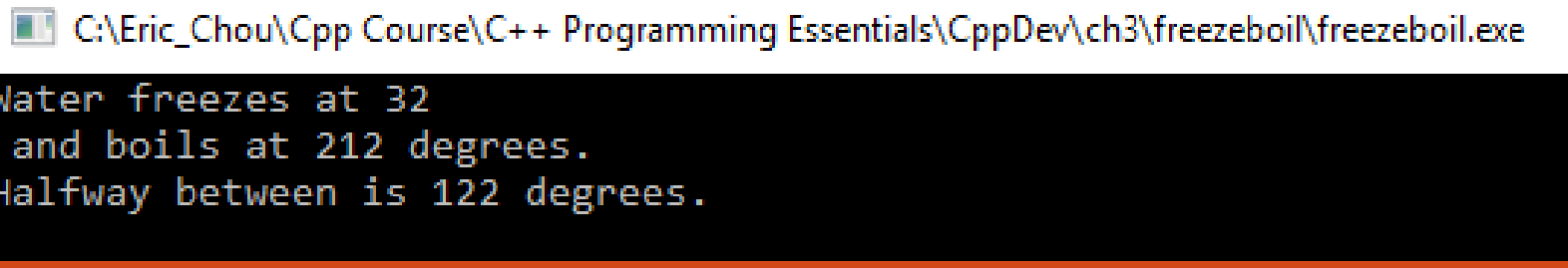
Function main Continued

```
    cout  <<  "Water freezes at " << FREEZE_PT  << endl ;
    cout  <<  " and boils at " << BOIL_PT  << " degrees."  << endl ;

    avgTemp  =  FREEZE_PT  +  BOIL_PT ;
    avgTemp  =  avgTemp  /  2.0 ;

    cout  <<  "Halfway between is " ;
    cout  <<  avgTemp  <<  " degrees."  <<  endl ;

    return  0 ;
}
```

Demo Program:

freezeboil.cpp

Go Dev C++!!!

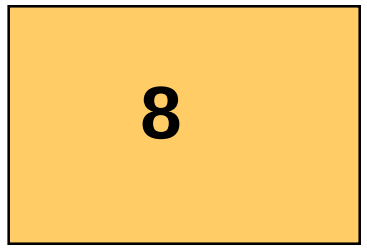


LECTURE 3

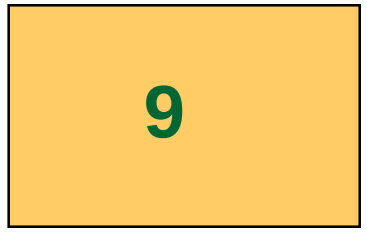
Post-Arithmetic and Pre-Arithmetic

More C++ Operators

```
int  age;

age = 8;

age = age + 1;
```



age

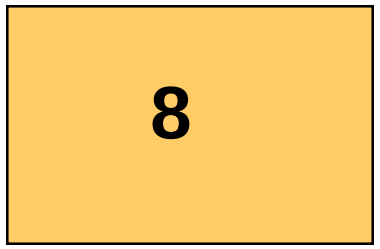


age

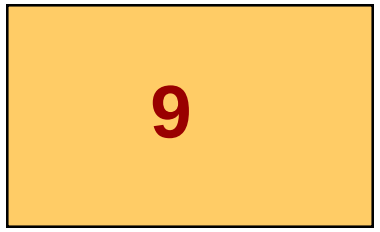
PREFIX FORM

Increment Operator (read as age after increment)

```
int  age;

age = 8;

++age;
```



age

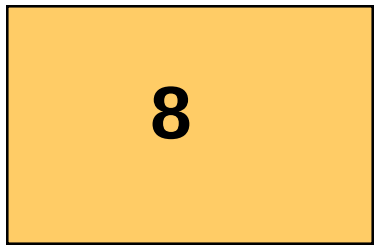


age

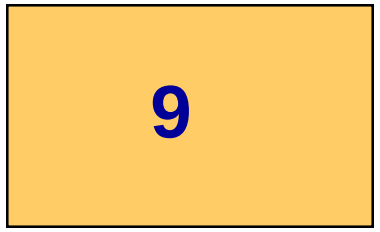
POSTFIX FORM

Increment Operator (read as age before increment)

```
int  age;

age = 8;

age++;
```



age

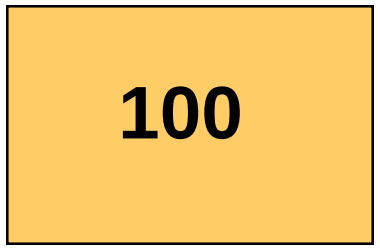


age

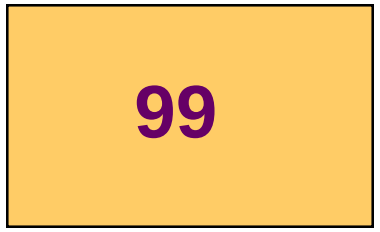
Decrement Operator

(read as dogs before decrement)

```
int  dogs;

dogs = 100;

dogs--;
```



dogs



dogs

Which Form to Use

- when the increment (or decrement) operator is used in a "*stand alone*" statement solely to add one (or subtract one) from a variable's value, it can be used in either prefix or postfix form

BUT…

- when the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield ***different*** results

WE'LL SEE HOW LATER . . .

LECTURE 4

C++ Expressions and Operators

What is an Expression in C++?

- An expression is a valid arrangement of variables, constants, and operators.
- in C++ each expression can be evaluated to compute a value of a given type
- the value of the expression

  9.3 * 4.5 is 41.85

Operators can be

- binary involving 2 operands 2 + 3
- unary involving 1 operand - 3
- ternary involving 3 operands *a == 3 ? 1 : 0;*

Some C++ Operators

Precedence	Operator	Description
Higher	( )	Function call
	+	Positive
	-	Negative
	*	Multiplication
	/	Division
	%	Modulus (remainder)
	+	Addition
	-	Subtraction
Lower	=	Assignment

Precedence

- higher Precedence determines which operator is applied first in an expression having several operators

Associativity

- left to right Associativity means that in an expression having 2 operators with the same priority, the left operator is applied first
- in C++ the binary operators

  * , / , % , + , - are all left associative
- expression 9 - 5 - 1 means (9 - 5) - 1

  4 - 1

  3

Evaluate the Expression

```
                7 * 10 - 5 % 3 * 4 + 9
means          (7 * 10) - 5 % 3 * 4 + 9
                  70 - 5 % 3 * 4 + 9
                70 - (5 % 3) * 4 + 9
                  70 -  2  * 4 + 9
                  70 - ( 2 * 4) + 9
                     70 - 8  +  9
                     ( 70 - 8 ) + 9
                          62  + 9
                              71
```

Parentheses

- parentheses can be used to change the usual order
- parts in ( ) are evaluated first
- evaluate **(7 * (10 - 5) % 3) * 4 + 9**

(7 * 5 % 3) * 4 + 9

(35 % 3) * 4 + 9

2 * 4 + 9

8 + 9

17

LECTURE 5

Demo Program: Mileage Program

Mileage Program

```
/* This program computes miles per gallon given four amounts
   for gallons used, and starting and ending mileage.

    Constants:   The gallon amounts for four fillups.
                 The starting mileage.
                 The ending mileage.

   Output (screen)      The calculated miles per gallon.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

#include <iostream>

using namespace std;
```

C++ Code Continued

```
const  float  AMT1 = 11.7 ;       // Number of gallons for fillup 1
const  float  AMT2 = 14.3 ;       // Number of gallons for fillup 2
const  float  AMT3 = 12.2 ;       // Number of gallons for fillup 3
const  float  AMT4 =   8.5 ;      // Number of gallons for fillup 4

const  float  START_MILES = 67308.0 ;     // Starting mileage
const  float  END_MILES    = 68750.5 ;    // Ending mileage

int  main( )
{
      float mpg ;              // Computed miles per gallon

      mpg = (END_MILES - START_MILES) /
                  (AMT1 + AMT2 + AMT3 + AMT4) ;
```

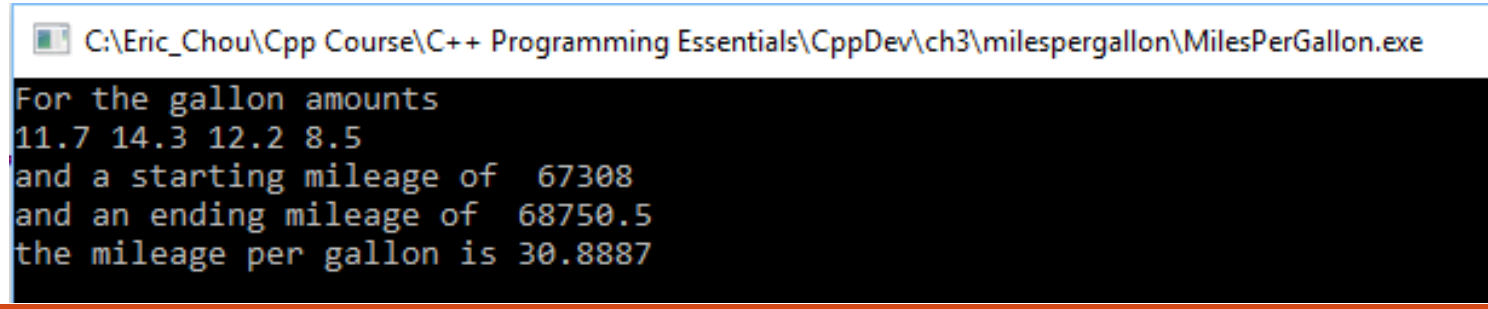
Main returns an int value to the operating system

```
        cout  <<  “For the gallon amounts “ << endl ;
        cout  <<  AMT1  <<  ‘ ‘ <<  AMT2  <<  ‘ ‘
              <<  AMT3  <<  ‘ ‘ <<  AMT4  <<  endl ;
        cout  <<  “and a starting mileage of  “
              << START_MILES  << endl ;

        cout  <<  “and an ending mileage of  “
              << END_MILES  << endl ;

        cout  <<  “the mileage per gallon is “ << mpg  << endl ;

        return 0;
}
```

Demo Program:

milespergallon.cpp

Go Dev C++!!!



LECTURE 6

Assignment Statements

Assignment Operator Syntax

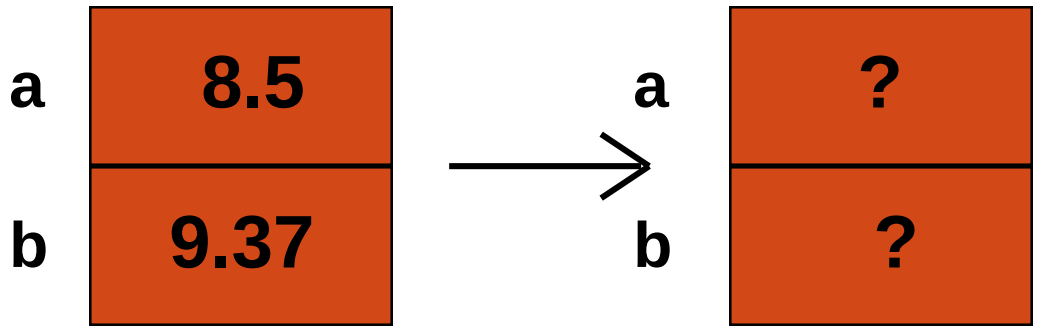
```
Variable  =  Expression
```

- first, Expression on right is evaluated
- then the resulting value is stored in the memory location of Variable on left

NOTE: An automatic type coercion occurs after evaluation but before the value is stored if the types differ for Expression and Variable

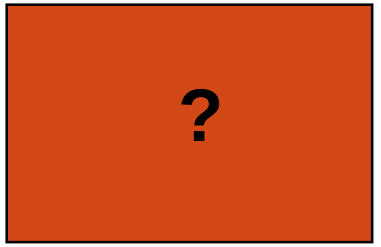
What value is stored?

```
float  a;
float  b;


a = 8.5;
b = 9.37;
a = b;
```



What is stored?

```
float  someFloat;
```



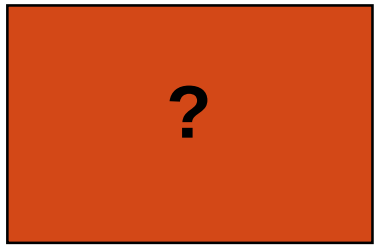
someFloat

```
someFloat = 12;    // causes implicit type conversion
```

someFloat

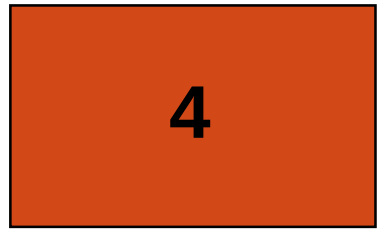
What is stored?

```
int  someInt;
```



someInt

```
someInt = 4.8;      // causes implicit type conversion
```



someInt

LECTURE 7

Type Casting and Rounding

Type Casting is Explicit Conversion of Type

int(4.8)	has value	4
float(5)	has value	5.0
float(7/4)	has value	1.0
float(7) / float(4)	has value	1.75

Some Expressions

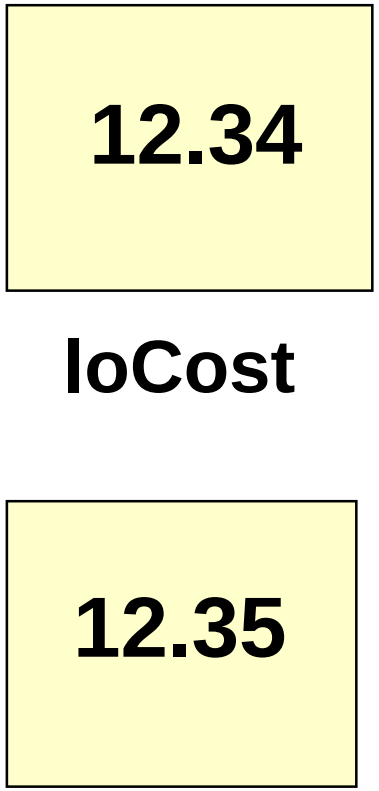
int age;

EXAMPLE	VALUE
age = 8	8
- age	- 8
5 + 8	13
5 / 8	0
6.0 / 5.0	1.2
float (4 / 8)	0.0
float (4) / 8	0.5
cout << “How old are you?”	cout
cin >> age	cin
cout << age	cout

What values are stored?

```
float  loCost;
float  hiCost;
loCost = 12.342;
hiCost  = 12.348;
loCost = float (int (loCost * 100.0 + 0.5) ) / 100.0;
hiCost  = float (int (hiCost * 100.0 + 0.5) ) / 100.0;
```

Values were rounded to 2 decimal places



Problem

- Given a character, a length, and a width, draw a box
- For example, given the values '&', 4, and 6, you would display

```
&&&&&&
&&&&&&
&&&&&&
&&&&&&
```

Demo Program:

draw.cpp

Go Dev C++!!!

```
1   #include <iostream>
2   using namespace std;
3   void draw(char ch, int length, int width){
4       for (int i=0; i<length; i++){
5           for (int j=0; j<width; j++){
6               cout << ch;
7           }
8           cout << endl;
9       }
10  }
11
12  int main(int argc, char** argv) {
13      char c = '&';
14      int l = 4;
15      int w = 6;
16      draw(c, l, w);
17      return 0;
18  }
```

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch3\draw\Draw.exe
&&&&&&
&&&&&&
&&&&&&
&&&&&&
```

LECTURE 8

Additional Operators

Additional C++ Operators

- Previously discussed C++ Operators include:
  - the assignment operator (=)
  - the arithmetic operators (+, -, *, /, %)
  - relational operators (==, !=, ,<=, >, >=)
  - logical operators (!, &&, ||)
- C++ has many specialized other operators seldom found in other programming languages

Additional C++ Operators

- Additional C++ Operators for a full list of:
  - Combined Assignment Operators
  - Increment and Decrement Operators
  - Bitwise Operators
  - More Combined Assignment Operators
  - Other Operators

Assignment Operators and Assignment Expressions

- (=) is the basic assignment operator
- Every assignment expression has a value and a side effect, the value that is stored into the object denoted by the left-hand side
- For example, delta = 2 * 12 has the value 24 and side effect of storing 24 into delta

Assignment Expressions

- In C ++, any expression become an expression statement when terminated by a semicolon
- The following are all valid C++ statements, first 2 have no effect at run time:

```
23;
2 * (alpha + beta);
delta = 2 * 12;
```

Increment and Decrement Operators

- The increment and decrement operators (++ and --) operate only on variables, not on constants or arbitrary expressions

1) Example of pre-incrementation

```
int1 = 14;
int2 = ++int1; // int1 == 15 && int2 == 15
```

2) Example of post-incrementation

```
int1 = 14;
int2 = int1++; // int1 == 15 && int2 == 14
```

Bitwise Operators

- Bitwise operators (e.g., <<, >>, and |) are used for manipulating individual bits within a memory cell
- << and >> are left and right shift operators, respectively that take bits within a memory cell and shift them to the left or the right
- Do not confuse relational && and || operators used in logical expressions with & and | bitwise operators

The Cast Operation

- Explicit type cast used to show that the type conversion is intentional
- In C++, the cast operation comes in three forms:

```
intVar = int(floatVar);                 // Functional notation
intVar = (int)floatVar;                 // Prefix notation
intVar = static_cast<int>(floatVar); // Keyword notation
```

The Cast Operation

- Restriction on use of functional notation: Data type name must be a single identifier
- If type name consists of more than one identifier, prefix notation or keyword notation must be used
- Most software engineers now recommend use of keyword cast because it is easier to find these keywords in a large program

The sizeof Operator

- **The sizeof** operator --a unary operator that yields the size, in bytes, of its operand
- The operand can be a variable name , as in **sizeof someInt**
- Alternatively, the operand can be the name of a data type enclosed in parentheses: **(sizeof float)**

The ? Operator

- ? : operator, also called the conditional operator is a three-operand operator
- Example of its use: set a variable max equal to the larger of two variables a and b.
- With the ?: operator , you can use the following assignment statement:

```
max = (a>b) ? a : b;
```

LECTURE 9

Operators' Precedence

Operator Precedence

- Following Table on slide 53 and slide 54 groups discussed operators by precedence levels for C++
- Horizontal line separates each precedence level from the next-lower level
- Column level Associativity describes grouping order.
- Within a precedence level, operators group Left to right or, Right to left

Operator	Associativity	Remarks
`()` `++` `--`	Left to right Right to left	Function call and function-style cast `++`and `--` as postfix operators

`++` `--`
`!` Unary
`+`Unary
`(`cast`)`
`sizeof`
	Right to left
	Right to left
		`++`and `--` as prefix operators

`*` `/` `%`	Left to right	
`+` `-`	Left to right	

Operator	Associativity	Remarks
< <= > >=	Left to right	
== !=	Left to right	
&&	Left to right	
||	Left to right	
?:	Right to left	
= += -= *= /=	Right to left	

LECTURE 10

Type Coercion and Relational Expression

Type Coercion in Arithmetic and Relational Expressions

- If two operands are of different types, one of them is temporarily **promoted** (or **widened**) to match the data type of the other
- Rule of type coercion in an arithmetic coercion:

Step 1: Each char, short, bool, or enumeration value is promoted (widened) to int. If both operands are now int, the result is an int expression.

Type Coercion in Arithmetic and Relational Expressions

Step 2: If Step 1 still leaves a mixed type expression, the following precedence of types is used:

int, unsigned int, long, float, double, long double

Type Coercion in Arithmetic and Relational Expressions

Example: expression someFloat+2

- Step 1 leaves a mixed type expression
- In Step 2, int is a lower type than the float value---for example, 2.0
- Then the addition takes place, and the type of the entire expression is float

Type Coercion in Arithmetic and Relational Expressions

- This description also holds for relational expressions such as someInt <= someFloat
- Value of someInt temporarily coerced to floating-point representation before the comparison occurs
- Only difference between arithmetic and relational expressions:
- Resulting type of relational expression is always bool---the value true or false