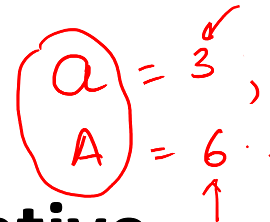
Fundamentals of Programming
CS-114

Lecture 3

What is an Identifier?

- **An identifier** is the **name** to denote **labels, types, variables, constants** or **functions,** in a C++ program.
- C++ is a **case-sensitive** language.
  - **Work** is not **work**



- **Identifiers should be descriptive**
  - Using meaningful identifiers is a good programming practice

Identifier

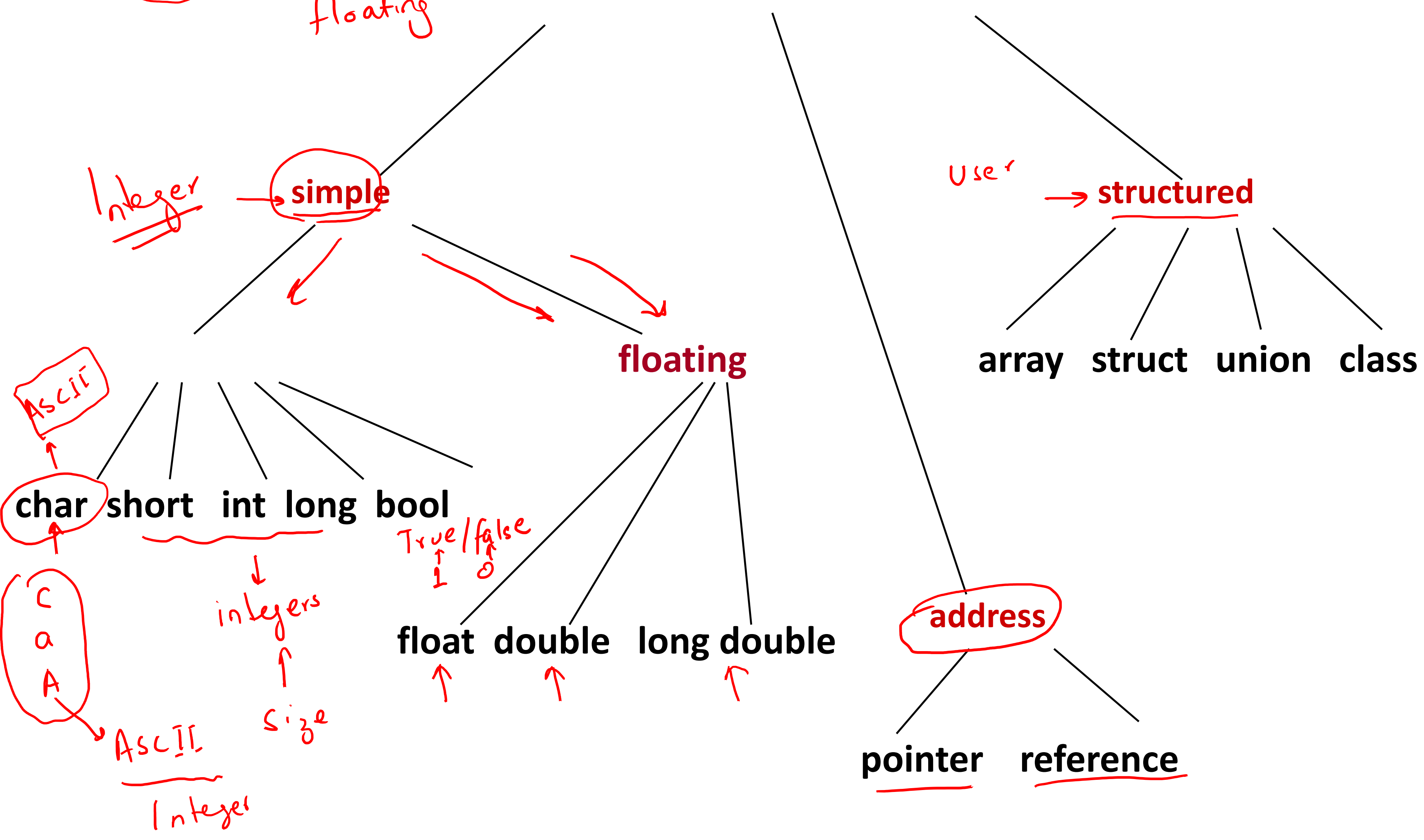
- Identifiers must be unique
- Identifiers cannot be reserved words (keywords)
  - **double main return**
- Identifier must start with a letter or underscore, and be followed by zero or more letters (A-Z, a-z), digits (0-9), or underscores
- **VALID**

age_of_dog **_taxRateY2K**
PrintHeading **ageOfHorse**

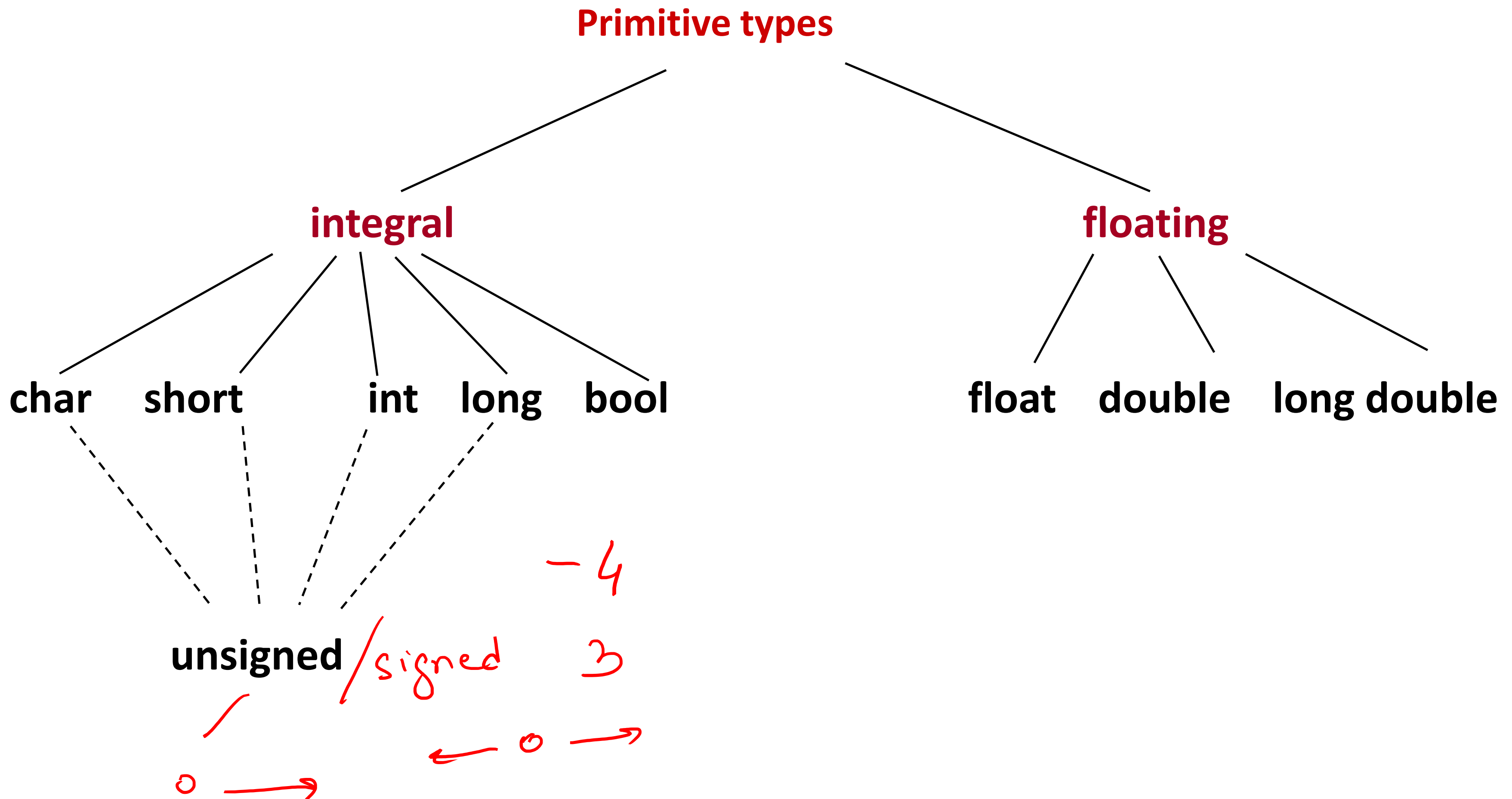
- **NOT VALID**

age# **2000TaxRate** **Age-Of-Dog** **main**

C++ Data Types



C++ Primitive Data Types



Premitive Data Types in C++

- **Integral Types**
  - represent whole numbers and their negatives
  - declared as int, short, or long
- **Character Types**
  - represent single characters
  - declared as char
  - Stored by ASCII values
- **Boolean Type**
  - declared as bool
  - has only 2 values true/false
  - will not print out directly
- **Floating Types**
  - represent real numbers with a decimal point
  - declared as float, or double
  - Scientific notation where e (or E) stand for “times 10 to the ” (.55-e6)

Samples of C++ Data Values

sample values

4578 **-4578** **0**

values

true **false**

sample values

95.274 **95.0** **.265**

sample values

'B' **'d'** **'4'** **'?'** **'*'**

integer

int

bool

char

float

boolean

float

char

Samples of C++ Data Values

int sample values

4578 **-4578** **0**

bool values

true **false**

float sample values

95.274 **95.0** **.265**

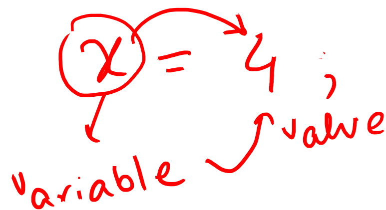
char sample values

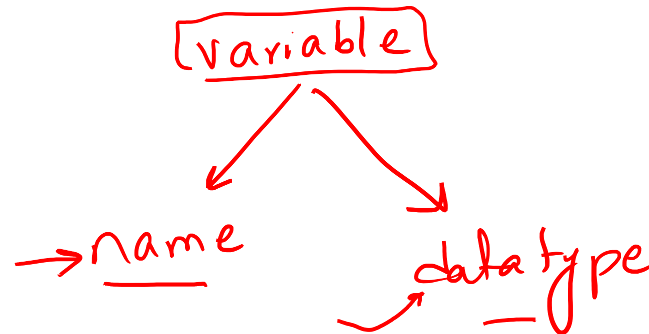
'B' **'d'** **'4'** **'?'** **'*'**

What is a Variable?

- A variable is a memory address where data can be stored and changed.

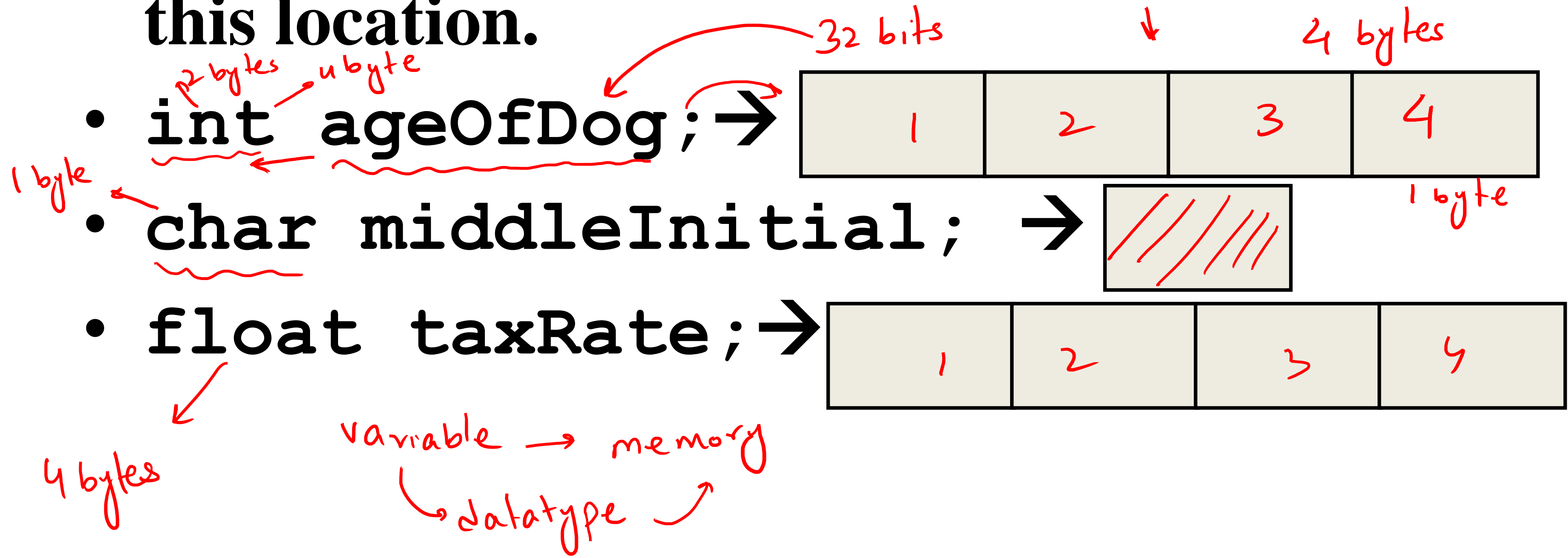
- Declaring a variable means specifying both its name and its data type.





What Does a Variable Declaration Do?

- A declaration tells the compiler to **allocate enough memory** to hold a value of this data type, and to **associate the identifier** with this location.



Variable Declaration

- All variables must declared **before** use.
  - At the top of the program
  - Just before use.
- Commas are used to separate identifiers of the same type.

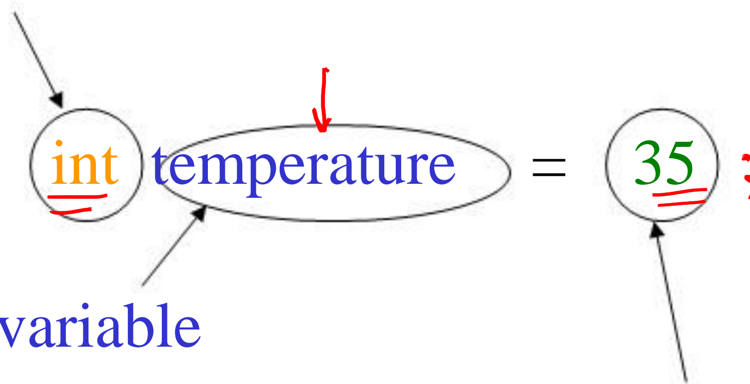
```
int count, age;
```

- Variables can be initialized to a starting value when they are declared

```
int count = 0;
int age, count = 0;
```

An Example of a Variable

Type of the variable is integer (written as "int" in C++)



A name of the variable

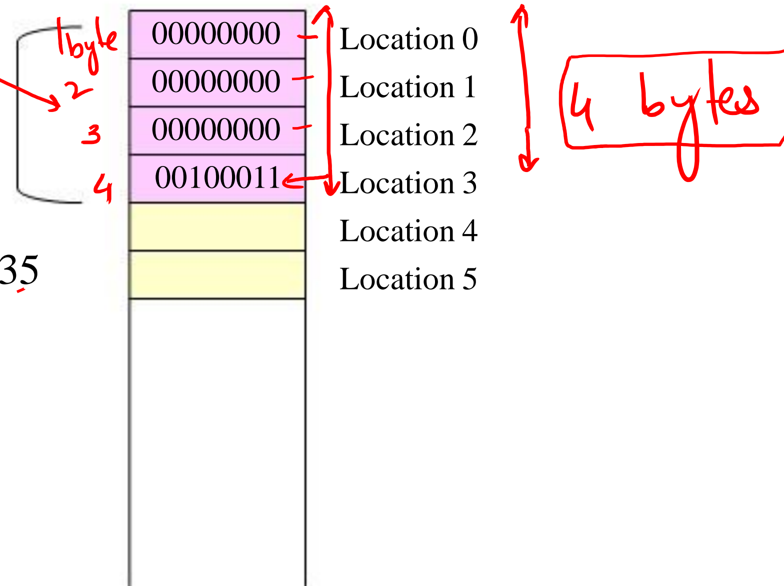
An initial value of the variable

Example of a Variable (Memory View)

int temperature = 35

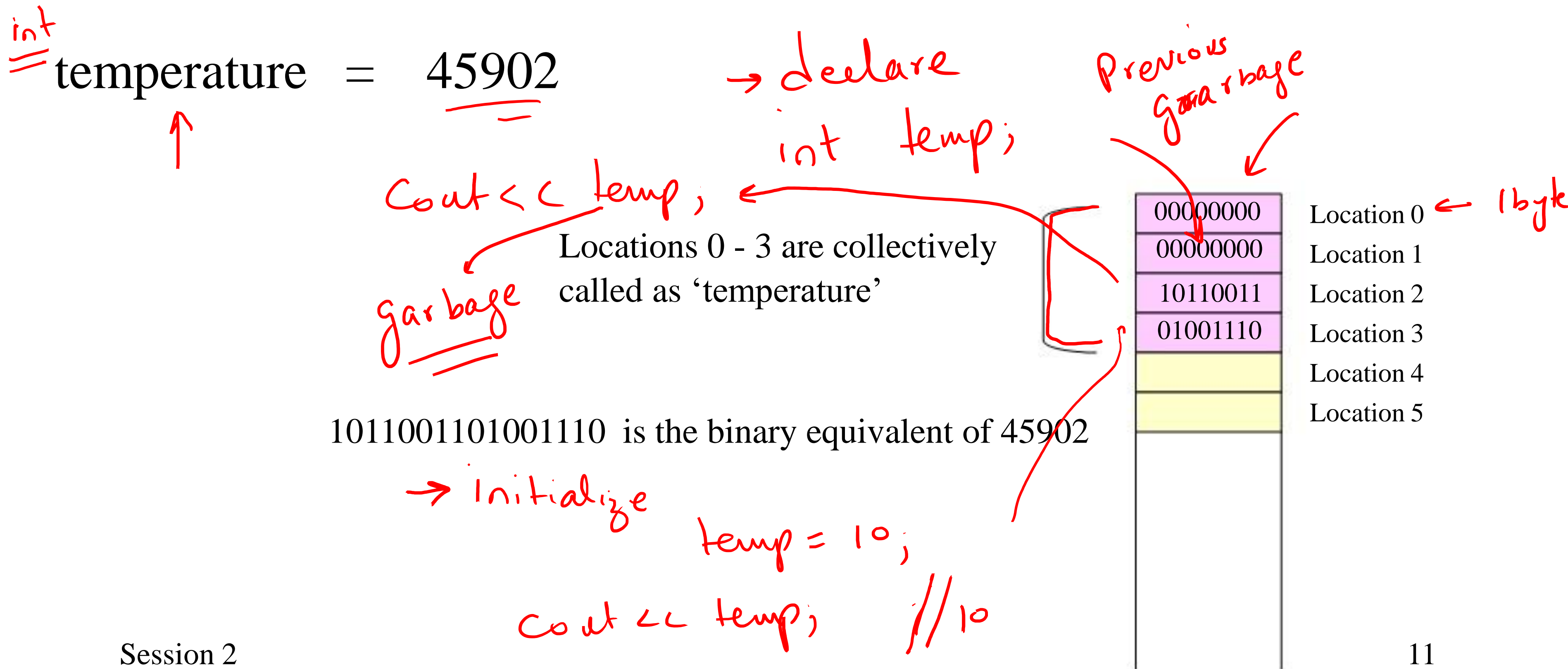
Locations 0 - 3 are collectively called as 'temperature'

100011 is the binary equivalent of 35

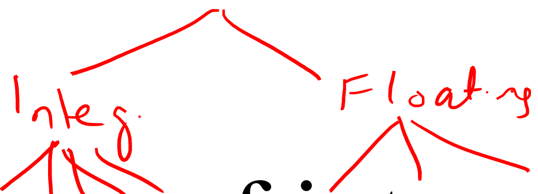


Changing the Value of Variable

- Lets change the value of ‘temperature’.



Relative Comparison of int and double

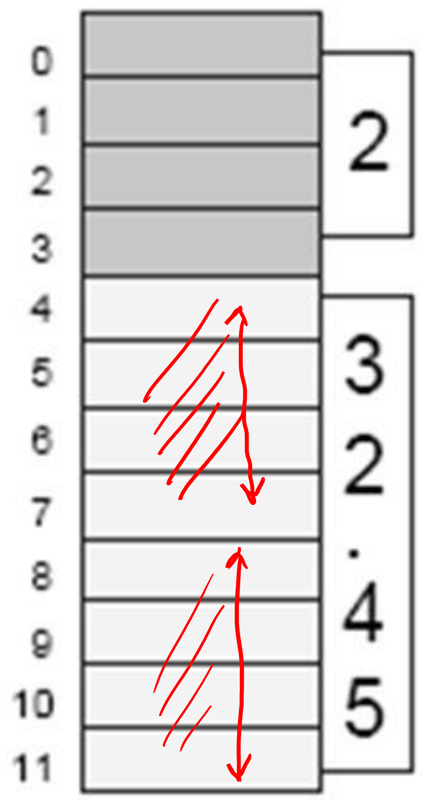


int numPeople = 2;

Reserves 32 bits (4 bytes) and sets the value stored in that space to 2. The name 'numPeople' is associated with this space.

double bill = 32.45;

Reserves 64 bits (8 bytes) and sets the value stored in that space to 32.45. The name 'bill' is associated with this space.



Commonly used data types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535

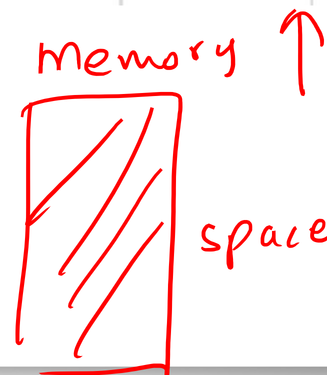
Type	Storage size	Value range
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

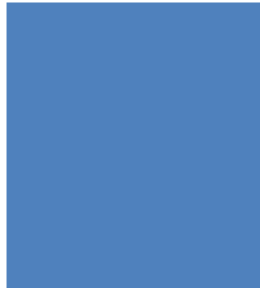
Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

signed unsigned

long int
4+4=8

c++
long





Variable types

Variable type	Keyword used in declaration	Size in bits	Range
integer	int	32 bits	-2147483648 to 2147483647
Short integer	short int	16 bits	-32768 to 32767
Long integer	long int	32 bits	-2147483648 to 2147483647
Floating point data	float	32 bits	$-1.0x10^{38}$ to $1.0x10^{38}$
Floating point data (with large fraction)	double	64 bits	$-1.0x10^{308}$ to $1.0x10^{308}$
Text type data	char	8 bits	-128 to 127
Boolean data (True or False)	bool	1 bit	1 or 0

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
- In C++, an expression can be:
  - A variable or a constant (count, 100)
  - An operation (a + b, a * 2)
  - Function call (getRectangleArea(2, 4))

Assignment Operator

- An operator to give (assign) a value to a variable.
- Denote as ‘=‘
- Only **variable** can be on the left side.
- An expression is on the right side.
- Variables keep their assigned values until changed by another **assignment statement** or by **reading in** a new value.

x = a + c ;
variable

Assignment Operator Syntax

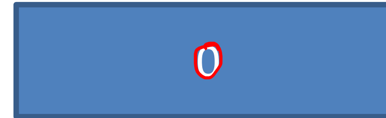
- Variable = Expression
  - First, expression on right is **evaluated**.
  - Then the resulting value is **stored** in the memory location of Variable on left.

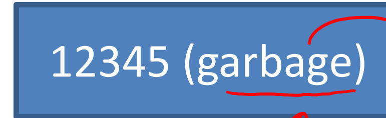
NOTE: An automatic type coercion occurs after evaluation but before the value is stored if the types differ for Expression and Variable

Assignment Operator Mechanism

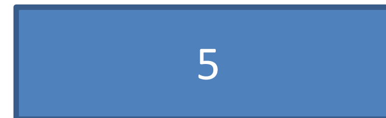
- Example:

```
int count = 0;
int starting;
starting = count + 5;
```





- Expression evaluation:
  - Get value of **`count`**: 0
  - Add 5 to it.
  - Assign to **`starting`**

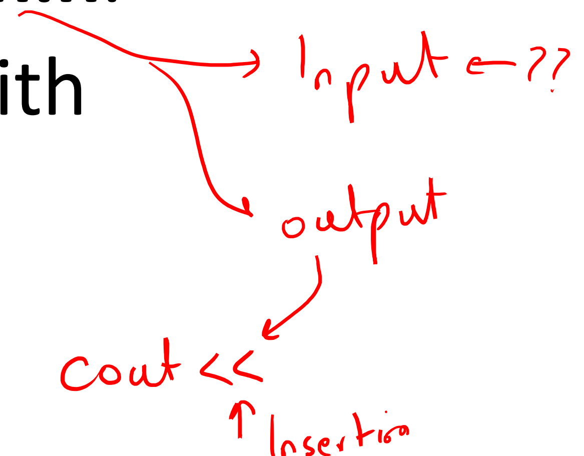


Input and Output

- C++ treats input and output as a **stream** of characters.
- **stream** : sequence of characters (printable or nonprintable)
- The functions to allow standard I/O are in **iostream** header file or **iostream.h**.
- Thus, we start every program with

  #include <iostream>

  using namespace std;

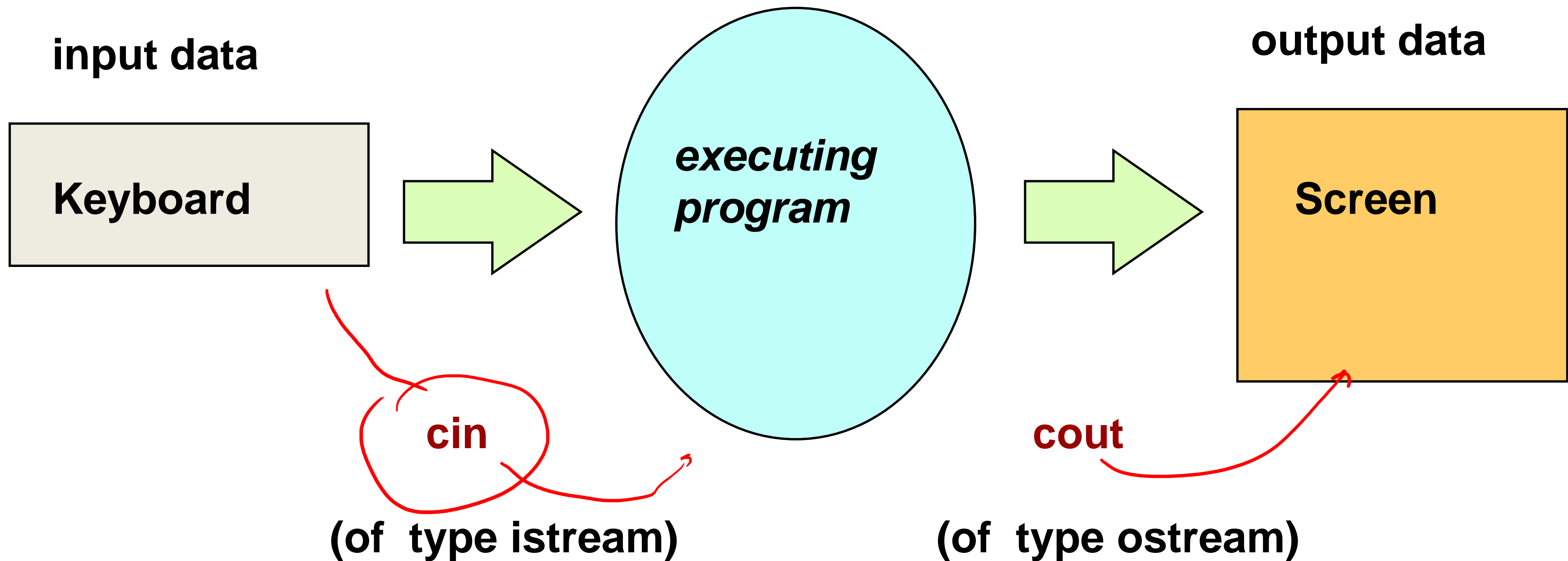


Include Directives and Namespaces

- **include:** directive copies that file into your program
- **namespace:** a collection of names and their definitions. Allows different namespaces to use the same names without confusion

Keyboard and Screen I/O

#include <iostream>



Insertion Operator (<<)

- Variable **cout** is predefined to denote an output stream that goes to the standard output device (display screen).

- The insertion operator **<<** called "put to" takes 2 operands.

- The left operand is a stream expression, such as **cout**. The right operand is an **expression** of simple type or a **string constant**.

Extraction Operator (>>)

- Variable **cin** is predefined to denote an input stream from the standard input device (the keyboard)
- The extraction operator **>>** called "get from" takes 2 operands. The left operand is a **stream expression**, such as **cin**--the right operand is a variable of simple type.
- Operator **>>** attempts to extract the next item from the input stream and store its value in the right operand variable.

Input Statements

SYNTAX

```
cin >> Variable >> Variable . . . ;
```

cin statements can be linked together using **>>** operator.

These examples yield the same output:

```
cin >> x;
cin >> y;
```

```
cin >> x >> y;
```

How Extraction Operator works?

- Input is not entered until user presses <ENTER> key.
- Allows backspacing to correct.
- Skips whitespaces (space, tabs, etc.)
- Multiple inputs are stored in the order entered:

```
cin>>num1>>num2;
```

User inputs: **3 4**

Assigns num1 = 3 and num2 = 4

Type compatibilities

- **Warning:** If you store values of one type in variable of another type the results can be inconsistent:
  - Can store integers in floating point or in char (assumes ASCII value)
  - bool can be stored as int: (true = nonzero, false = 0)
- Implicit promotion: integers are promoted to doubles

  double var = 2; // results in var = 2.0

- On integer and doubles together:
  - Mixed type expressions: Both must be **int** to return **int**, otherwise **float**.

Type compatibilities (Implicit Conversion)

- The compiler tries to be value-preserving.
- General rule: promote up to the first type that can contain the value of the expression.
- Note that representation doesn't change but values can be altered .
- Promotes to the smallest type that can hold both values.
- If assign **float** to **int** will truncate

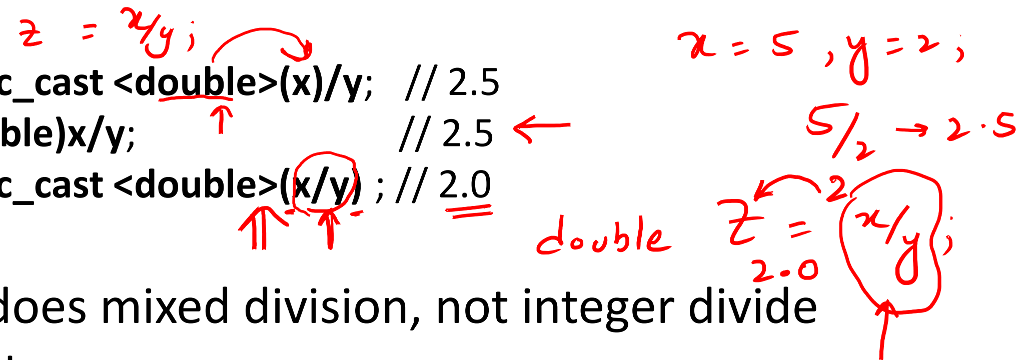
  int_variable = 2.99; // results in 2 being stored in int_variable
- If assign **int** to **float** will promote to double:

  double dvar = 2; // results in 2.0 being stored in dvar

Type compatibilities (Explicit Conversion)

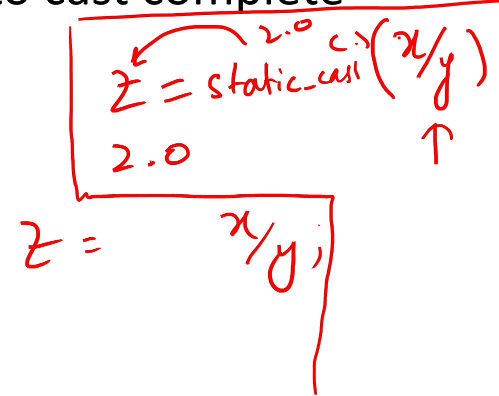
- Casting - forcing conversion by putting (type) in front of variable or expression. Used to insure that result is of desired type.
- Example: If you want to divide two integers and get a real result you must **cast** one to double so that a real divide occurs and store the result in a double.

```
int x=5, y=2; double z;   z = static_cast <double>(x)/y;   // 2.5
int x=5, y=2; double z;   z = (double)x/y;                 // 2.5
int x=5, y=2; double z;   z = static_cast <double>(x/y) ; // 2.0
```



- converts x to double and then does mixed division, not integer divide
- **static_cast<int> (z)** - will truncate z
- **static_cast <int> (x + 0.5)** - will round positive x {use () to cast complete expression)
- Cast division of integers to give real result:

```
int x=5, y=2; double z;   z = static_cast <double>(x/y) ; // 2.0
```



Operators

Operators

- “Operators are words or symbols that cause a program to do something to variables.”

OPERATOR TYPES:

Type	Operators	Usage
Arithmetic	‘+’ ‘-’ ‘*’ ‘/’ ‘%’	a+b a-b a*b a/b a%b
Arithmetic assignment	‘+=’ ‘-=’ ‘*=’ ‘/=’ ‘%=’	a+=b is the same as a=a+b a-=b a*=b a/=b a%=b
Increment and decrement	‘++’ ‘- -’	a++ is the same as a=a+1 a-- is the same as a=a-1
Relational	‘<’ ‘>’ ‘<=’ ‘>=’ ‘==’ ‘!=’	
Logical	‘&&’ ‘||’	

Manipulating Values

- Mathematical Operators
  - Common mathematical operators are available in C++ for manipulating values e.g. addition(+), subtraction(-), multiplication(*), division(/), and modulus (%).
- C++ has many other operators also which we will study in due course.

Arithmetic Expression Evaluation

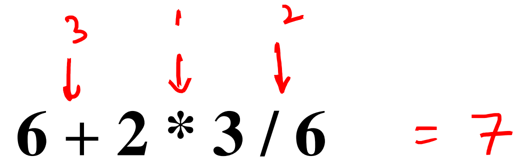
- To evaluate an arithmetic expression two concepts needs to be understood
  - Operator Precedence
    - Operator precedence controls the order in which operations are performed
  - Operator Associativity
    - The associativity of an operator specifies the order in which operations of the same precedence are performed

Operator Precedence and Associativity

- Operators Precedence and Associativity for C++ is following

1. *, /, % → Do all multiplications, divisions and remainders from left to right.
2. +, - → Do additions and subtractions from left to right.

Evaluating an Expression



- Three operators are in this expression.
- However, * and / both have the same precedence and + has lower precedence then these two.
- * and / will be evaluated first but both have the same precedence level.
- Therefore, operator associatively will be used here to determine the first to get evaluated i.e. left to right.
- The left most sub expression will be evaluated followed by the next right one and so on.

- * will be evaluated first then /

Arithemetic Operator

Type	Operators	Usage
Arithmetic	'+' '-' '*' '/' '%'	a+b a-b a*b a/b a%b

- The Modulus Operator
  - ***%*** is known as the Modulus Operator or the Remainder Operator.
  - It calculates the remainder of two variables
  - It can only be used with two ***ints..***

  - ***3%2 =1***
  - ***5%2=1***
  - ***6%3=0***
  - ***8%5=3***

I/p number → even / odd

x % 2 == 0

Arithmetic Operator Precedence and associativity

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

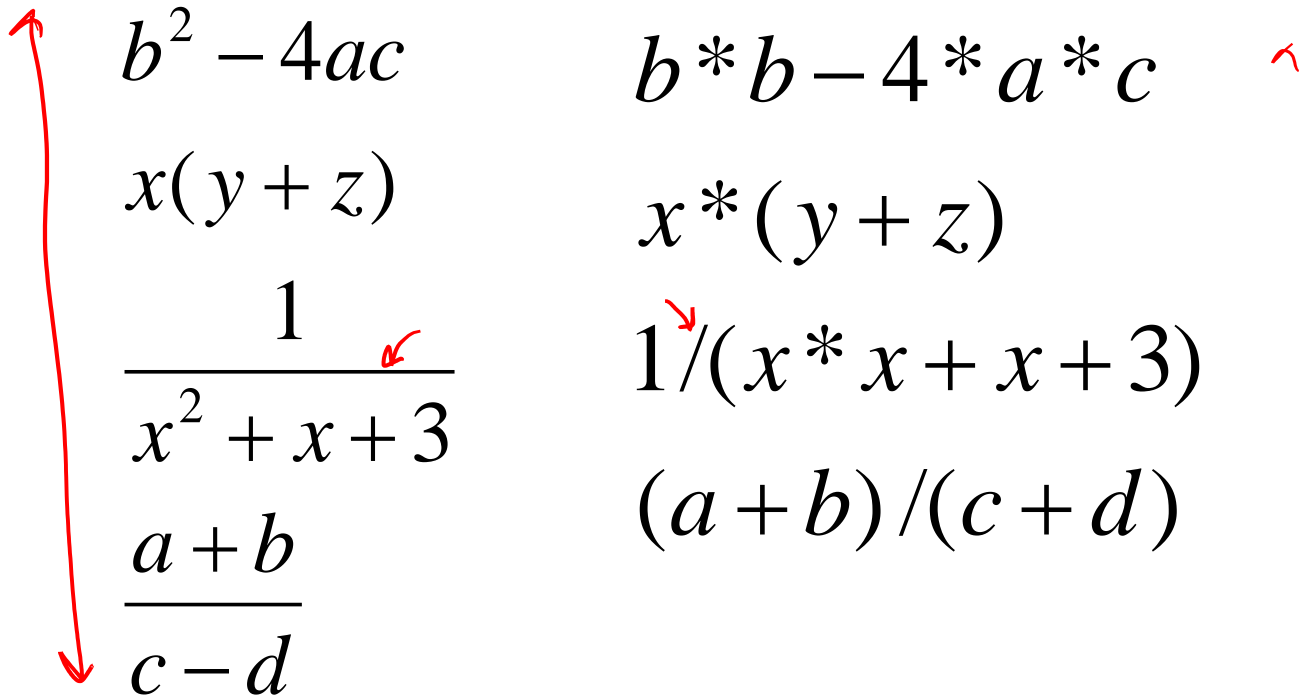
Precedence Chart

a++

- ++, --, !, - (unary minus), + (unary plus)
- *, /, %
- + (addition), - (subtraction)
- <<, >>
- <, <=, >, >=
- ==, !=
- &&
- ||
- =

Arithmetic Expressions

- Arithmetic operations can be used to express the mathematic expression in C++:



$$b^2 - 4ac \qquad b*b-4*a*c$$

$$x(y+z) \qquad x*(y+z)$$

$$\frac{1}{x^2+x+3} \qquad 1/(x*x+x+3)$$

$$\frac{a+b}{c-d} \qquad (a+b)/(c+d)$$

Polynomials in C++

- In algebra

$$m = \frac{a+b+c+d+e}{5}$$

- In c++

$$m = (a+b+c+d+e)/5$$

- What happens if I don’t put the parenthesis ()?

Evaluate

In what order will the expression be evaluated?

Algebra $z = pr\%q + w/x - y$

C++ $z = p * r\%q + w/x - y$

6 1 2 4 3 5

Evaluate

In what order will the expression be evaluated?

Run Code

2nd Degree Polynomial

$$y = ax^2 + bx + c$$

```
y = a * x * x + b * x + c;
```

6 1 2 4 3 5

2nd Degree Polynomial

$$y = ax^2 + bx + c$$

```
y = a * x * x + b * x + c;
```

6 1 2 4 3 5

Run Code

Arithmetic Assignment Operators

Type	Operators	Usage
Arithmetic assignment	‘+=’ ‘-=’ ‘*=’ ‘/=’ ‘%=’	a+=b is the same as a=a+b a-=b a*=b a/=b a%=b

Relational and Equality Operators

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

(Deitel and Deitel (5th Ed), fig 2.12)

Logical Operators

Type	Operators	Usage
Logical	‘&&’ ‘||’	

- Logical operators are carried out on statements, e.g. statement1 && statement 2, etc.
- Logical AND (&&)
  - false && false= false
  - false && true = false
  - true && false= false
  - true && true = true
- Logical OR (||)

Logical Operators

- Logical OR (||)
  - false || false = false
  - false || true = true
  - true || false = true
  - true || true = true
- Logical NOT (!)
  - !false = true
  - !true = false

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	
(5 != 6)	
(5 < 3) && (6 <= 6) || (5 != 6)	
(5 < 3) && ((6 <= 6) || (5 != 6))	
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

<table>
<tr><th>Expression</th><th>True or False</th></tr>
<tr><td>(6 <= 6) && (5 < 3)</td><td>False</td></tr>
<tr><td>(6 <= 6) || (5 < 3)</td><td>True</td></tr>
<tr><td>(5 != 6)</td><td></td></tr>
<tr><td>(5 < 3) && (6 <= 6) || (5 != 6)</td><td></td></tr>
<tr><td>(5 < 3) && ((6 <= 6) || (5 != 6))</td><td></td></tr>
<tr><td>!((5 < 3) && ((6 <= 6) || (5 != 6)))</td><td></td></tr>
</table>

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	
(5 < 3) && ((6 <= 6) || (5 != 6))	
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

<table>
<tr><th>Expression</th><th>True or False</th></tr>
<tr><td>(6 <= 6) && (5 < 3)</td><td>False</td></tr>
<tr><td>(6 <= 6) || (5 < 3)</td><td>True</td></tr>
<tr><td>(5 != 6)</td><td>True</td></tr>
<tr><td>(5 < 3) && (6 <= 6) || (5 != 6)</td><td>True</td></tr>
<tr><td>(5 < 3) && ((6 <= 6) || (5 != 6))</td><td></td></tr>
<tr><td>!((5 < 3) && ((6 <= 6) || (5 != 6)))</td><td></td></tr>
</table>

Expression	True or False
(6 <= 6) && (5 < 3)	False
(6 <= 6) || (5 < 3)	True
(5 != 6)	True
(5 < 3) && (6 <= 6) || (5 != 6)	True
(5 < 3) && ((6 <= 6) || (5 != 6))	False
!((5 < 3) && ((6 <= 6) || (5 != 6)))	

<table>
<tr><th>Expression</th><th>True or False</th></tr>
<tr><td>(6 <= 6) && (5 < 3)</td><td>False</td></tr>
<tr><td>(6 <= 6) || (5 < 3)</td><td>True</td></tr>
<tr><td>(5 != 6)</td><td>True</td></tr>
<tr><td>(5 < 3) && (6 <= 6) || (5 != 6)</td><td>True</td></tr>
<tr><td>(5 < 3) && ((6 <= 6) || (5 != 6))</td><td>False</td></tr>
<tr><td>!((5 < 3) && ((6 <= 6) || (5 != 6)))</td><td>True</td></tr>
</table>

Unary Increment and Decrement Operators

Type	Operators	Usage
Increment and decrement	‘++’ ‘- -’	a++ is the same as a=a+1 a-- is the same as a=a-1

- Prefix
  - ++a;
  - --a;
- Postfix
  - a++;
  - a--;

Acknowledgements

Material in these slides has been taken from, the following resources

1. Deitel and Deitel: C++ How to Program, 7th Edition, Prentice Hall Publications
2. Robert Lafore: Object-Oriented Programming in C++, Fourth Edition, December 2001,Sams Publishing .
3. A Structured Programing Approach Using C++ by Behrouz A. Forouzan
4. www.cplusplus.com