# CS1010E Topic 2b:
# C Basic: Data types and operations

Siau-Cheng KHOO

Block COM2, Room 04-11, +65 6516 6730

www.comp.nus.edu.sg/~khoosc

khoosc@nus.edu.sg

Semester II, 2017/2018

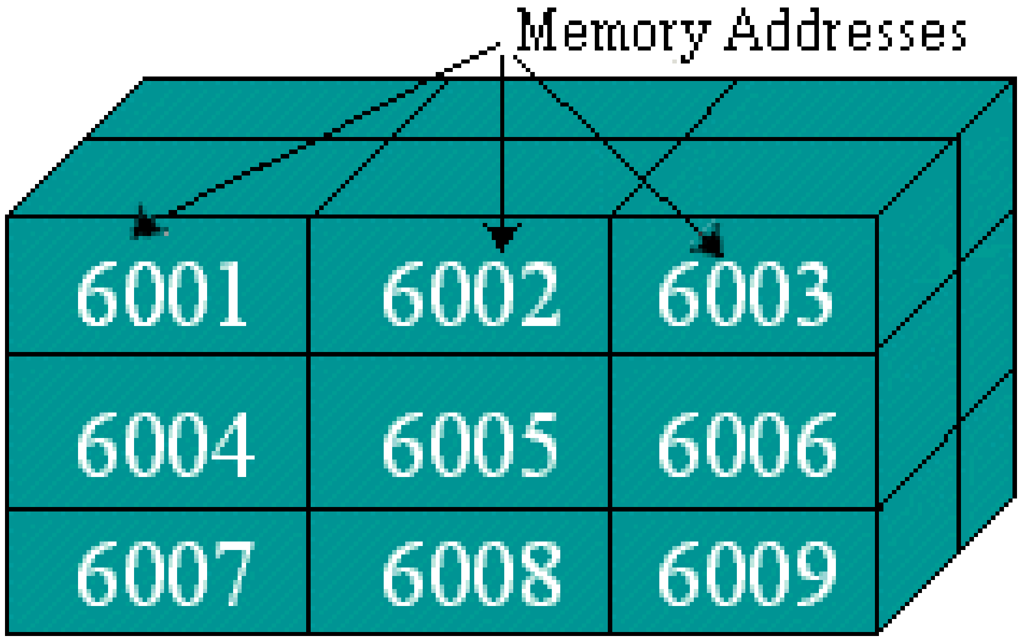
# Lecture Outline

- Data type and values
- Symbolic Constants
- Arithmetic operators and Assignment
- Assignment operators

# Real number representation

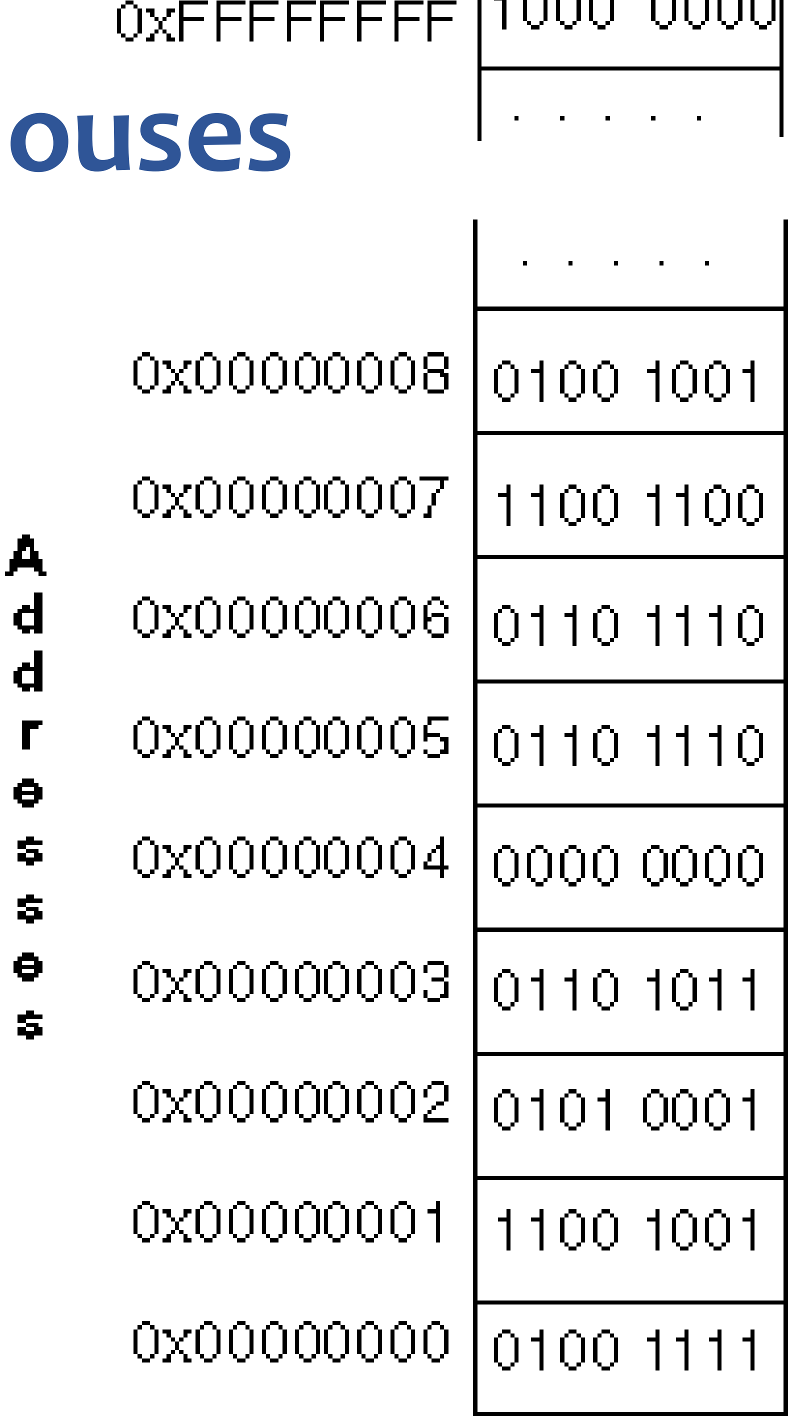
- A real number is usually called a **Floating-point value**
- A floating-point value can represent both integer and real values
    - Eg: 2.5, -0.004, 45.0
- It can also be expressed in **scientific notation**
    - Eg: 2.5 x $10^0$, -4.0 x $10^{-3}$, 4.5 x $10^1$
    - Mantissa: at least 1.0
    - Power : always 10
- Computer expresses this as **expression notation**
    - Eg: 2.5e0, -4.0e-3, 4.5e1

# The bits, the bytes and the Houses



**1 byte = 8 bits**

**1 bit contains a binary 0 or 1**



| | |
|---|---|
| 0xFFFFFFFF | 1000 0000 |
| | . . . . . |
| | . . . . . |
| 0x00000008 | 0100 1001 |
| 0x00000007 | 1100 1100 |
| 0x00000006 | 0110 1110 |
| 0x00000005 | 0110 1110 |
| 0x00000004 | 0000 0000 |
| 0x00000003 | 0110 1011 |
| 0x00000002 | 0101 0001 |
| 0x00000001 | 1100 1001 |
| 0x00000000 | 0100 1111 |

# Simple Data Type – Numeric data

- Integers :

```
short  smallNumber ; // 2 bytes -32,768 . . . 32,767
int    aNumber ;     // 4 bytes usually same as long
long   bigNumber ;   // 4 bytes
                     // -2,147,483,648 … 2,147,483,647
```

- Floating-point values:

```
float          smallReal ;   // 4 bytes    max: 3.402823e+38
double         aReal;        // 8 bytes
long double    bigReal ;     // 16 bytes
```

# Symbolic Constants

- This is defined with a pre-processor directive that assigns an identifier to the constant

```c
#include <stdio.h>

int main(void) {
     double perts ;
     /* take pi = 3.15 */
     perts = 2 * 3.15 * 5  ;
     perts = perts + 2 * 3.15 * 7 ;
     perts = perts + 2 * 3.15 * 10 ;
     . . .
}
```

```c
#include <stdio.h>

int main(void) {
     double perts ;
     /* take pi = 3.14159 */
     perts = 2 * 3.14159 * 5  ;
     perts = perts + 2 * 3.14159 * 7 ;
     perts = perts + 2 * 3.14159 * 10 ;
     . . .
}
```

# Symbolic Constants

- This is defined with a pre-processor directive that assigns an identifier to the constant

```
#include <stdio.h>
#define PI ~~3.15~~   3.14159

int main(void) {
     double perts ;

     perts = 2 * PI * 5  ;
     perts = perts + 2 * PI * 7 ;
     perts = perts + 2 * PI * 10 ;
     . . .
}
```

Conventionally, we write symbolic constants in uppercase.

# Assignment Statements

```
vel = distance_km / time_sec ;
```

- LHS of ‘=’ should be a variable
- RHS of ‘=‘ can be a constant, a variable, or composition of operations
- The statement:
  - assigns a value computed from RHS to an identifier in LHS
  - and more … (discussed later)

# Arithmetic Operators

- +, -, * , / , % are arithmetic operators
  - Eg: `5 % 2` → `1 ;`

```
// integer divide
int a = 9, b = 5 ;
float x ;
x = a / b ;
```

But I really want x to contain the real value 1.8!

- Mixed operators: An operation between values of different types
  - The following all return the same **floating-point values**

```
y = 9.0 / 5  ⬄ y = 9 / 5.0  ⬄ y = 9.0 / 5.0
```

# Cast Operators

- A unary operator that allows us to **specify a type change** in the value before the next computation.

```
int sum = 18 , count = 5 ;
float average ;
average = (float) sum / count ;
```

What are the values assigned to `average` in these two cases?

```
average = sum / (float) count ;

average = (float) (sum / count) ;
```

# Priority of Operators

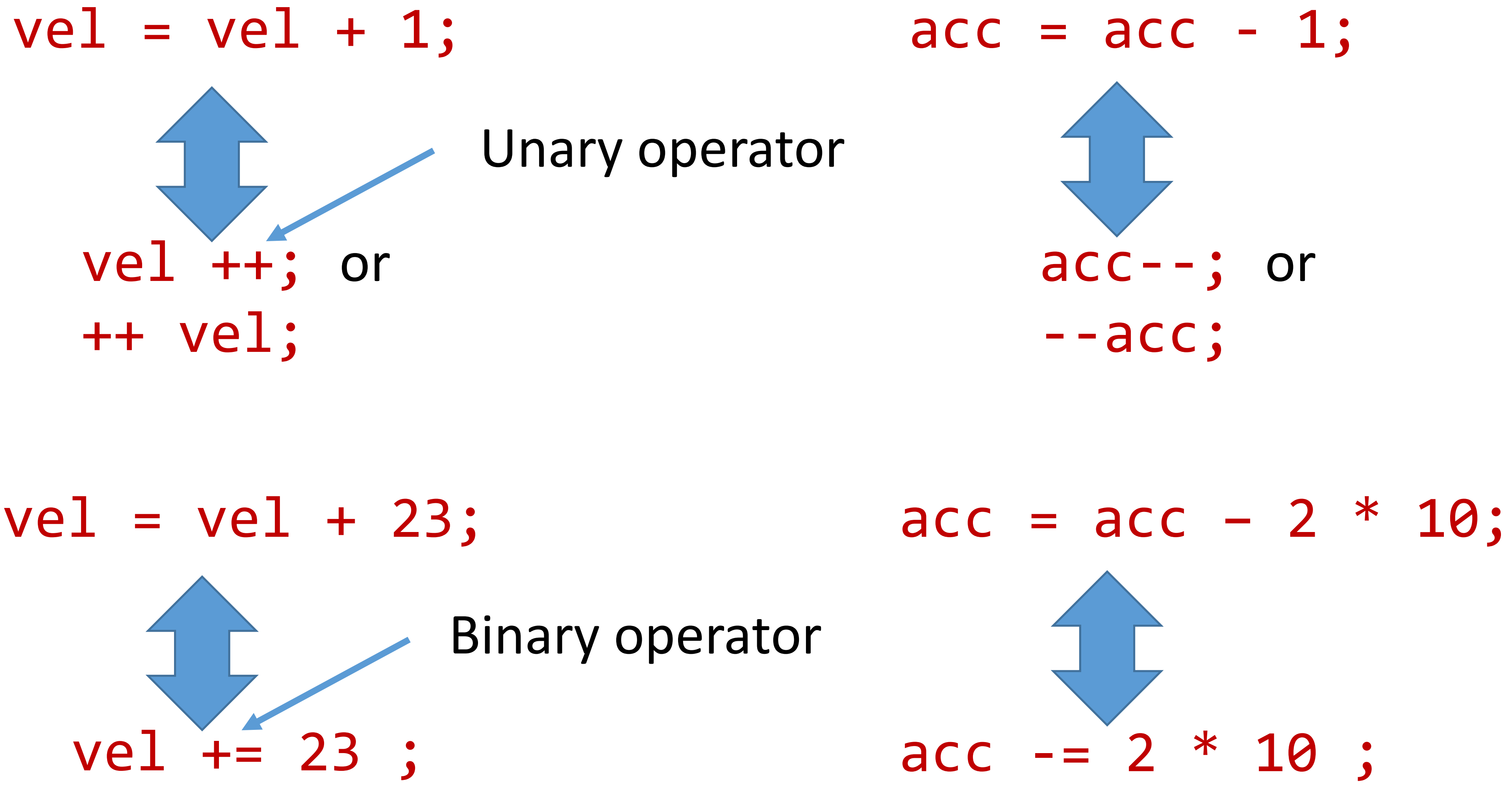
- If RHS contains more than one operator, then we must know the order in which the operators are performed

```
weather = (float) a * b + b / c * d ;
```

- Because multiplication and division have the same precedence level, and because the associativity is from left to right, this RHS is expressed as:

(((float)a) * b) + (( b / c) * d)

# Short forms in writing assignments



```
vel = vel + 1;
```

```
vel ++; or
++ vel;
```

Unary operator

```
acc = acc - 1;
```

```
acc--; or
--acc;
```

```
vel = vel + 23;
```

```
vel += 23 ;
```

Binary operator

```
acc = acc – 2 * 10;
```

```
acc -= 2 * 10 ;
```

# Assignment Operators

- Every operator application returns a value
- The Assignment itself ' = ' is a binary operator.
  - This operation returns the value of its RHS operand
  - Execution of the assignment `vel = 15.0 / 3.0`
    - Assigns `5.0` to LHS variable `vel`
    - Returns `5.0` as its value
- Multiple assignment: `x = y = z = 5 ;`
- The following is also valid, but try not to use it.

```
a = b += c + d ;
```

# Precedence of Arithmetic and Assignment Operators

| Precedence | Operator | Associativity |
|---|---|---|
| 1 | Parenthesis: `(   )` | Innermost first |
| 2 | Unary Operations:<br>`+ , - , ++, -- (type)` | Right to left |
| 3 | Binary operators:<br>`*   /   %` | Left to Right |
| 4 | Binary operators:<br>`+   -` | Left to Right |
| 5 | Assignment operators:<br>`=   +=   -=   *=   /=   %=` | Right to Left |

Etter Sections 2.1 to 2.3

# ++count vs. Count++ (Prefix vs Postfix)

```
w = ++count – y ;
```

count is incremented, then the new value of count is used in evaluating the rest of the expression.

| | count | y | w |
|---|---|---|---|
| Before | 10 | 5 | ? |
| After | 11 | 5 | 6 |

# ++count vs. Count++ (Prefix vs Postfix)

```
w = ++count – y ;
```

| | count | y | w |
|---|---|---|---|
| Before | 10 | 5 | ? |
| After | 11 | 5 | 6 |

```
w = count++ – y ;
```

The old value of `count` is used in evaluating the rest of the expression, and then `count` is incremented.

| | count | y | w |
|---|---|---|---|
| Before | 10 | 5 | ? |
| After | 11 | 5 | 5 |

# Summary

- Data type and values
- Symbolic Constants
- Arithmetic operators and Assignment
- Assignment operators