CS2100 COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

Lecture #2b

Overview of C Programming

Questions?

Ask at https://app.sli.do/event/qVCWNryB45Bnh6p2HRfnFG

OR

Scan and ask your questions here!
(May be obscured in some slides)

5.3 Compute (1/10)

Preprocessor
Input
Compute
Output

- Computation is through function
  - So far, we have used one function: int main(void)
    main() function: where execution of program begins
- A function body has two parts
  - Declarations statements: tell compiler what type of memory cells needed
  - Executable statements: describe the processing on the memory cells

```
int main(void) {
    /* declaration statements */
    /* executable statements */
    return 0;
}
```

Python

```
def main():
  # statements
  return 0
if __name__ == "__main__":
  main()
```

5.3 Compute (2/10)

Preprocessor
Input
Compute
Output

- Declaration Statements: To declare use of variables

```
int count, value;
```

Data type

Names of variables

- User-defined Identifier
  - Name of a variable or function
  - May consist of letters (a-z, A-Z), digits (0-9) and underscores (_), but MUST NOT begin with a digit
  - Case sensitive, i.e. `count` and `Count` are two distinct identifiers
  - Guideline: Usually should begin with lowercase letter
  - Must not be reserved words (next slide)
  - Should avoid standard identifiers (next slide)
  - Eg: *Valid identifiers:*
    `maxEntries, _X123, this_IS_a_long_name`
    Invalid:
    `1Letter, double, return, joe's, ice cream, T*S`

Preprocessor
Input
Compute
Output

5.3 Compute (3/10)

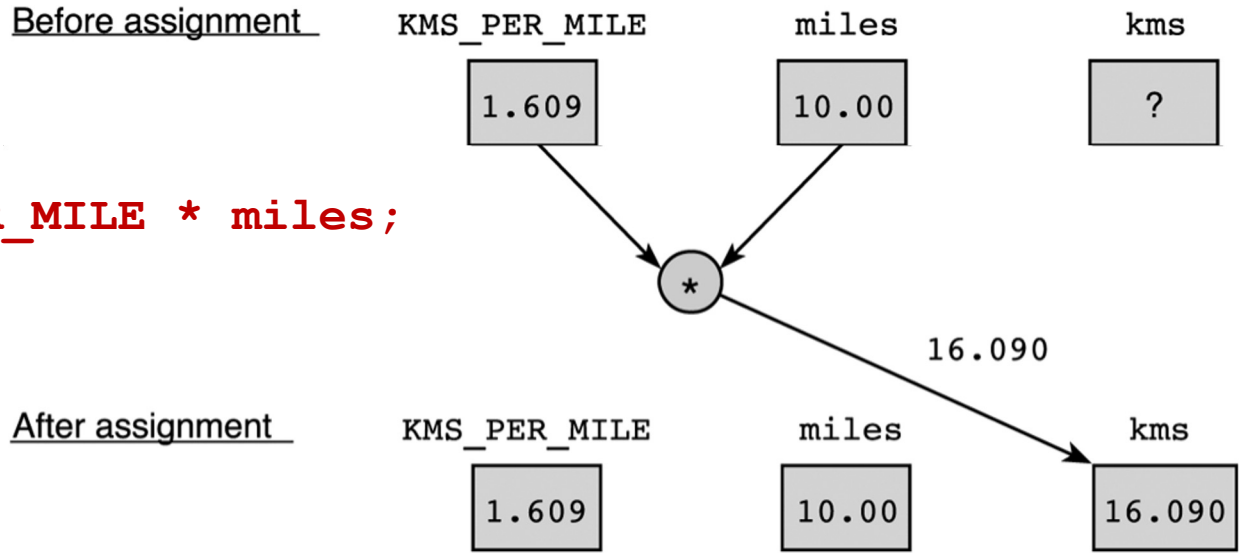
- Reserved words (or keywords)
  - Have special meaning in C
  - Eg: `int`, `void`, `double`, `return`
  - Complete list: http://c.ihypress.ca/reserved.html
  - Cannot be used for user-defined identifiers (names of variables or functions)
- Standard identifiers
  - Names of common functions, such as `printf`, `scanf`
  - Avoid naming your variables/functions with the same name of built-in functions you intend to use

5.3 Compute (4/10)

Preprocessor
Input
Compute
Output

- Executable statements
  - I/O statements (eg: **`printf, scanf`**)
  - Computational and assignment statements
- Assignment statements
  - Store a value or a computational result in a variable
  - (Note: '=' means **'assign value on its right to the variable on its left'**; it does NOT mean equality)
  - Left side of '=' is called lvalue

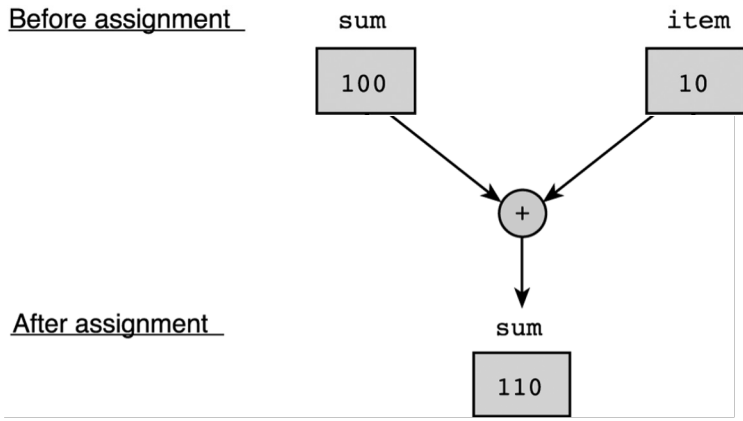
Eg: **`kms = KMS_PER_MILE * miles;`**



5.3 Compute (5/10)

Preprocessor
Input
Compute
Output

Eg: `sum = sum + item;`



- Note: lvalue must be assignable

- Examples of invalid assignment (result in compilation error "lvalue required as left operand of assignment"):
  - `32 = a; // '32' is not a variable`
  - `a + b = c; // 'a + b' is an expression, not variable`
- Assignment can be cascaded, with associativity from right to left:
  - `a = b = c = 3 + 6; // 9 assigned to variables c, b and a`
  - The above is equivalent to: `a = (b = (c = 3 + 6));` which is also equivalent to:

```
c = 3 + 6;
b = c;
a = b;
```

Python

Can write: `a = b = c = 3 + 6`
CANNOT: `a = 5 + (b = 3)`

5.3 Compute (6/10)

Preprocessor
Input
Compute
Output

- Side effect:
  - An assignment statement does not just assigns, it also has the side effect of returning the value of its right-hand side expression
  - Hence `a = 12;` has the side effect of returning the value of 12, besides assigning 12 to `a`
  - Usually we don't make use of its side effect, but sometimes we do, eg:

```
z = a = 12; // or: z = (a = 12);
```

  - The above makes use of the side effect of the assignment statement `a = 12;` (which returns 12) and assigns it to `z`
  - Side effects have their use, but avoid convoluted codes:

```
a = 5 + (b = 10); // assign 10 to b, and 15 to a
```

  - Side effects also apply to expressions involving other operators (eg: logical operators). We will see more of this later.

Preprocessor
Input
Compute
Output

5.3 Compute (7/10)

- Arithmetic operations
  - Binary Operators: +, –, *, /, % (*remainder*)
    - Left Associative (from left to right)
      - `46 / 15 / 2` → `3 / 2` → `1`
      - `19 % 7 % 3` → `5 % 3` → `2`
  - Unary operators: +, –
    - Right Associative
      - `x = - 23` `p = +4 * 10`
  - Execution from left to right, respecting parentheses rule, and then precedence rule, and then associative rule (slide 30)
    - addition, subtraction are lower in precedence than multiplication, division, and remainder
  - Truncate result if result can't be stored (slide 31)
    - `int n;  n = 9 * 0.5;` results in `4` being stored in `n`.

5.3 Compute (8/10)

Preprocessor
Input
Compute
Output

ArithOps.c

```
// To illustrate some arithmetic operations in C
#include <stdio.h>
int main(void) {
   int x, p, n;

   // to show left associativity
   printf("46 / 15 / 2 = %d\n", 46/15/2);
   printf("19 %% 7 %% 3 = %d\n", 19%7%3);

   // to show right associativity
   x = -23;
   p = +4 * 10;
   printf("x = %d\n", x);
   printf("p = %d\n", p);

   // to show truncation of val
   n = 9 * 0.5;
   printf("n = %d\n", n);

   return 0;
}
```

```
$ gcc ArithOps.c -o ArithOps
$ ArithOps
46 / 15 / 2 = 1
19 % 7 % 3 = 2
x = -23
p = 40
n = 4
```

5.3 Compute (9/10)

Preprocessor
Input
Compute
Output

- Arithmetic operators: Associativity & Precedence

Operator Type	Operator	Associativity
Primary expression operators	`( )` *expr*`++` *expr*`--`	Left to right
Unary operators	`*` `&` `+` `-` `++`*expr* `--`*expr* `(`*typecast*`)`	Right to left
Binary operators	`*` `/` `%`	Left to right
	`+` `-`	
Assignment operators	`=` `+=` `-=` `*=` `/=` `%=`	Right to left

Python

`expr++`, `expr--`, `++expr`, `--expr` are not available

5.3 Compute (10/10)

Preprocessor
Input
Compute
Output

- Mixed-Type Arithmetic Operations

```
int   m = 10/4;      means   m = 2;
float p = 10/4;      means   p = 2.0;
int   n = 10/4.0;    means   n = 2;
float q = 10/4.0;    means   q = 2.5;
int   r = -10/4.0;   means   r = -2;
```

Caution!

- Type Casting
  - Use a cast operator to change the type of an expression
    - syntax: (*type*) expression

```
int aa = 6; float ff = 15.8;
float pp = (float) aa / 4;      means  pp = 1.5;
int   nn = (int) ff / aa;       means  nn = 2;
float qq = (float) (aa / 4);    means  qq = 1.0;
```

Try out TypeCast.c

5.3 Compute: Difference with Python

- Python Floor Division

```
a = 10/4      means   a = 2.5
b = 10//4     means   b = 2
c = -10/4     means   c = -2.5
d = -10//4    means   d = -3
```

- Modulo
  - Python % is modulo

```
a = 10%4   → a = 2
b = -10%4  → b = 2
```

  - C % is remainder

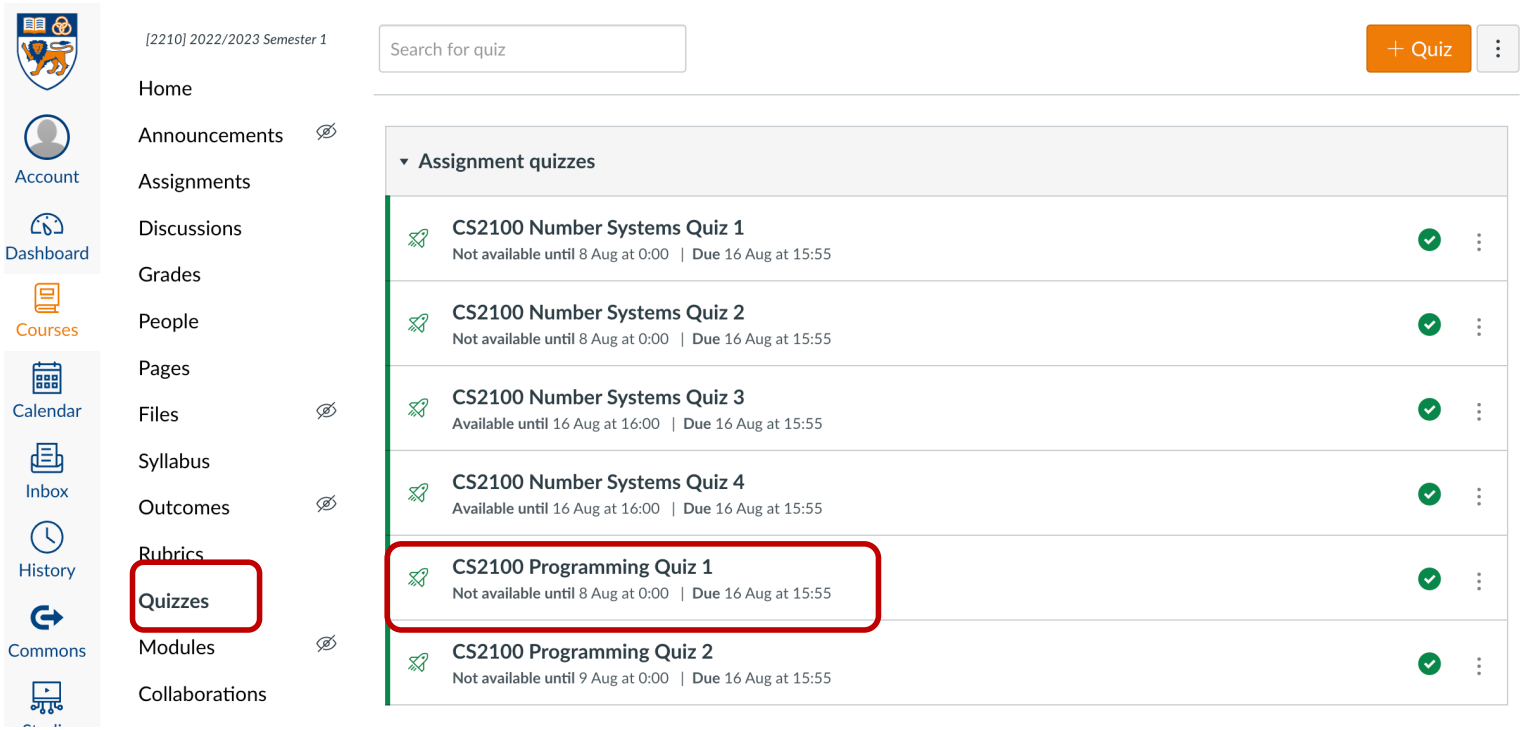
```
a = 10%4   → a = 2
b = -10%4  → b = -2
```

  - NOTE: be careful with negative values for % operation

Try out Modulo.c and compare with Modulo.py

Quiz

- Please complete the “CS2100 C Programming Quiz 1” in Canvas.
  - Access via the “Quizzes” tool in the left toolbar and select the quiz on the right side of the screen.



End of File