CS2100 COMPUTER ORGANISATION

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

Lecture #2a

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)

Lecture #2: Overview of C Programming (1/2)

1. A Simple C Program
2. von Neumann Architecture
3. Variables
4. Data Types
5. Program Structure
   - 5.1 Preprocessor Directives
   - 5.2 Input/Output
   - 5.3 Compute
     - Arithmetic operators
     - Assignment statements
     - Typecast operator

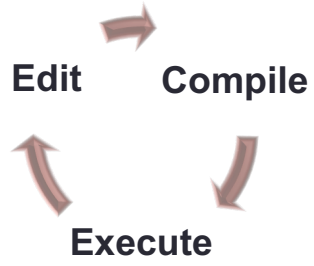
Lecture #2: Overview of C Programming (2/2)

6. Selection Statements
   - 6.1 Condition and Relational Operators
   - 6.2 Truth Values
   - 6.3 Logical Operators
   - 6.4 Evaluation of Boolean Expressions
   - 6.5 Short-Circuit Evaluation
7. Repetition Statements
   - 7.1 Using ‘break’ in a loop
   - 7.2 Using ‘continue’ in a loop

Introduction

- C: A general-purpose computer programming language developed in 1972 by Dennis Ritchie (1941 – 2011) at Bell Telephone Lab for use with the UNIX operation System

Quick Review: Edit, Compile, Execute



Edit
eg: vim first.c

produces

Source code

first.c

Compile
eg: gcc first.c

produces

Executable code

a.out

Cannot compile?

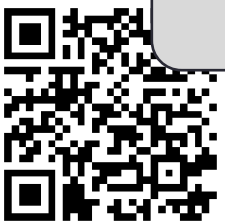
Execute
eg: a.out

produces

Program output

```
The value of c is 3.
```

Incorrect result?



1. A Simple C Program (1/3)

General form

```
preprocessor directives

main function header
{
    declaration of variables
    executable statements
}
```

"Executable statements" usually consists of 3 parts:

- Input data
- Computation
- Output results

1. A Simple C Program (2/3)

MileToKm.c

```c
// Converts distance in miles to kilometres.
#include <stdio.h>   /* printf, scanf definitions */
#define KMS_PER_MILE 1.609 /* conversion constant */

int main(void) {
    float miles,    // input - distance in miles
          kms;      // output - distance in kilometres

    /* Get the distance in miles */
    printf("Enter distance in miles: ");
    scanf("%f", &miles);

    // Convert the distance to kilometres
    kms = KMS_PER_MILE * miles;

    // Display the distance in kilometres
    printf("That equals %9.2f km.\n", kms);

    return 0;
}
```

Sample run

```
$ gcc MileToKm.c
$ a.out
Enter distance in miles: 10.5
That equals     16.89 km.
```

(Note: All C programs in the lectures are available on LumiNUS as well as the CS2100 website. Python versions are also available.)

1. A Simple C Program (3/3)

```
// Converts distance in miles to kilometres.

#include <stdio.h>    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609 /* conversion constant */

int main(void) {
   float miles,    // input - distance in miles
         kms;      // output - distance in kilometres

   /* Get the distance in miles */
   printf("Enter distance in miles: ");
   scanf("%f", &miles);

   // Convert the distance to kilometres
   kms = KMS_PER_MILE * miles;

   // Display the distance in kilometres
   printf("That equals %9.2f km.\n", kms);

   return 0;
}
```

standard header file

preprocessor directives

constant

reserved words

variables

functions

comments

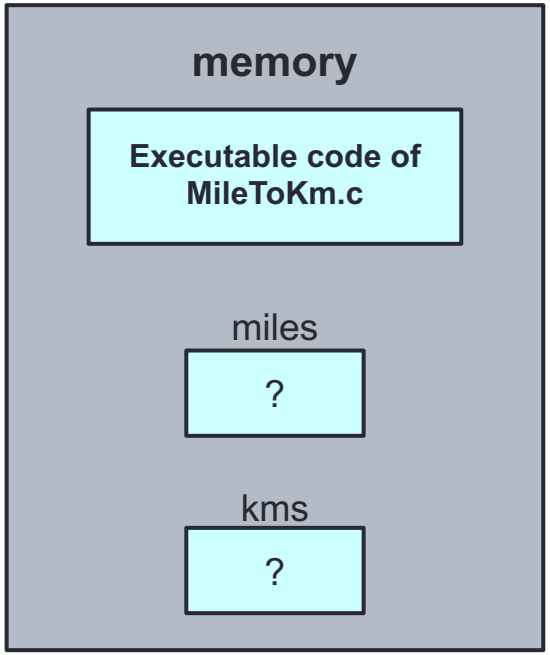
(Only /* ... */ is ANSI C)

special symbols

In C, semi-colon (;) terminates a statement.
Curly bracket { } indicates a block.
In Python: block is by indentation

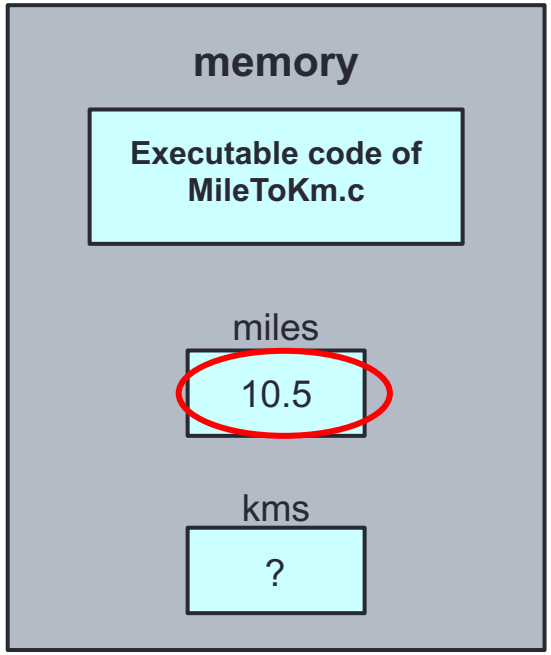
2. von Neumann Architecture (1/2)

What happens in the computer memory?



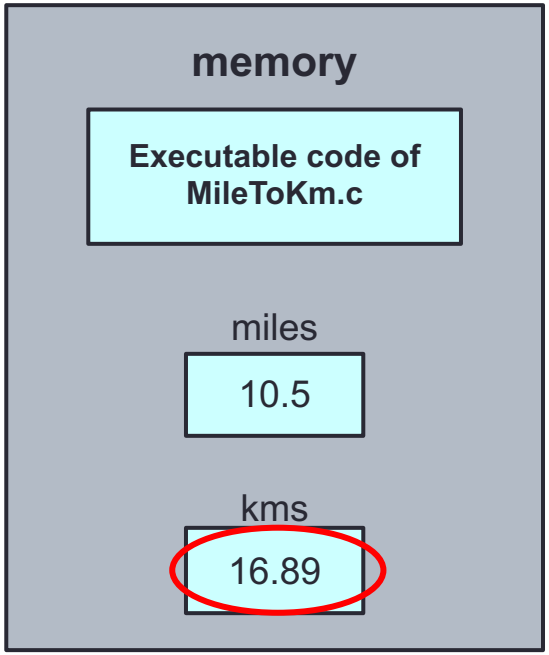
At the beginning

Do not assume that uninitialised variables contain zero! **(Very common mistake.)**



After user enters: 10.5 to

```
scanf("%f", &miles);
```

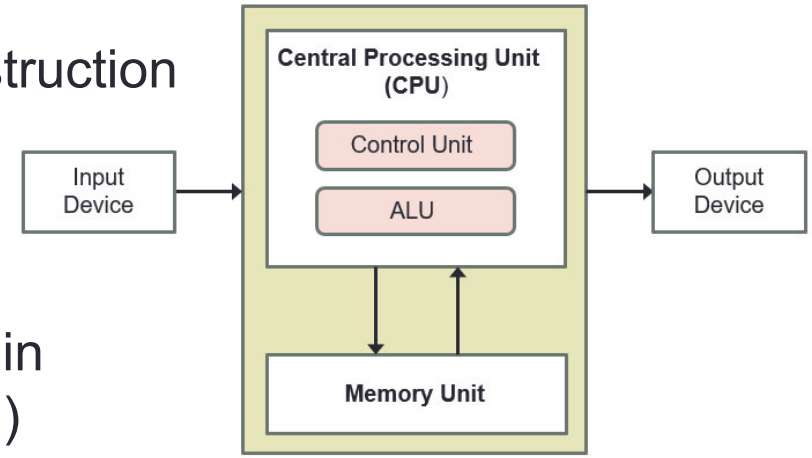


After this line is executed:

```
kms = KMS_PER_MILE * miles;
```

2. von Neumann Architecture (2/2)

- John von Neumann (1903 – 1957)
- von Neumann architecture* describes a computer consisting of:
  - Central Processing Unit (CPU)
    - Registers
    - A control unit containing an instruction register and program counter
    - An arithmetic/logic unit (ALU)
  - Memory
    - Stores both program and data in random-access memory (RAM)
  - I/O devices



(* Also called *Princeton architecture*, or *stored-program architecture*)

3. Variables

```
float miles, kms;
```

- Data used in a program are stored in variables
- Every variable is identified by a name (identifier), has a data type, and contains a value which could be modified
- (Each variable actually has an address too, but for the moment we will skip this until we discuss pointers.)
- A variable is declared with a data type
  - `int count; // variable 'count' of type 'int'`
- Variables may be initialized during declaration:
  - `int count = 3; // 'count' is initialized to 3`
- Without initialization, the variable contains an unknown value (Cannot assume that it is zero!)

Python

Declaration via assignment in function/global

```
count = 3
```

3. Variables: Mistakes in Initialization

- No initialization

  -Wall option turns on all warnings

InitVariable.c

```
int main(void) {
    int count;
    count = count + 12;
    return 0;
}
```

Python

Cannot declare without initialization

```
$ gcc -Wall InitVariable.c
InitVariable.c: In function 'main':
InitVariable.c:3:8: warning: 'count' is used
uninitialized in this function
  count = count + 12;
        ^
```

- Redundant initialization

```
int count = 0;
count = 123;
```

```
int count = 0;
scanf("%d", &count);
```

4. Data Types (1/3)

```
float miles, kms;
```

- Every variable must be declared with a data type
  - To determine the type of data the variable may hold
- Basic data types in C:
  - int: For integers
    - 4 bytes (in sunfire); -2,147,483,648 (-2^{31}) through +2,147,483,647 ($2^{31} - 1$)

Python `int` | JS `number`

  - float or double: For real numbers
    - 4 bytes for float and 8 bytes for double (in sunfire)
    - Eg: 12.34, 0.0056, 213.0
    - May use scientific notation; eg: 1.5e-2 and 15.0E-3 both refer to 0.015; 12e+4 and 1.2E+5 both refer to 120000.0

Python `float` | JS `number`

  - char: For characters
    - Enclosed in a pair of single quotes
    - Eg: `'A'`, `'z'`, `'2'`, `'*'`, `' '`, `'\n'`

Python `str` | JS `string`

4. Data Types (2/3)

- A programming language can be strongly typed or weakly typed
  - Strongly typed: every variable to be declared with a data type. (C: `int count; char grade;`)
  - Weakly typed: the type depends on how the variable is used (JavaScript: `var count; var grade;`)
  - The above is just a simple explanation.
    - Much subtleties and many views and even different definitions. Other aspects include static/dynamic type checking, safe type checking, type conversions, etc.
    - Eg: Java, Pascal and C are strongly typed languages. But Java /Pascal are more strongly typed than C, as C supports implicit type conversions and allows pointer values to be explicitly cast.
    - One fun video: https://www.youtube.com/watch?v=bQdzwJWYZRU

4. Data Types (3/3)

DataTypes.c

```c
// This program checks the memory size
// of each of the basic data types
#include <stdio.h>

int main(void) {
   printf("Size of 'int' (in bytes): %d\n", sizeof(int));
   printf("Size of 'float' (in bytes): %d\n", sizeof(float));
   printf("Size of 'double' (in bytes): %d\n", sizeof(double));
   printf("Size of 'char' (in bytes): %d\n", sizeof(char));

   return 0;
}
```

Python

Use `sys.getsizeof`

```
import sys
sys.getsizeof(1)
```

```
$ gcc DataTypes.c -o DataTypes
$ DataTypes
Size of 'int' (in bytes): 4
Size of 'float' (in bytes): 4
Size of 'double' (in bytes): 8
Size of 'char' (in bytes): 1
```

`-o` option specifies name of executable file (default is 'a.out')

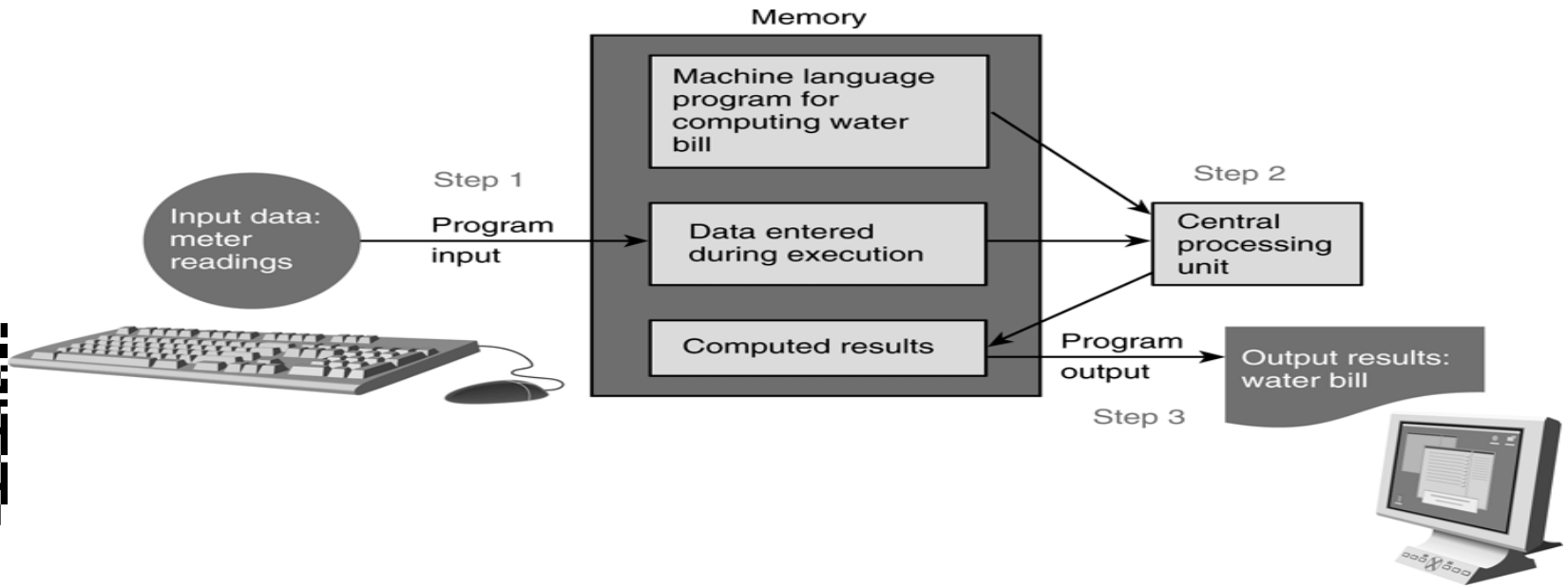
Programming Samples

- All sample programs are available at the Lecture Slides section here:

https://www.comp.nus.edu.sg/~cs2100/2_resources/lectures.html

5. Program Structure

- A basic C program has 4 main parts:
  - Preprocessor directives:
    - eg: #include <stdio.h>, #include <math.h>, #define PI 3.142
  - Input: through stdin (using scanf), or file input
  - Compute: through arithmetic operations and assignment statements
  - Output: through stdout (using printf), or file output



5.1 Preprocessor Directives (1/2)

Preprocessor
Input
Compute
Output

- The C preprocessor provides the following
  - Inclusion of header files
  - Macro expansions
  - Conditional compilation
  - We will focus on inclusion of header files and simple application of macro expansions (defining constants)
- Inclusion of header files
  - To use input/output functions such as scanf() and printf(), you need to include <stdio.h>: **`#include <stdio.h>`**
  - To use functions from certain libraries, you need to include the respective header file, examples:
    - To use mathematical functions, **`#include <math.h>`**
      (In sunfire, need to compile with –lm option)
    - To use string functions, **`#include <string.h>`**

5.1 Preprocessor Directives (2/2)

Preprocessor
Input
Compute
Output

- Macro expansions
  - One of the uses is to define a macro for a constant value
  - Eg: `#define PI 3.142 // use all CAP for macro`

```
#define PI 3.142

int main(void) {
   ...
   areaCircle = PI * radius * radius;
   volCone = PI * radius * radius * height / 3.0;
}
```

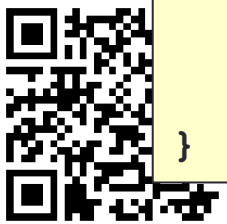
Preprocessor replaces all instances of PI with 3.142 before passing the program to the compiler.

In Python, there is no parallel, but closest is simply declare global variable

```
PI = 3.142
areaCircle = PI * radius * radius
volCone = PI * radius * height / 3.0
```

What the compiler sees:

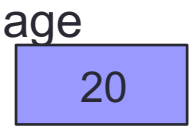
```
int main(void) {
   ...
   areaCircle = 3.142 * radius * radius;
   volCone = 3.142 * radius * radius * height / 3.0;
}
```



5.2 Input/Output (1/3)

Preprocessor
Input
Compute
Output

- Input/output statements:
  - scanf (format string, input list);
  - printf (format string);
  - printf (format string, print list);



Address of variable 'age' varies each time a program is run.

"age" refers to value in the variable age.
"&age" refers to (address of) the memory cell where the value of age is stored.

One version:

```
int age;
double cap; // cumulative averag
printf("What is your age? ");
scanf("%d", &age);
printf("What is your CAP? ");
scanf("%lf", &cap);
printf("You are %d years old, and your CAP is %f\n", age, cap);
```

InputOutput.c

Another version:

```
int age;
double cap; // cumulative average point
printf("What are your age and CAP? ");
scanf("%d %lf", &age, &cap);
printf("You are %d years old, and your CAP is %f\n", age, cap);
```

InputOutputV2.c

5.2 Input/Output (2/3)

Preprocessor
Input
Compute
Output

- %d and %lf are examples of format specifiers; they are placeholders for values to be displayed or read

Placeholder	Variable Type	Function Use
%c	char	printf / scanf
%d	int	printf / scanf
%f	float or double	printf
%f	float	scanf
%lf	double	scanf
%e	float or double	printf (for scientific notation)

Python

All inputs are read as **`string`**

- Examples of format specifiers used in printf():
  - %5d: to display an integer in a width of 5, right justified
  - %8.3f: to display a real number (float or double) in a width of 8, with 3 decimal places, right justified
- Note: For scanf(), just use the format specifier without indicating width, decimal places, etc.

5.2 Input/Output (3/3)

Preprocessor
Input
Compute
Output

- \n is an example of escape sequence
- Escape sequences are used in printf() function for certain special effects or to display certain characters properly
- These are the more commonly used escape sequences:

Escape sequence	Meaning	Result
`\n`	New line	Subsequent output will appear on the next line
`\t`	Horizontal tab	Move to the next tab position on the current line
`\"`	Double quote	Display a double quote "
`%%`	Percent	Display a percent character %

Try out TestIO.c and compare with TestIO.py

End of File