CS1010E Lecture 1

Basics of C Programming with Numerical Computations

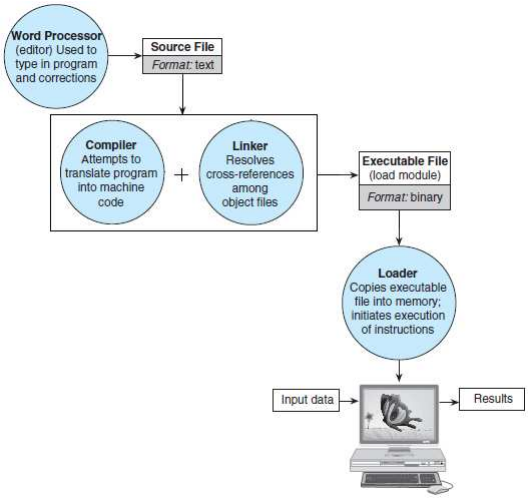
Henry Chia (hchia@comp.nus.edu.sg)

Semester 1 2016 / 2017

Lecture Outline

- Edit-compile-run cycle
- Declaring variables
- Program input/output
- Assignment statement
- Arithmetic with typed-expressions
- Types of errors
- Program style

Edit-Compile-Run Cycle



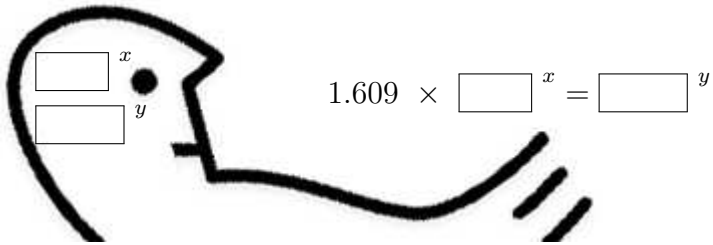
- Edit
  - `vim` editor to create `.c` source file
- Compile (include linking)
  - `gcc` compiler generates `a.exe` or `a.out`
- Run
  - loads the executable
  - `./a.exe` or `./a.out`

Motivating Example

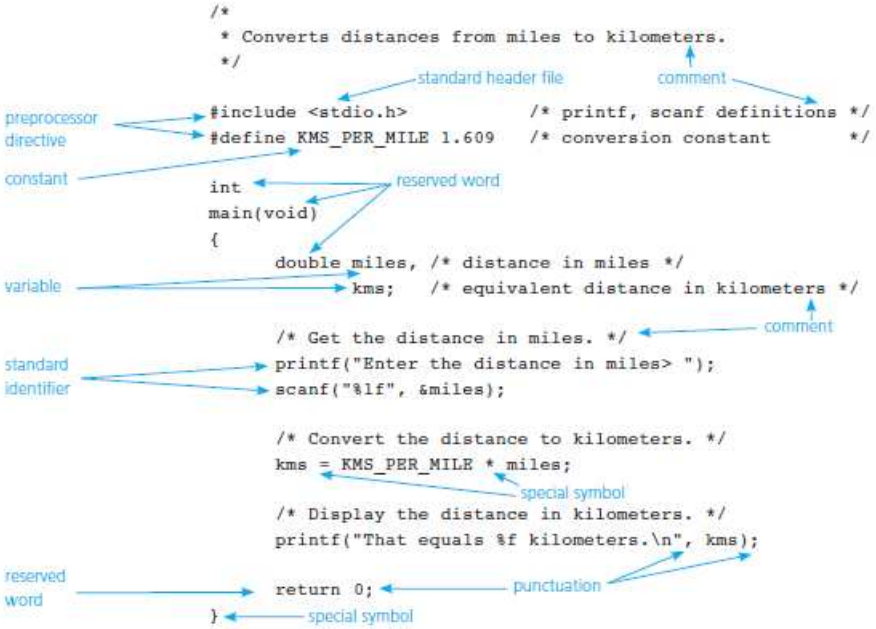
- Given x (miles), convert to y (kms) using

$$y = kx$$

  - x and y and **variables**;
  - k is a **constant** of proportionality ≈ 1.609;
- Given a value for x, what is y?

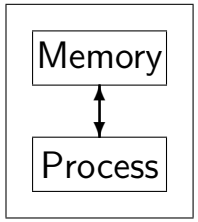


Sample C Program



The `main` Function

```c
/* preprocessor directives */

int main(void) {
   /* declarations (memory) */

   /* statements (process) */

   return 0;
}
```



- Program execution must begin with the **main** function
- The statement

  `return 0;`

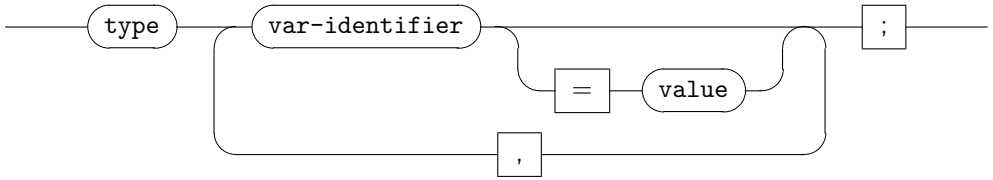
  in the `main` function signifies the successful termination of the program

Declaring Variables – Defining Memory

- **type**
  - integer: `int`
  - floating point (double precision): `double`
- **value**
  - Examples of interger values: `1`, `0`, `-100`
  - Examples of floating point values: `1.0`, `0.123`, `-1.23`
- **identifier** – a meaningful name (case-sensitive)
  - must consist only of letters, digits, and underscores
  - cannot begin with a digit
  - cannot be a reserved word
  - avoid using standard identifier names

Declaring Variables – Syntax

declaration



- Examples:
  - Unknown value: `int x;`
  - Initialized: `int height=3;`
  - Multiple: `double r1, r2=1.23, r3, radius=4.0;`
- It is advisable to always declare and intialize variables to an initial value (typically zero).

Declaring Variables – Example

```c
/* preprocessor directives */

int main(void) {
   double miles, kms; /* distances in miles and kilometers */

   /* statements */

   return 0;
}
```

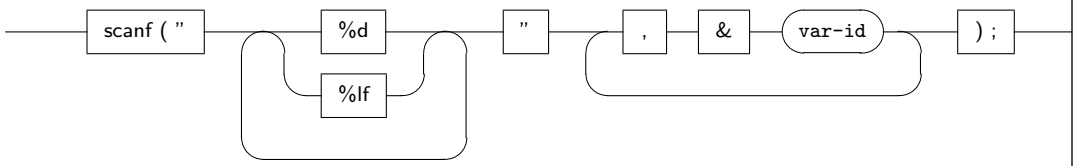
miles	?	kms	?

```c
/* preprocessor directives */

int main(void) {
   double miles=0, kms=0; /* distances in miles and kilometers */

   /* statements */

   return 0;
}
```

miles	0	kms	0

Program Input – `scanf <stdio.h>`

scanfStatement



- Reads(`scanf`) a floating-point value(`%lf`) into(`&`) `miles`

```c
scanf("%lf", &miles);
```

- To read an integer value, use %d
- Requires preprocessor directive: `#include <stdio.h>`
- More examples:

```c
scanf("%lf%lf", &miles, &kms);
scanf("%d%lf", &miles, &kms); /* type-inconsistent? */
```

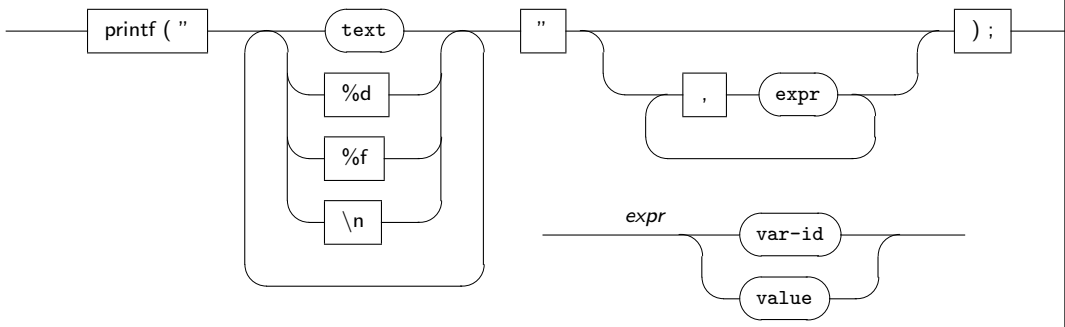

Program Input – Example

```c
#include <stdio.h>

int main(void) {
   double miles=0, kms=0; /* distances in miles and kilometers */

   /* Get the distance in miles */
   scanf("%lf", &miles); /* assume 10.0 is read as input */

   return 0;
}
```

miles	~~0~~ 10.0	kms	0

Program Output – `printf <stdio.h>`

printfStatement



- %d and %f as placeholders to output `int` and `double` values
- Requires preprocessor directive: `#include <stdio.h>`
- Examples:

```c
printf("This is fun :)");
printf("%f miles = %f kms\n", miles, kms);
printf("%f miles = %d kms\n", miles, kms); /* type-inconsistent? */
```

Program Output – Example

```c
#include <stdio.h>

int main(void) {
   double miles=0, kms=0; /* distances in miles and kms */

   /* Get the distance in miles */
   printf("Enter the distance in miles> ");
   scanf("%lf", &miles); /* assume 10.0 is read as input */

   /* Verify the distance entered */
   printf("The distance entered is %f miles\n", miles);

   return 0;
}
```

miles ~~0~~ 10.0 kms 0

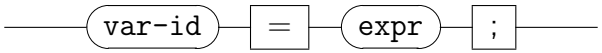
- How about the following?

```c
printf("The distance entered is %d miles\n", miles);
```

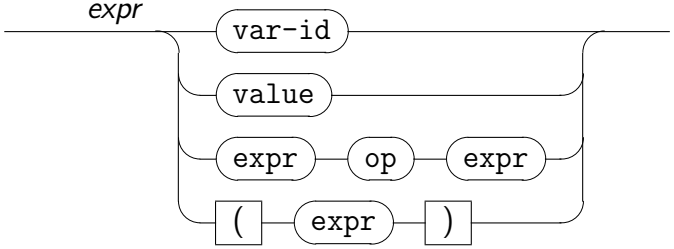

Assignment Statement

- Set variable (`var-id`) to the value of expression `expr`

assignmentStatement



expr



Constant

- A named **constant** can be used in place of a value
- `#define` preprocessor directive for defining constants

```c
#include <stdio.h>

#define KMS_PER_MILE 1.609

int main(void) {
   double miles=0, kms=0; /* distances in miles and kms */

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

   /* Verify the distance entered */
   printf("The distance entered is %f miles\n", miles);
   printf("One mile is equivalent to %f kms\n", KMS_PER_MILE);

   return 0;
}
```

- No variable is declared for the constant

Arithmetic

- Expression involving an operation over two other expressions
- Arithmetic operations: +, -, *, /, % (remainder or modulo)
- Example expression involving operators:

```
    PI    *   radius   *   radius
3.14159         2.0          2.0
      6.28318                    
                   12.56636
```

- Operations over expressions are grouped in order of
  1. Precedence: 2+3*4 → (2+(3*4)) since * before +
  2. Associativity: 2*3*4 → ((2*3)*4) since * is L→R
- Use parentheses ( ) to group explicitly

Typed Expressions

- All expressions are typed. In particular, for $op \in \{+, -, *, /\}$
  - $expr_{\texttt{int}}\ op\ expr_{\texttt{int}} \rightarrow expr_{\texttt{int}}$
  - $expr_i\ op\ expr_j \rightarrow expr_{\texttt{double}}$ if i or j is `double`
- Exercise:
  - `22+7` → ☐
  - `22.0-7.0` → ☐
  - `22.0*7` → ☐
  - `22/7.0` → ☐
  - `22/7` → ☐ (quotient)
  - `22%7` → ☐ (remainder)
- % operates over integers only
- What happens when $v = \frac{4}{3}\pi r^3$ is written as

```
v = 4/3 * 3.142 * r * r * r;
```


Type Conversions

- Numeric conversions:
  - Safe
    - `1.0 *` $expr_{\texttt{int}} \rightarrow expr_{\texttt{double}}$
    - `(double)` $expr_{\texttt{int}} \rightarrow expr_{\texttt{double}}$
  - Unsafe:
    - `(int)` $expr_{\texttt{double}} \rightarrow expr_{\texttt{int}}$
- Examples:
  - `1.0*22/7` → `22.0/7` → `3.142857..`
  - `(double)22/7` → `22.0/7` → `3.142857..`
  - `1.0*(22/7)` → `1.0*3` → `3.0`
  - `(double)(22/7)` → `(double)3` → `3.0`
  - `(int)(22.0/7)` → `(int)3.142857..` → `3`

Typed Assignment

- Expressions are evaluated before assignment. Consider:
  - typed expression evaluation, then
  - possible type conversion during assignment
- Study the following program fragment:

```
int miles=3;
double kms;

miles = miles * 1.609;
kms = miles;
```

What are the values stored in the variables miles and kms?

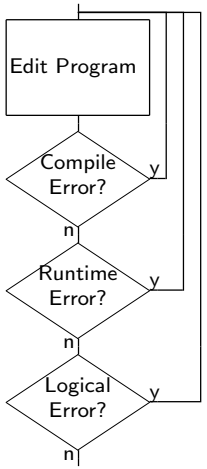
miles$_{\texttt{(int)}}$ [3] kms$_{\texttt{(double)}}$ [?]

Sample Program

```
#include <stdio.h>

#define KMS_PER_MILE 1.609

int main(void) {
   double miles=0, kms=0; /* distances in miles and kms */

   /* Get the distance in miles */
   printf("Enter the distance in miles> ");
   scanf("%lf", &miles); /* assume 10.0 is read as input */

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

   /* Display the distance in kilometers */
   printf("%f miles is equivalent to %f kms\n", miles, kms);

   return 0;
}
```

miles [~~0~~ 10.0] kms [~~0~~ 16.09]

Errors



- Compile error
  - Syntax errors or inconsistencies that are detected by the compiler
- Runtime error
  - Program compiles, starts to execute but terminates prematurely
- Logical error
  - Program compiles, executes and terminates, but with wrong result

Program Style

```c
/*
   This program converts miles to kilometers.
*/
#include <stdio.h>

int main(void) {
   /* statements within a block are indented */
}
```

- Comments:
  - Use block comments: `/* ... */`
  - Use comments, *only when necessary*
  - The header comment is always useful
- Spaces:
  - Blank spaces to improve statement readability
  - Blank lines to separate different sections of code
  - Indentation to define blocks of code `{ ... }`

Program Style

- Compare the following programs. Which is more readable?

```c
/*
   This program converts miles to kilometers.
*/
#include <stdio.h>
#define K 1.609 /* kms per mile */

int main(void) {
   double x=0, y=0; /* x miles; y kms */

   /* Get the distance in miles */
   printf("Enter the distance in miles> ");
   scanf("%lf", &x);

   /* Convert the distance to kilometers */
   y = K * x;

   /* Display the distance in kilometers */
   printf("%f miles = %f kms\n", x, y);

   return 0;
}
```

```c
/*
   This program converts miles to kilometers.
*/
#include <stdio.h>
#define KMS_PER_MILE 1.609
int main(void) {
   double miles=0, kms=0;

   printf("Enter the distance in miles> ");
   scanf("%lf", &miles);

   kms = KMS_PER_MILE * miles;

   printf("%f miles = %f kms\n", miles, kms);

   return 0;
}
```


Lecture Summary

- Importance of **type-awareness** in C programming
- Variables declared with appropriate type according to their usage within the program
- When writing program instructions, keep in mind the type of variables/values and its effect on the instructions
- Maintain **type-consistency** with operations, assignments, input and output
- Handle any instance of type-inconsistency carefully