

# Admin

- 16 Feb = CNY
  - Make-up
- The whole set of slides is in IVLE
  - Some for self-read, will skip some in lecture

**CS1010**

*Programming Methodology*

<http://www.comp.nus.edu.sg/~cs1010/>

## UNIT 4

---

# Overview of C Programming



Leading The World With Asia's Best

# Let's start!



# Unit 3: Overview of C Programming

## Objectives:

- Learn basic C constructs, interactive input, output, and arithmetic operations
- Learn some data types and the use of variables to hold data
- Understand basic programming style

## References:

- Chapter 2 Variables, Arithmetic Expressions and Input/Output
- Chapter 3 Lessons 3.1 Math Library Functions and 3.2 Single Character Data

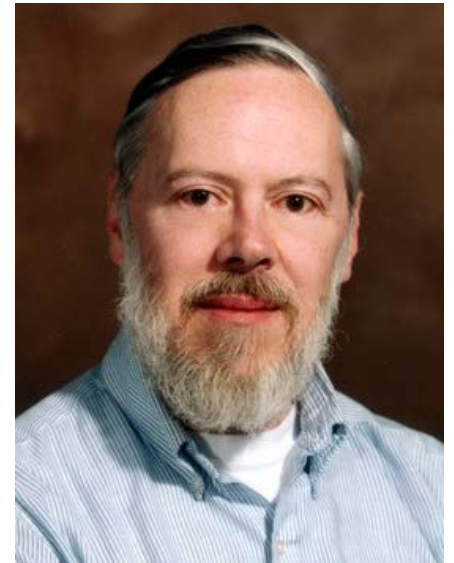
# Unit 3: Overview of C Programming

1. A Simple C Program
2. Variables and Data Types
3. Program Structure
  - Preprocessor directives
  - Input
  - Compute
  - Output
4. Math Functions
5. Programming Style
6. Common Mistakes

# 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
- We will follow the **ANSI C** (C90) standard

[http://en.wikipedia.org/wiki/ANSI\\_C](http://en.wikipedia.org/wiki/ANSI_C)



# British and American English



 apartment  
 flat



 cab  
 taxi



 can  
 tin



 candy  
 sweet



 chips  
 crisps



 closet  
 wardrobe



 cookie  
 biscuit



 corn  
 maize



 diaper  
 nappy



 drapes  
 curtains



 overalls  
 dungarees



 elevator  
 lift



 eraser  
 rubber



 fall  
 autumn



 faucet  
 tap



 flashlight  
 torch



 fries  
 chips





 garbage  
 rubbish



 gasoline  
 petrol



 highway  
 motorway





 hood  
 bonnet



 jello  
 jelly



 license plate  
 number plate

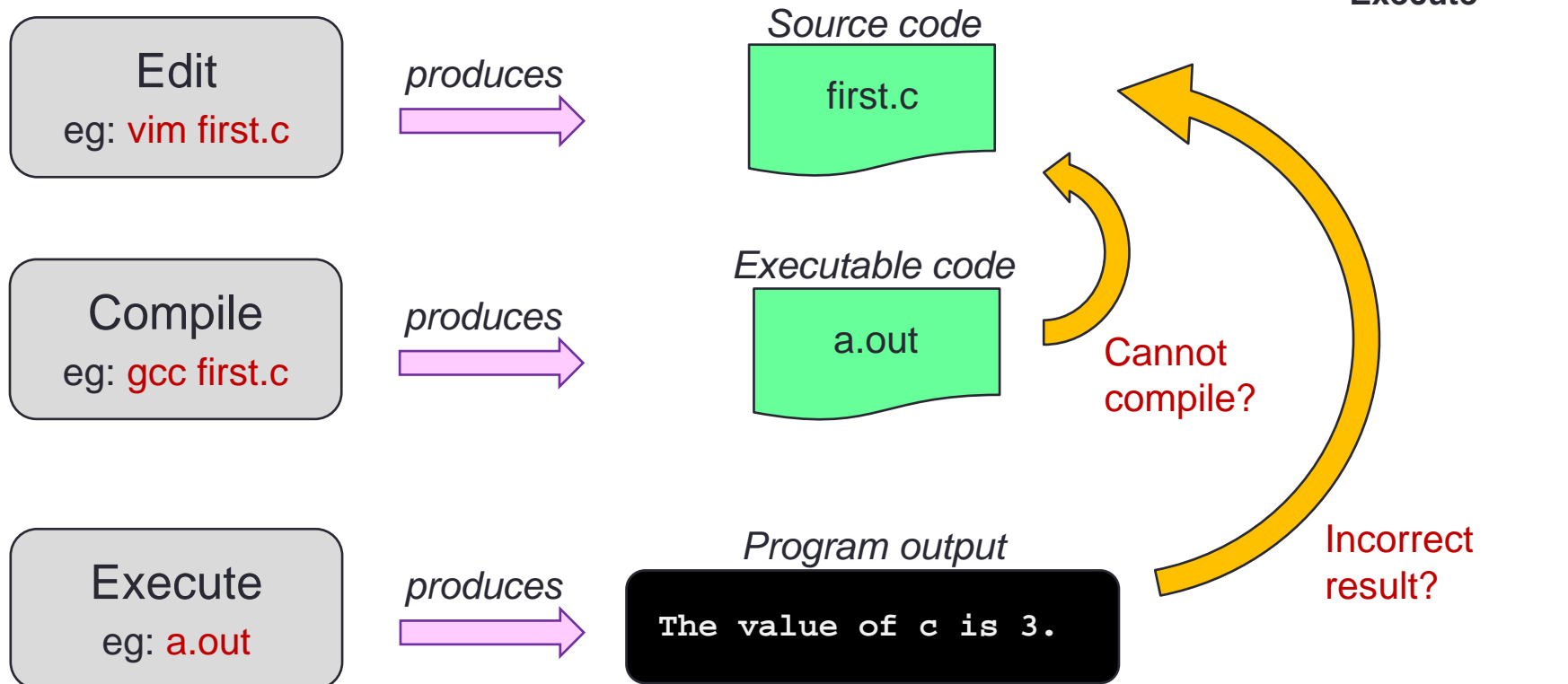


 line  
 queue



# Quick Review: Edit, Compile, Execute

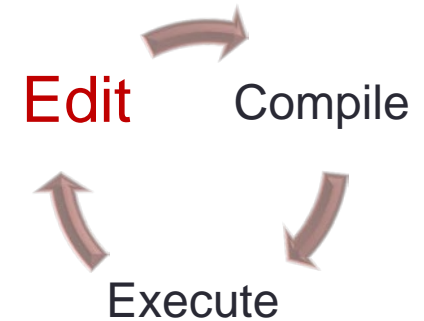
*Test, test, and test!*





# The Simple C Program

- Use vim to create this C program **first.c**



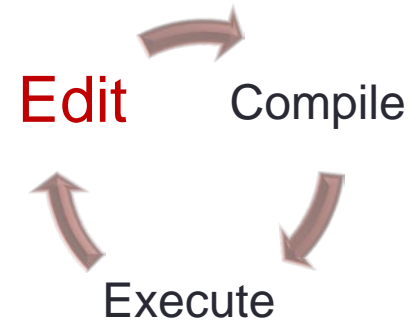
```
#include <stdio.h>

int main(void) {
    printf(" @..@\n");

    return 0;
}
```

# The Less Simple C Program

- Use vim to create this C program `second.c`
  - Hint: copy `first.c` to `second.c` and modify



```
#include <stdio.h>
```

```
int main(void) {
```

```
int a,b,c;
```

```
a = 2001;
```

```
b = 4002;
```

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

```
printf(" The value of %d + %d = %d\n",a,b,c);
```

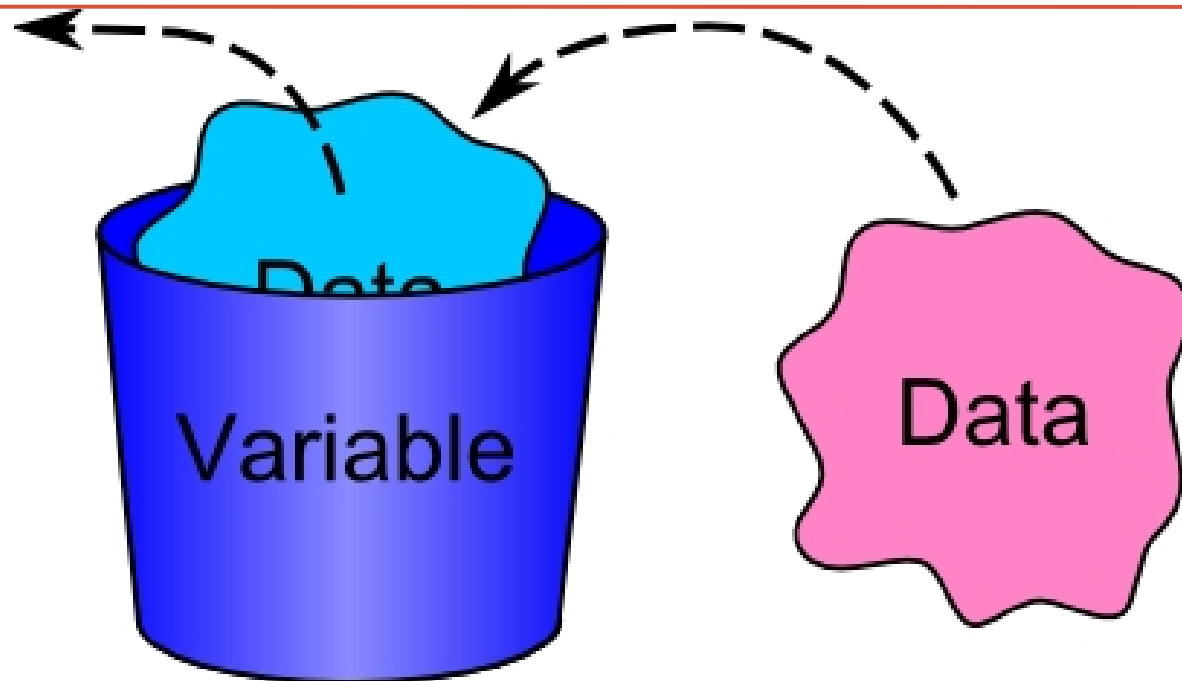
```
return 0;
```

```
}
```

Variables

# VARIABLES

---



# A Simple C Program (2/3)

Unit3\_MileToKm.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;

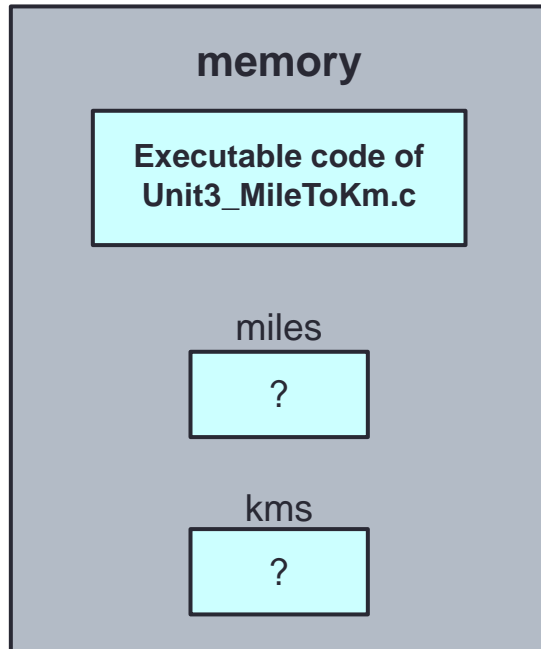
    // Display the distance in kilometres
    printf("That equals %9.2f km.\n", kms);

    return 0;
}
```

## Sample run

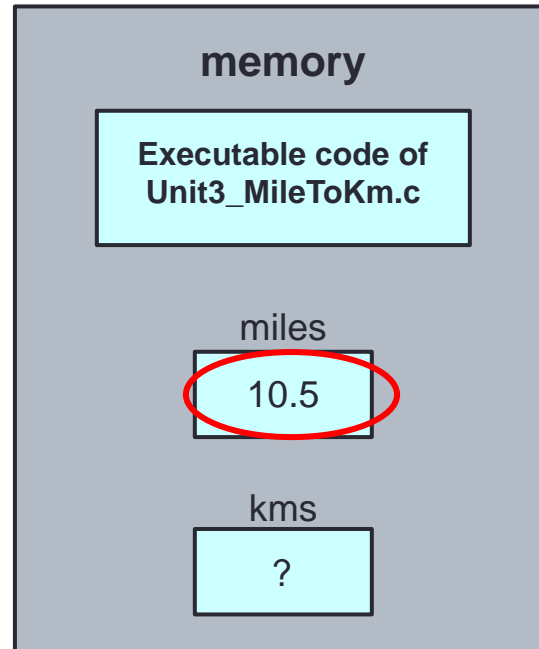
```
$ gcc -Wall Week2_MileToKm.c
$ a.out
Enter distance in miles: 10.5
That equals      16.89 km.
```

# What Happens in the Computer Memory



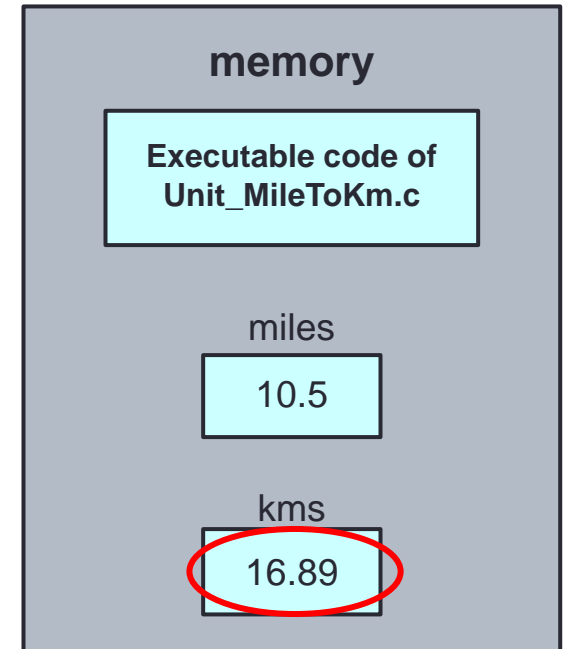
At the beginning

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:

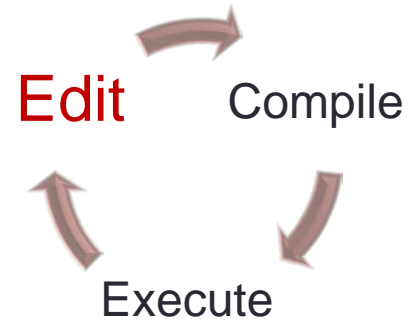
```
kms = KMS_PER_MILE * miles;
```

# Variables

- 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
- A variable is declared with a data type
  - Eg: `int count; // variable 'count' of type 'int'`
- Variables may be initialized during declaration:
  - Eg: `int count = 3; // count is initialized to 3`
- Without initialization, the variable contains an unknown value (Cannot assume that it is zero)

# The Less Simple C Program

- Use vim to create this C program `second.c`

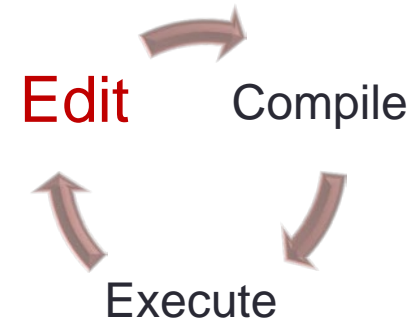


```
#include <stdio.h>

int main(void) {
    int a,b,c;
    a = 2001;
    b = 4002;
    c = a + b;
    printf(" The value of %d + %d = %d\n",a,b,c);
    return 0;
}
```

# The Less Simple C Program

- Use vim to create this C program `second.c`



```
#include <stdio.h>
```

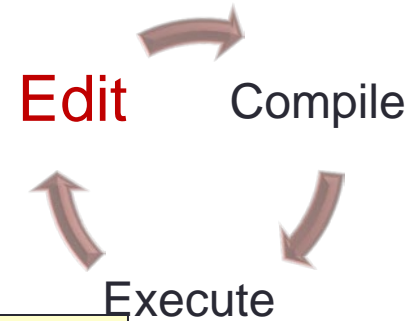
```
int main(void) {  
    int a=2001,b=4002,c;
```

What is the value of "c" before the next line?

```
    c = a + b;  
    printf(" The value of %d + %d = %d\n",a,b,c);  
    return 0;  
}
```



# What if

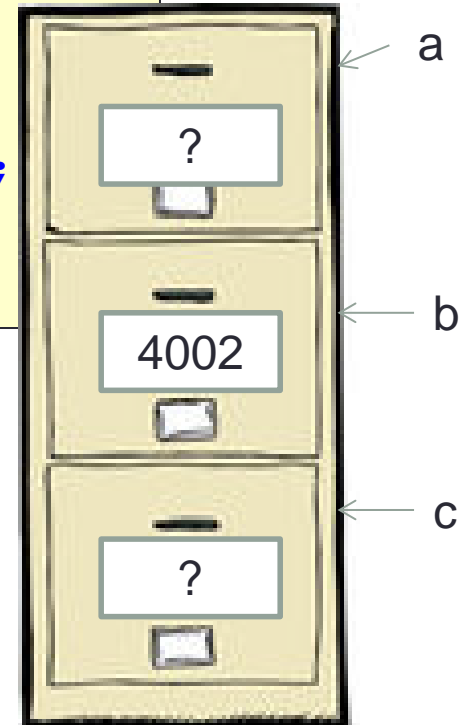


```
#include <stdio.h>

int main(void) {
    int a,b,c;

    b = 4002;
    c = a + b;
    printf(" The value of %d + %d = %d\n",a,b,c);
    return 0;
}
```

*Read the data in a and b, add them and put the sum in c*



# Variables: Mistakes in Initialization

## ■ Incorrect: No initialization

```
int count;  
count = count + 12; ←
```

Does 'count' contain 12 after this statement?

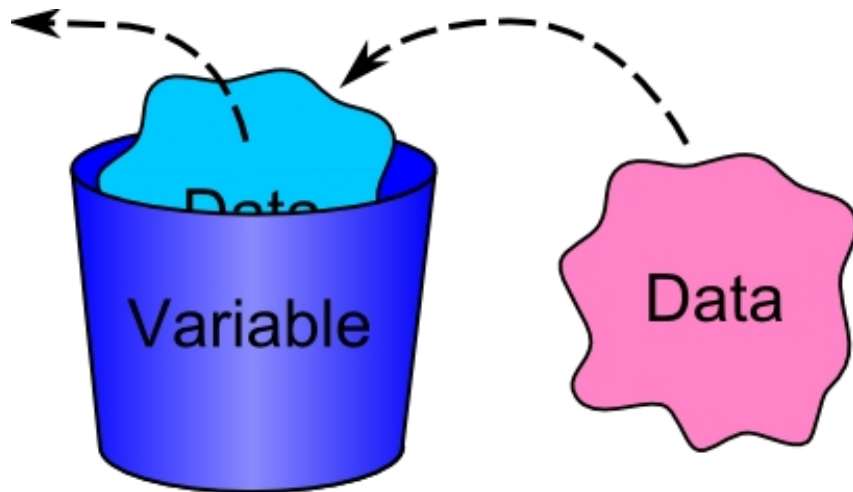
## ■ Redundant initialization

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

Initialization here is redundant.

# Personality Type

- What attitude does you usually contain?
- Data type
  - What type of data can the variable contain?



Academic      Affectionate      Calm

Confident      Cool      Easygoing

✿ WHAT IS YOUR PERSONALITY TYPE? ✿

[www.theenglishstudent.com](http://www.theenglishstudent.com)

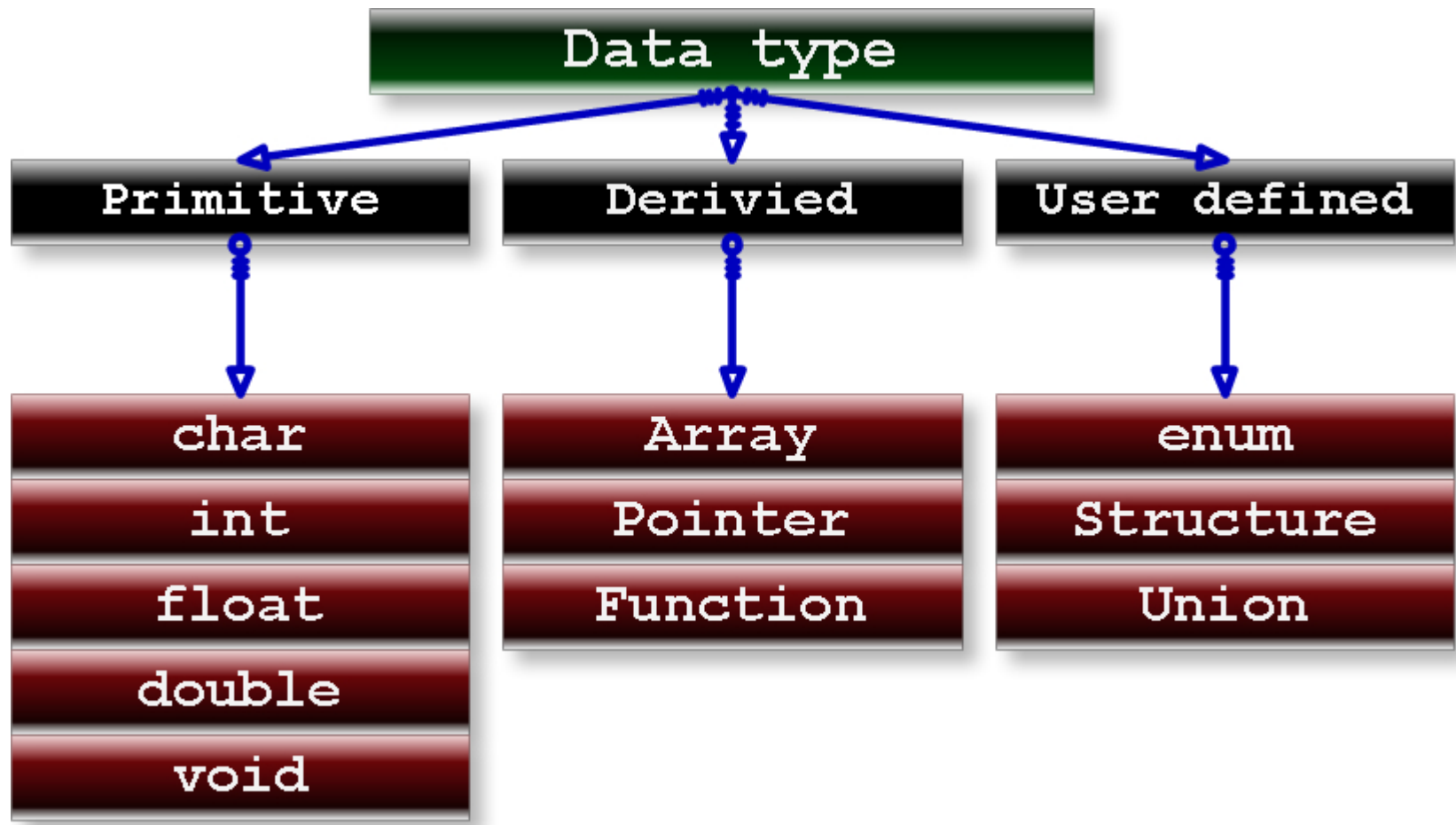
Enthusiastic      Emotional      Formal

Funny      Gentle      Strict

# Data Types

- To determine the type of data a variable may hold
- Basic data types in C (more will be discussed in class later):
  - **int**: For integers
    - 4 bytes (in sunfire); -2,147,483,648 ( $-2^{31}$ ) through +2,147,483,647 ( $2^{31} - 1$ )
  - **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
    - **Not exact!!!!!!!!!!!!**
  - **char**: For individual characters
    - Enclosed in a pair of single quotes
    - Eg: 'A', 'z', '2', '\*', ' ', '\n'

# All Data Types in C



# Exercise #1: Size of Data Types

- We will do an exercise in class to explore the aforementioned information about data types
  - `Unit3_DataTypes.c`
  - Copy the above program into your current directory  

```
cp ~cs1010/lect/prog/unit3/Unit3_DataTypes.c .
```

Pathname of source file

Destination directory;  
'.' means current directory
  - Or download program from CS1010 Lectures page and transfer it into your UNIX account:  
[http://www.comp.nus.edu.sg/~cs1010/2\\_resources/lectures.html](http://www.comp.nus.edu.sg/~cs1010/2_resources/lectures.html)

# Notes (1/2)



- Basic steps of a simple program
  1. Read inputs (scanf)
  2. Compute
  3. Print outputs (printf)
- For now we will use interactive inputs
  - Standard input stream (stdin) – default is keyboard
  - Use the **scanf()** function
- Assume input data always follow specification
  - Hence no need to validate input data (for now)
- Outputs
  - Standard output stream (stdout) – default is monitor
  - Use the **printf()** function

## Notes (2/2)



- Include header file `<stdio.h>` to use `scanf()` and `printf()`
  - Include the header file (for portability sake) even though some systems do not require this to be done
- Important! (CodeCrunch issue)
  - Make sure you have a newline character (`'\n'`) at the end of your last line of output, or CodeCrunch may mark your output as incorrect.

```
printf("That equals %9.2f km.\n", kms);
```



# My own suggestion

- Create a “blank” c template

```
#include <stdio.h>

int main(void) {

    return 0;
}
```

# A Simple C Program (1/3)

- General form of a simple C program

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

“Executable statements”  
usually consists of 3 parts:

- Input data
- Computation
- Output results

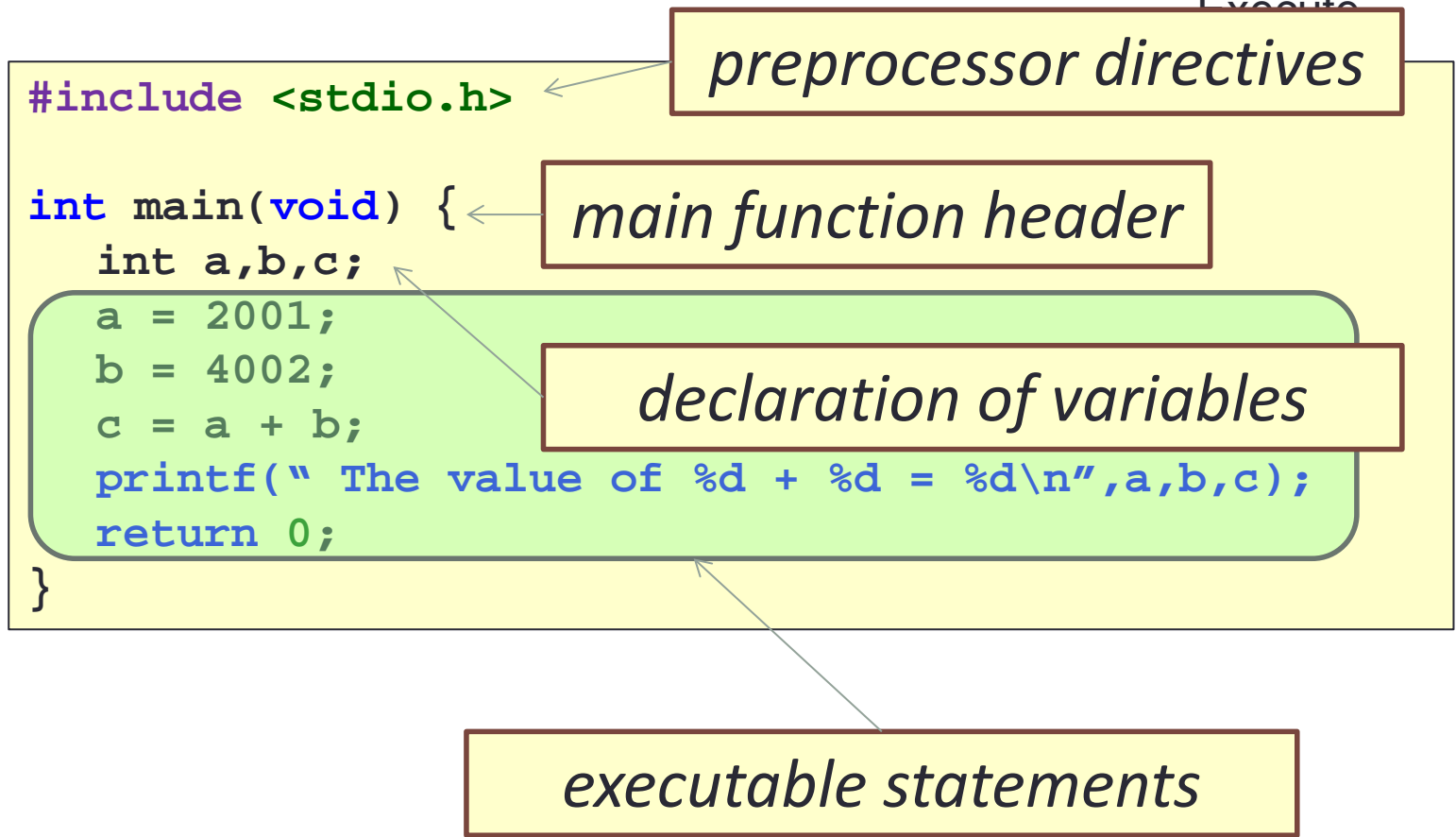
# The Less Simple C Program

- Use vim to create this C program `first.c`

Edit

Compile

Execute



# A Simple C Program (2/3)

Unit3\_MileToKm.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 -Wall Week2_MileToKm.c
$ a.out
Enter distance in miles: 10.5
That equals      16.89 km.
```

# 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;
}
```

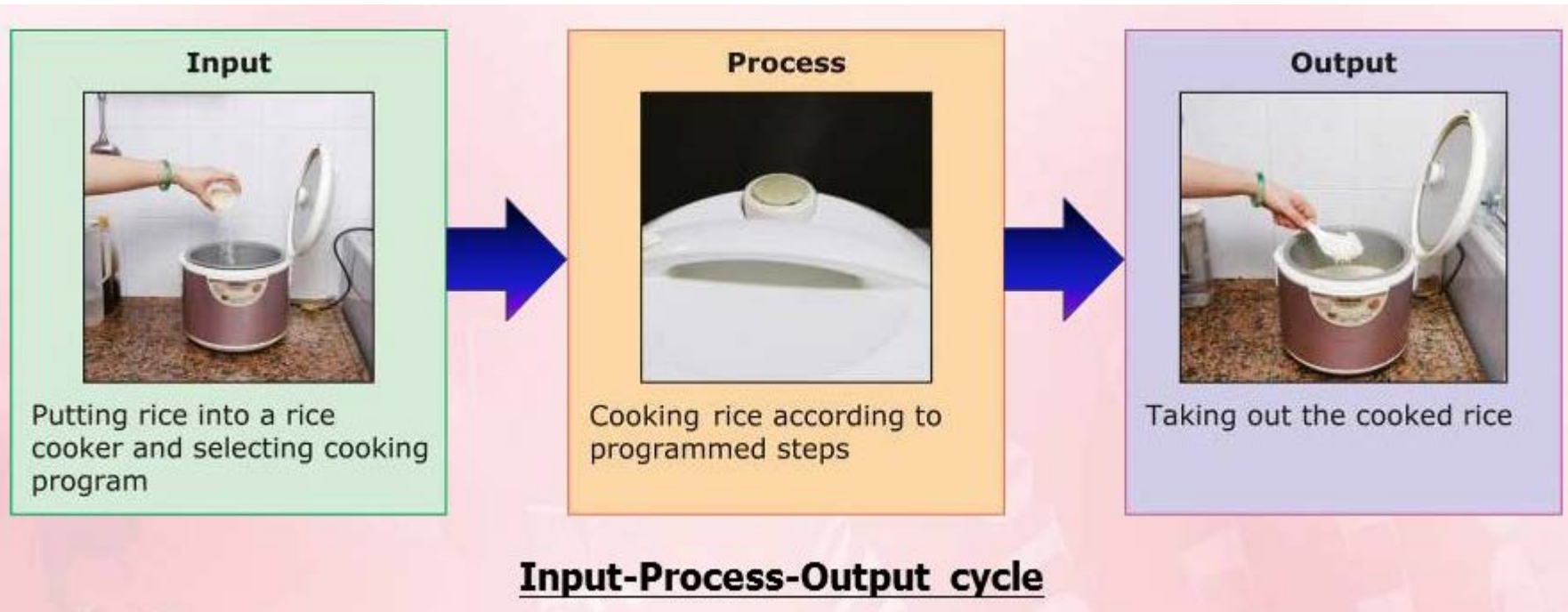
The diagram illustrates the syntax of a C program with the following annotations:

- preprocessor directives**: Points to `#include <stdio.h>` and `#define KMS_PER_MILE 1.609`.
- standard header file**: Points to `<stdio.h>`.
- constant**: Points to `1.609`.
- reserved words**: Points to `int`, `float`, and `main`.
- variables**: Points to `miles` and `kms`.
- functions**: Points to `printf` and `scanf`.
- comments**: Points to `/* Get the distance in miles */`, `/* printf, scanf definitions */`, `/* conversion constant */`, `/* input - distance in miles`, and `/* output - distance in kilometres`.
- special symbols**: Points to `<`, `>`, `*`, `.`, `f`, `\n`, and `;`.
- punctuations**: Points to `{`, `}`, `,`, `;`, and `;`.

# Program Structure



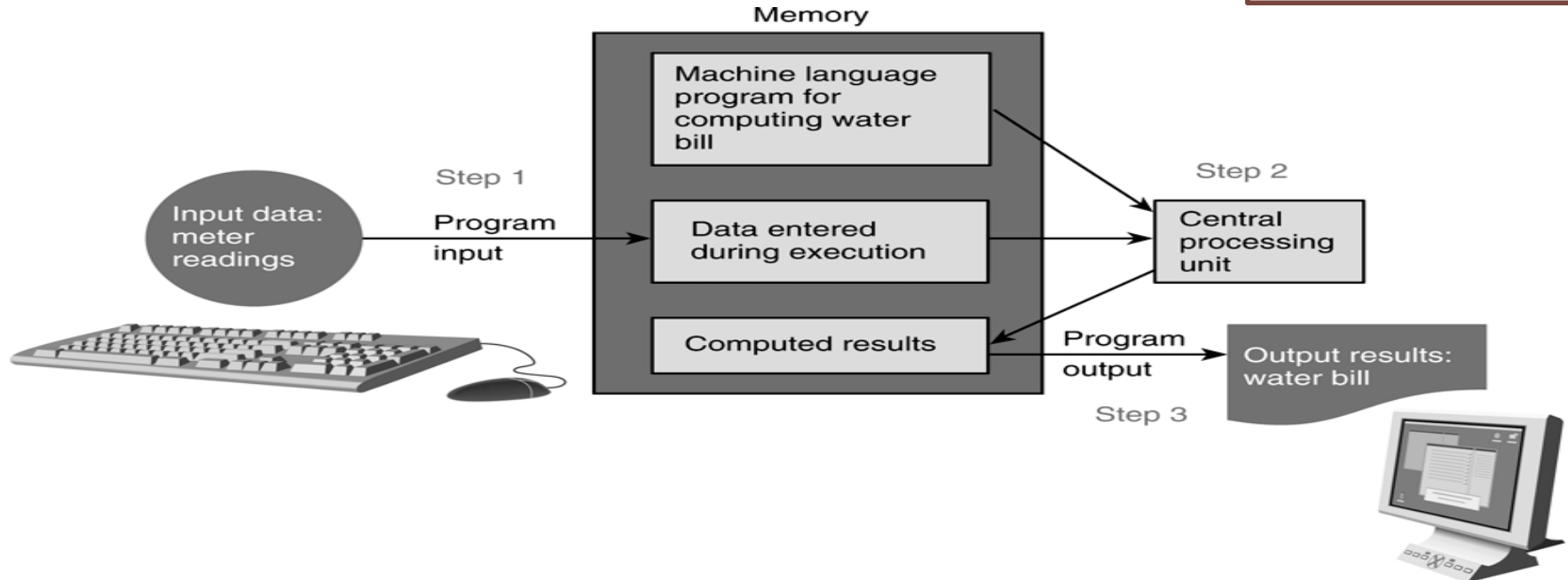
# Basic Structure of a Program



# 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
  - **Output:** through stdout (using `printf`), or file output

We will learn file input/output later.





# A Simple C Program (2/3)

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

Preprocessor  
directives

```
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);
```

Input

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

Compute

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

Output

```
return 0;
```

```
}
```

# Program Structure: Preprocessor Directives (1/2)

- The C preprocessor provides the following
  - Inclusion of header files
  - Macro expansions
  - Conditional compilation
  - For now, we will focus on inclusion of header files and simple application of macro expansions
- 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 mathematical functions, you need to include `<math.h>`: **`#include <math.h>`**

Preprocessor  
Input  
Compute  
Output

# Preprocessor Directives

Unit3\_MileToKm.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;
}
```

# stdio.h

```

/*
 * Copyright (c) 1990 The Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley. The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 *      @(#)stdio.h      5.3 (Berkeley) 3/15/86
 */

/*
 * NB: to fit things in six character monospace externals, the
 * stdio code uses the prefix `__s' for stdio objects, typically
 * followed by a three-character attempt at a mnemonic.
 */

#ifndef _STDIO_H_
#ifdef __cplusplus
extern "C" {
#endif
#define _STDIO_H_

#define _RSTDIO      /* `function_stdio.h' */

```

# Preprocessor Directives

Unit3\_MileToKm.c

```
// Converts distance in miles to kilometres.
#include <stdio.h> /* printf, scanf definitions */
```

The whole "stdio.h" inserted here

```
#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;

```

```
}
```

```
/*
 * Copyright (c) 1990 The Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley. The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */
@(#)stdio.h 5.3 (Berkeley) 3/15/86
/*
 * NB: to fit things in six character monospace externals, the
 * stdio code uses the prefix `_' for stdio objects, typically
 * followed by a three-character attempt at a mnemonic.
 */
#ifdef _STDIO_H
#ifdef __cplusplus
extern "C" {
#endif
#define _STDIO_H_
#define _PCMDIO_ /* \function_stdio! */
```

# Program Structure: Preprocessor Directives (2/2)

## ■ 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

Preprocessor  
Input  
Compute  
Output

```
#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.

What the compiler sees:

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

# Preprocessor Directives

Unit3\_MileToKm.c

```
// Converts distance in miles to kilometres.
#include <stdio.h> /* printf, scanf definitions */
```

The whole "stdio.h" inserted here

```
#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 = 1.609 * miles;

```

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

```

```
    return 0;

```

```
}
```

```
/*
 * Copyright (c) 1990 The Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley. The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */
@(#)stdio.h 5.3 (Berkeley) 3/15/86
/*
 * NB: to fit things in six character monospace externals, the
 * stdio code uses the prefix `_' for stdio objects, typically
 * followed by a three-character attempt at a mnemonic.
 */
#ifndef _STDIO_H
#ifdef __cplusplus
extern "C" {
#endif
#define _STDIO_H_
#define _PCMDIO /* \function_stdio! */
```

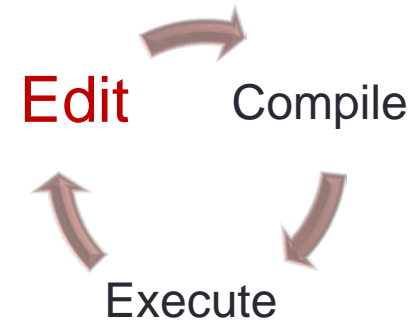
# OUTPUT

---



# Our “second.c”

- Use vim to create this C program `second.c`



```
#include <stdio.h>

int main(void) {
    int a,b,c;
    a = 2001;
    b = 4002;
    c = a + b;
    printf(" The value of %d + %d = %d\n", a,b,c);
    return 0;
}
```



# Program Structure: Input/Output (1/3)

- **%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)

- 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
- See [Table 2.3 \(page 65\)](#) for sample displays
- **Note**: For **scanf()**, just use the format specifier without indicating width, decimal places, etc.

# How to ...

- Write a program and output this:

```
hcheng@suna0:~/c[1031]$ a.out
I am going to score $100 in my class "CS1010"
hcheng@suna0:~/c[1032]$
```

---

## Program Structure: Input/Output (2/3)

- `\n` is an example of **escape sequence**
- Escape sequences are used in `printf()` function for certain special effects or to display certain characters properly
- See [Table 1.4 \(pages 32 – 33\)](#)
- These are the more commonly used escape sequences:

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

Note the error in Table 1.4. It should be `%%` and not `\%`

# INPUT

---

# Input

- What if I want to input different values of `a` and `b` by keyboard?

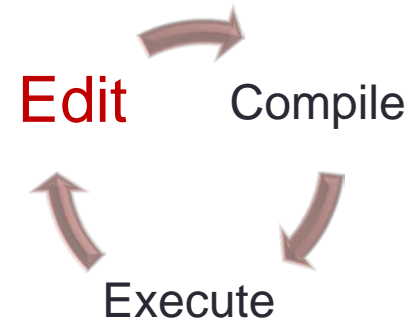


```
#include <stdio.h>

int main(void) {
    int a,b,c;
a = 2001;
b = 4002;
    c = a + b;
    printf(" The value of %d + %d = %d\n",a,b,c);
    return 0;
}
```

# Input

- What if I want to input different values of `a` and `b` by keyboard?



```
#include <stdio.h>

int main(void) {
    int a,b,c;
    printf("Input the first number a: ");
    scanf("%d", &a);
    printf("Input the second number b: ");
    scanf("%d", &b);
    c = a + b;
    printf(" The value of %d + %d = %d\n",a,b,c);
    return 0;
}
```

# Program Structure: Input/Output (3/3)

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

age

20

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

One version:

```
int age;
double cap; // cumulative average
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);
```

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

Unit3\_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);
```

Unit3\_InputOutputV2.c



# Input

- What if I want to input different values of `a` and `b` by keyboard?

```
#include <stdio.h>

int main(void) {
    int a,b,c;
    printf("Input the first number a: ");
    scanf("%d", &a);
    printf("Input the second number b: ");
    scanf("%d", &b);
    c = a + b;
    printf(" The value of %d + %d = %d\n",a,b,c);
    return 0;
}
```

**MOST  
IMPORANTE FOR  
INPUT!!!!!!!**

**No need for output**

## Exercise #2: Testing scanf() and printf()

- We will do an exercise in class to explore `scanf()` and `printf()` functions
  - `Unit3_TestIO.c`
  - Copy the above program into your current directory  
`cp ~cs1010/lect/prog/unit3/Unit3_TestIO.c .`
  - Or download program from CS1010 Lectures page and transfer it into your UNIX account:  
[http://www.comp.nus.edu.sg/~cs1010/2\\_resources/lectures.html](http://www.comp.nus.edu.sg/~cs1010/2_resources/lectures.html)

## Exercise #3: Distance Conversion (1/2)

- Convert distance from miles to kilometres
  - `Unit3_MileToKm.c`
  - The program is given (which you can copy to your directory as earlier instructed), but for this exercise we want you to type in the program yourself as a practice in using `vim`
  - The program is shown in the next slide

# Exercise #3: Distance Conversion (2/2)

Unit3\_MileToKm.c

```
// Unit3_MileToKm.c
// Converts distance in miles to kilometers.
#include <stdio.h>
#define KMS_PER_MILE 1.609

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

    /* 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;
}
```

# Computation



# Program Structure: Compute (1/9)

- 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;  
}
```

# Program Structure: Compute (2/9)

- **Declaration Statements:** To declare use of variables

`int count, value;`

Data type → `int`

Names of variables ← `count, value;`

- **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`

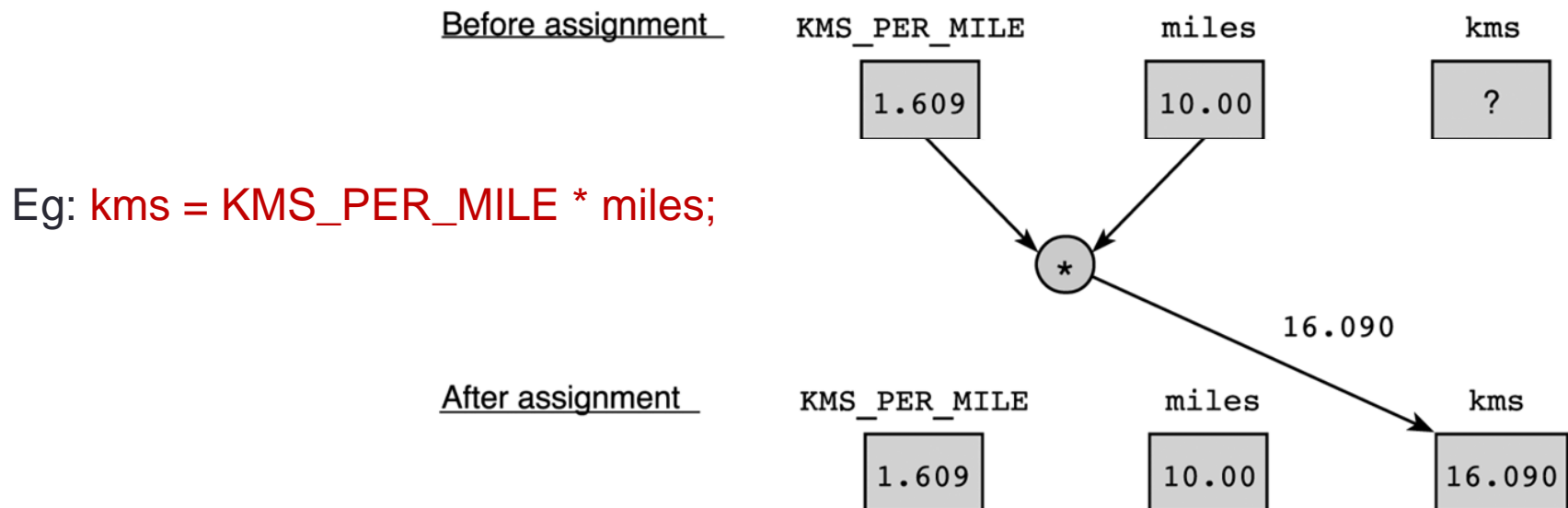
# Program Structure: Compute (3/9)

- **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



# Program Structure: Compute (4/9)

- 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**



# Which lines are valid in C++

```
#include <stdio.h>

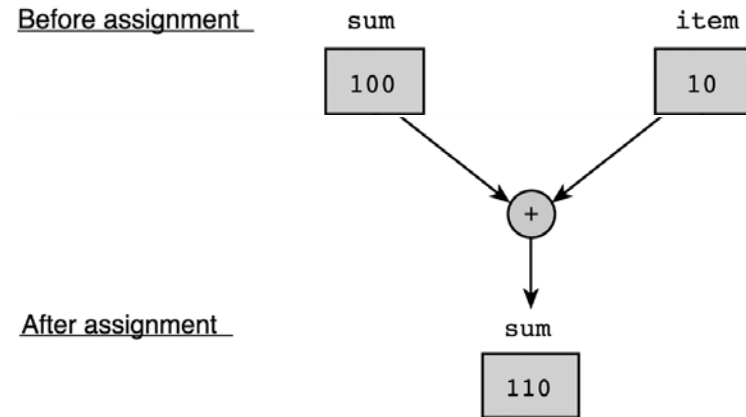
int main(void) {
    int a,b,c;
    a = 10;           ← ok
    b = a + 5;       ← ok
    a = f + 1;       ← NOT ok
    b = b + b;       ← ok
    a = b + c;       ← ok at compilation
                    May not be ok at run time
    35 = a;          ← NOT ok
    a + b = 2;       ← NOT ok

    return 0;
}
```

# Program Structure: Compute (5/9)

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;`

# Is this ok?

```
#include <stdio.h>

int main(void) {
    int a,b,c;

    a = b + c = 3;

    return 0;
}
```

- I thought it was equivalent to

```
a = b + (c = 3);
```

- But, no.... gcc understand this as:

```
a = (b + c) = 3;
```

WHY?!

Wait until your Compiler class

# Program Structure: Compute (6/9)

## □ 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.

# Program Structure: Compute (7/9)

- **Arithmetic operations**
  - **Binary Operators: +, -, \*, /, %** (modulo or remainder)
    - **Left Associative** (from left to right)
      - $46 / 15 / 2 \Rightarrow (46 / 15) / 2 \Rightarrow 3 / 2 \Rightarrow 1$
      - $19 \% 7 \% 3 \Rightarrow (19 \% 7) \% 3 \Rightarrow 5 \% 3 \Rightarrow 2$
    - **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 (next page)
    - addition, subtraction are lower in precedence than multiplication, division, and remainder
  - Truncated result if result can't be stored (the page after next)
    - `int n;    n = 9 * 0.5;`    results in 4 being stored in n.

Try out **Unit3\_ArithOps.c**

# Program Structure: Compute (8/9)

- Arithmetic operators: Associativity & Precedence

Operator Type	Operator	Associativity
Primary expression operators	() expr++ expr--	L to R
Unary operators	* & + - ++expr --expr (typecast)	R to L
Binary operators	* / %	L to R
	+ -	
Assignment operators	= += -= *= /= %=	R to L

# Program Structure: Compute (9/9)

## ■ Mixed-Type Arithmetic Operations

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

Caution!

## ■ Type Casting

### □ Use a cast operator to change the type of an expression

- syntax: `(type) expression`

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

Try out **Unit3\_MixedTypes.c** and **Unit3\_TypeCast.c**



# IF NOT ENOUGH TIME

---

Jump to if-else

# Exercise #4: Temperature Conversion

- Instructions will be given out in class
- We will use this formula

$$celsius = \frac{5}{9} \times (fahrenheit - 32)$$

## Exercise #5: Freezer (1/2)

- Write a program `freezer.c` that estimates the temperature in a freezer (in °C) given the elapsed time (hours) since a power failure. Assume this temperature ( $T$ ) is given by

$$T = \frac{4t^2}{t + 2} - 20$$

where  $t$  is the time since the power failure.

- Your program should prompt the user to enter how long it has been since the start of the power failure in hours and minutes, both values in integers.
- Note that you need to convert the elapsed time into hours in real number (use type `float`)
  - For example, if the user entered `2 30` (2 hours 30 minutes), you need to convert this to `2.5 hours` before applying the above formula.

## Exercise #5: Freezer (2/2)

- Refer to the sample run below. Follow the output format.

```
Enter hours and minutes since power failure: 2 45
Temperature in freezer = -13.63
```

- How long does it take the freezer to get to zero degree?  
Which of the following is the closest answer?
  - a) 3 hours
  - b) 4 hours 10 minutes
  - c) 6 hours 30 minutes
  - d) 8 hours
- This exercise is mounted on CodeCrunch as a practice exercise.

# Math Functions (1/2)

- In C, there are many libraries offering functions for you to use.
- Eg: `scanf()` and `printf()` – requires to include `<stdio.h>`
- In Exercise #5, for  $t^2$  you may use `t*t`, or the `pow()` function in the math library: `pow(t, 2)`
  - `pow(x, y)` //computes x raised to the power of y
- To use math functions, you need to
  - Include `<math.h>` AND
  - Compile your program with `-lm` option (i.e. `gcc -lm ...`)
- See Tables 3.3 and 3.4 (pages 88 – 89) for some math functions

# Math Functions (2/2)

- Some useful math functions
  - Function `abs(x)` from `<stdlib.h>`; the rest from `<math.h>`

Function	Arguments	Result
<code>abs(x)</code>	int	int
<code>ceil(x)</code>	double	double
<code>cos(x)</code>	double (radians)	double
<code>exp(x)</code>	double	double
<code>fabs(x)</code>	double	double
<code>floor(x)</code>	double	double
<code>log(x)</code>	double	double
<code>log10(x)</code>	double	double
<code>ceil(x)</code>	double	double
<code>pow(x, y)</code>	double, double	double
<code>sin(x)</code>	double (radians)	double
<code>sqrt(x)</code>	double	double
<code>tan(x)</code>	double (radians)	double

*Function prototype:*

`double pow(double x, double y)`



function return type

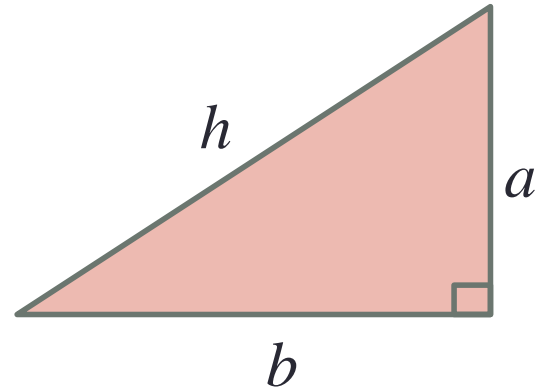
Q: Since the parameters `x` and `y` in `pow()` function are of double type, why can we call the function with `pow(t, 2)`?

A: Integer value can be assigned to a double variable/parameter.

# Math Functions: Example (1/2)

- Program `Unit3_Hypotenuse.c` computes the hypotenuse of a right-angled triangle given the lengths of its two perpendicular sides

$$h = \sqrt{a^2 + b^2}$$



# Math Functions: Example (2/2)

Unit3\_Hypotenuse.c

```
// Unit3_Hypotenuse.c
// Compute the hypotenuse of a right-angled triangle.
#include <stdio.h>
#include <math.h> ← Remember to compile with -lm option!

int main(void) {
    float hypot, side1, side2;

    printf("Enter lengths of the 2 perpendicular sides: ");
    scanf("%f %f", &side1, &side2);

    hypot = sqrt(side1*side1 + side2*side2);
    // or hypot = sqrt(pow(side1, 2) + pow(side2, 2));

    printf("Hypotenuse = %6.2f\n", hypot);

    return 0;
}
```



# Notes (1/2)



- Basic steps of a simple program
  1. Read inputs (scanf)
  2. Compute
  3. Print outputs (printf)
- For now we will use interactive inputs
  - Standard input stream (stdin) – default is keyboard
  - Use the **scanf()** function
- Assume input data always follow specification
  - Hence no need to validate input data (for now)
- Outputs
  - Standard output stream (stdout) – default is monitor
  - Use the **printf()** function

## Notes (2/2)



- Include header file `<stdio.h>` to use `scanf()` and `printf()`
  - Include the header file (for portability sake) even though some systems do not require this to be done
- Read
  - Lessons 1.6 – 1.9
- Important! (CodeCrunch issue)
  - Make sure you have a newline character (`'\n'`) at the end of your last line of output, or CodeCrunch may mark your output as incorrect.

```
printf("That equals %9.2f km.\n", kms);
```

# Type of Errors

## ■ Syntax errors (and warnings)

- Program violates syntax rules
- Warning happens, for example, incomparable use of types for output
- Advise to use **gcc -Wall** to compile your programs

Easiest to spot – the compiler helps you!

## ■ Run-time errors

- Program terminates unexpectedly due to illegal operations, such as dividing a number by zero, or user enters a real number for an integer data type

Moderately easy to spot

## ■ Logic errors

- Program produces incorrect result

Hard to spot

## ■ Undetected errors

- Exist if we do not test the program thoroughly enough

May never be spotted!

The process of correcting errors in programs is called **debugging**.  
This process can be **very** time-consuming!

# Programming Style (1/2)

- Programming style is just as important as writing a correct program
- Refer to some C Style Guides on the CS1010 website  
[http://www.comp.nus.edu.sg/~cs1010/2\\_resources/online.html](http://www.comp.nus.edu.sg/~cs1010/2_resources/online.html)
- In your lab assignments, marks will be awarded to style besides program correctness
  - Correctness: 60%
  - **Style: 20%**
  - Design: 20%

# Programming Style (2/2)

- Identifier naming for variables and functions
  - Use lower-case with underscore or capitalise first character of every subsequent word (Eg: `celsius`, `sum`, `second_max`, `secondMax`; NOT `Celsius`, `SUM`, `SecondMax`)
  - Must be descriptive (Eg: `numYears` instead of `ny`, `abc`, `xbrt`)
- User-defined constants
  - Use upper-case with underscore (Eg: `KMS_PER_MILE`, `DAYS_IN_YEAR`)
- Consistent indentation
- Appropriate comments
- Spacing and blank lines
- And many others



In vim, typing  
`gg=G`  
would auto-indent your  
program nicely!

# Type of Errors

## ■ Syntax errors (and warnings)

- Program violates syntax rules
- Warning happens, for example, incomparable use of types for output
- Advise to use **gcc -Wall** to compile your programs

Easiest to spot – the compiler helps you!

## ■ Run-time errors

- Program terminates unexpectedly due to illegal operations, such as dividing a number by zero, or user enters a real number for an integer data type

Moderately easy to spot

## ■ Logic errors

- Program produces incorrect result

Hard to spot

## ■ Undetected errors

- Exist if we do not test the program thoroughly enough

May never be spotted!

The process of correcting errors in programs is called **debugging**.

This process can be **very** time-consuming!

# There are two types of people

```
#include <stdio.h>

int main(void) {
    int a,b,c;
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    int a,b,c;
    return 0;
}
```

# Common Mistakes (1/2)

- Not initialising variables

**EXTREMELY COMMON MISTAKE**

- Program may work on some machine but not on another!

Cannot assume that the initial value of b is zero!

```
int a, b;  
a = b + 3; // but what is the value of b?
```

- Unnecessary initialisation of variables

```
int x = 0;  
x = 531;
```

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

- Forgetting **&** in a scanf() statement

```
int x;  
scanf("%d", x);
```



```
int x;  
scanf("%d", &x);
```





## Common Mistakes (2/2)

- Forgetting to compile with `-lm` option when the program uses math functions.
- Forgetting to recompile after modifying the source code.



Sometimes when your program crashes, a “core dump” may happen. Remove the file “core” (UNIX command: `rm core`) from your directory as it takes up a lot of space.

# FAQs

Why is there a `'return 0;'` at the end of every program?

Our program is a function (the 'main' function), and it is defined as `'int main(void)'` – it has no parameters and it returns an integer value. Hence we add `'return 0;'` at the end of the function to return 0. Return to where? Return to the operation system in which the program is run. In our case it is UNIX. UNIX takes the return value of 0 to mean a successful run.

There are so many ways to format outputs in `printf()`, do I have to know them all?

You will use the basic ones like `%d` for integers, `%f` for float and double, and `%c` for characters for now, and simple formatting such as the number of decimal places to be displayed.

We will focus more on the [problem solving aspects](#) in this module than on complex output formatting.



# Learning Outcomes



SUMMARY

Knowing the basic **C** constructs, interactive input, output, and arithmetic operations

Knowing some **data types** and the use of **variables** to hold data

Be aware of some basic **common mistakes**

Using some **math functions**

Understanding good **programming style**



## 2.1 *if* and *if-else* Statements

- *if* statement

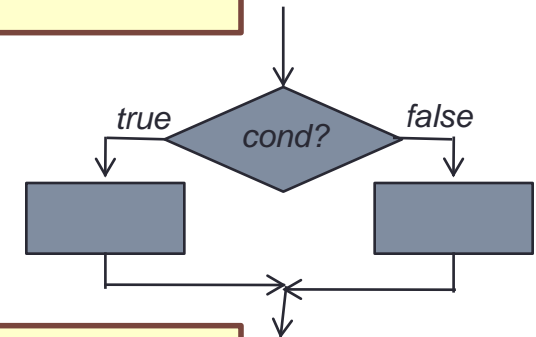
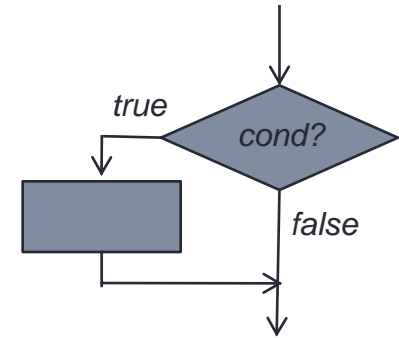
How are conditions specified and how are they evaluated?

```
if ( condition ) {
    /* Execute these statements if TRUE */
}
```

Braces { } are optional only if there is one statement in the block.

- *if-else* statement

```
if ( condition ) {
    /* Execute these statements if TRUE */
}
else {
    /* Execute these statements if FALSE */
}
```



## 2.2 Condition

- A **condition** is an expression evaluated to true or false.
- It is composed of expressions combined with **relational operators**.
  - Examples:  $(a \leq 10)$ ,  $(\text{count} > \text{max})$ ,  $(\text{value} \neq -9)$

Relational Operator	Interpretation
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

## 2.8 *if* and *if-else* Statements: Examples (1/2)

*if* statement  
without *else* part

```
int a, b, t;
. . .
if (a > b) {
    // Swap a with b
    t = a; a = b; b = t;
}
// After above, a is the smaller
```

*if-else* statement

```
int a;
. . .
if (a % 2 == 0) {
    printf("%d is even\n", a);
}
else {
    printf("%d is odd\n", a);
}
```

```
#include <stdio.h>

int main() {
    int a,b;
    printf("Please enter the first number: ");
    scanf("%d",&a);
    printf("Please enter the second number: ");
    scanf("%d",&b);

    if (a > b)
    {
        printf(" The first number is bigger.\n");
    }
    else
    {
        printf(" The second number is bigger.\n");
    }

    return 0;
}
```