Programming Refresher Workshop

11, 12, 14 July 2017

Contents

- Objectives
- Staff
- Seniors
- Programme
- Website/Topics
- Useful software and documents

Objectives

- To provide a refresher on programming and problem-solving skills covered in the first programming course (CS1010 or its equivalent)
- Targeted at incoming students holding polytechnic diploma who are exempted from CS1010 (Programming Methodology) or its equivalent.
- To allow students to better assess their preparedness for the follow-up module(s) after CS1010
  - IS/BZA students take CS1020 (Data Structures and Algorithms I)
  - CS students take CS2040 (Data Structures and Algorithms) and CS2030 (Programming Methodology II)
  - InfoSec/CEG students take CS2040C (Data Structures and Algorithms)
- If you decide to take CS1010 after this workshop, please request for a form from us, fill it up and submit to our UG office.

Staff

A/P Tan Sun Teck
tanst@comp.nus.edu.sg

Mr Aaron Tan
tantc@comp.nus.edu.sg

Dr Henry Chia
hchia@comp.nus.edu.sg

Seniors

Chua Zhi Jie

Koh Min Xuan

Choo Wen Xin

Puah Jia Hui

Ng Tzer Bin

Sim Kwan Tiong, Damien

Programme

- Three days with six sessions:
  - 11 July 2017, Tuesday: Session 1 (AM) and Session 2 (PM)
  - 12 July 2017, Wednesday: Session 3 (AM) and Session 4 (PM)
  - 14 July 2017, Friday: Session 5 (AM) and Session 6 (PM)
- Each session
  - AM session: 9 am – 12 noon
  - PM session: 1 – 4pm
  - Venue: PL2 (COM1 basement)

Website/Topics

- http://www.comp.nus.edu.sg/~tantc/refresher

Day	Session	Lecturer	Topics
Day 1	AM	A/P Tan Sun Teck	Intro; S/W development cycle; Control structures; CodeCrunch
	PM	Mr Aaron Tan	Subprograms; parameters; pre- and post-conditions; program testing
Day 2	AM	A/P Tan Sun Teck	1-dimensional arrays
	PM	Mr Aaron Tan	2-dimensional arrays
Day 3	AM	Dr Henry Chia	Recursion
	PM	Dr Henry Chia	Number processing

Useful Software and Documents

- CodeCrunch
  - A lab exercise portal.
  - Can support C/C++ and Java.
  - Download the exercise.
  - Develop your solution in your computer.
  - Submit your solution to CodeCrunch.
  - Check the result.

Useful Software and Documents

- Cygwin
  - A UNIX-like environment.
  - Need to know UNIX commands
    - ls, mkdir, cd, cp, mv, etc
  - Commands to compile programs
    - **javac myProg.java** for Java program
    - **gcc myProg.c** for c program
    - **g++ myProg.cpp** for C++ program
    - You may add different compilation options as required. For example, to highlight all warnings.
- You should try to learn a UNIX editor such as vim

Useful Software and Documents

- Cygwin/MinGW Installation Guide
- Introduction to Unix commands and Running Java Programs
- CodeCrunch guide

Programming Refresher Workshop

Session 1 A/P Tan Sun Teck

Contents

- Problem Solving Life Cycle
- Different view of programming

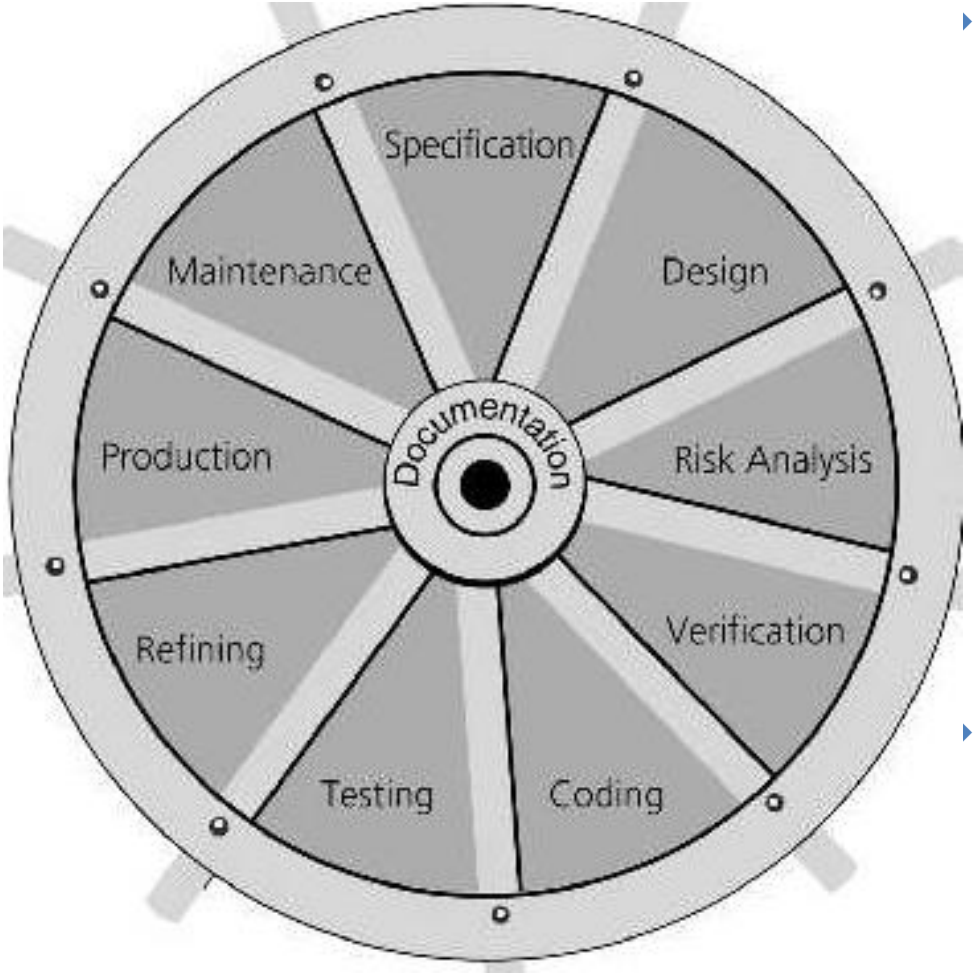
What is Algorithmic Problem Solving?

- The entire process of taking the statement of a problem and developing a computer program to solve the problem
  - Example: To solve a quadratic equation

Program = algorithm + data structure

  - Algorithm: a step-by-step specification of a method to solve a problem within a finite amount of time
  - Data structure: ways to store information

The Life Cycle of Software as a Water Wheel



- We'll cover only aspects that play a crucial role in data structures
  - Specification
  - Design
  - Verification
  - Coding
  - Testing
- The other parts will be covered in later semesters, especially in Software Engineering

Phase 1: Specification

Make the problem statement precise and detailed

For example:

- What is the input data?
- What data is valid and what data is not valid?
- Who will use the software, what user interface should be used?

A prototype program can clarify the problem: a simple program that simulates the behavior and illustrates the user interface

Phase 2: Design

Divide a large problem into small modules:

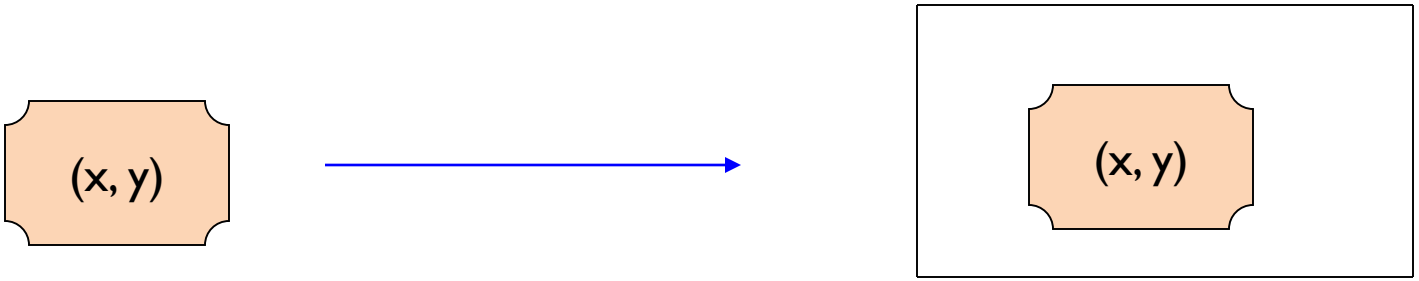
- Loosely coupled modules are independent
- Each module should perform one well-defined task (highly cohesive)
- Specify data flow among modules
  - E.g., purpose, assumptions, input, and output
  - It is NOT a description of what methods to use to solve the problem; just a decomposition into smaller tasks

Phase 2: Design (cont.)

- View Specifications as a contract

  Example: To design a method for a shape object that moves it to a new location on the screen. Possible specifications:

  - *The method will receive an (x, y) coordinate.*
  - *The method will move the shape to the new location on the screen*



Phase 2: Design (cont.)

- A module's specification should not describe a method of solution.
- Method specifications include precise *pre-conditions* and *post-conditions ;* identify the method's formal parameter, etc.
- Incorporate existing software components in your design.

Phase 2: Design (cont.)

First-draft specifications

move (x, y)
// Move a shape to a new location on the screen
// **Pre-condition:** The calling program provides an
// (x, y) pair, both integers.
// **Post-condition:** The shape is moved to the new location
// (x, y)

Phase 2: Design (cont.)

Revised specifications

move (x, y)
// Move a shape to a new location on the screen
// **Pre-condition:** The calling program provides an
// (x, y) pair, both integers, where
// 0 <= x <= MAX_XCOOR, 0 <= y <= MAX_YCOOR,
// where MAX_XCOOR and MAX_YCOOR are class
// constants that specify the maximum coordinate values.
// **Post-condition:** The shape is moved to the new location
// (x, y)

Algorithm

- Similar to a recipe for cooking
  - You must know how to cook a dish before you can write the recipe.
- It is a step by step instruction for solving a problem.
  - You must know how to solve the problem before you can write the program.
- An algorithm is commonly presented in *pseudo-code*.

Phase 4: Verification

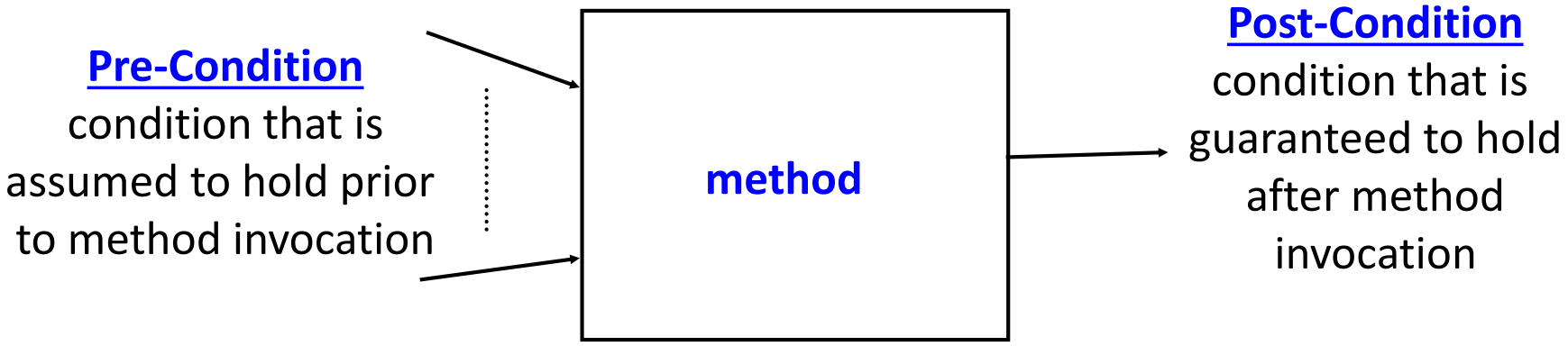
- Formal theoretical methods are available for proving the correctness of an algorithm
  - still a research subject
- Some aspects of the verification process
  - Assertion
  - Invariant

An assertion is a statement about a particular condition at a certain point in an algorithm

An invariant is a condition that is always true at a particular point of the algorithm

Phase 4: Verification - Assertion

- An assertion is a statement about a particular condition at a certain point in an algorithm.
  - special case: pre/post-conditions



Phase 4: Verification - Example

Revised specifications

move (x, y)
// Move a shape to a new location on the screen
// **Pre-condition:** The calling program provides an
// (x, y) pair, both integers, where
// 0 <= x <= MAX_XCOOR, 0 <= y <= MAX_YCOOR,
// where MAX_XCOOR and MAX_YOOR are class
// constants that specify the maximum coordinate values.
// **Post-condition:** The shape is moved to the new location
// (x, y)

Phase 5: Coding

- Translating the design into a particular programming language

- Coding is a relatively minor phase in the software life cycle.

Phase 6: Testing

- Design a set of test data to test the program

- Testing is both a science and an art

What is a good solution?

- When the total cost incurred over all phases of the life cycle is minimal
- Programs must be well structured and documented
- Efficiency is important
  - Using the proper algorithms and data structures can lead to significant differences in efficiency
  - In many instances, the specific style of coding matters less than the choice of data structures

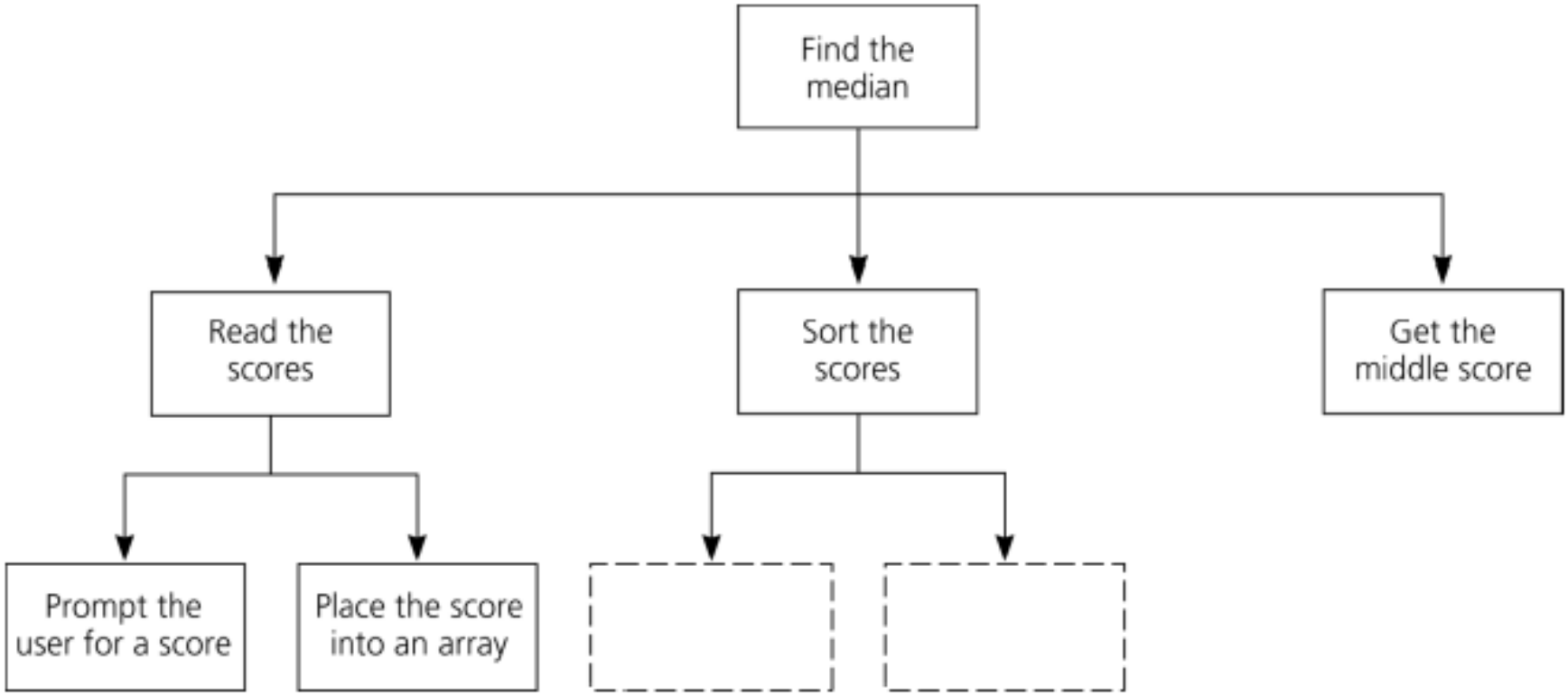
Top-Down Design

Use it:

- When designing an algorithm for a method
- When the emphasis is on algorithms and not on the data.

- A structure chart shows the relationship among modules.
- A solution consisting of independent tasks.

Example: Find the Median Score



Six Key Programming Issues

1. Modularity
2. Modifiability
3. Ease of use
4. Fail-safe programming
5. Style
6. Debugging

Modularity

- Facilitates programming
- Isolates errors
- Programs are easy to read
- Isolates modifications
- Eliminates redundancies

Modifiability

- Methods make a program easier to modify

- Named constants make a program easier to modify

Ease of Use

- A good user interface, for example, prompt user for input
- A good manual

Fail-Safe Programming

A fail-safe program is one that will perform reasonably no matter how anyone use it:

- Check for errors in input
- Check for errors in logic
- Methods should check their invariants
- Methods should enforce their preconditions
- Methods should check the values of their arguments

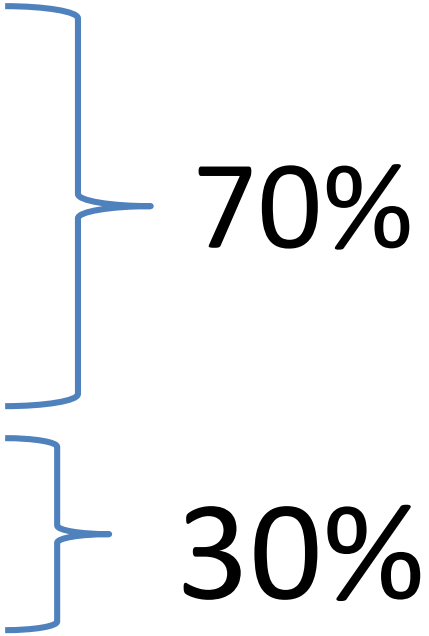
Debugging

- Use either watches, assertions or temporary System.out.println/printf/cout statements to find logic errors

- Systematically check a program's logic to determine where an error occurs

Problem Solving Life Cycle

- Understand the problem
- Specification
- Analysis
- Algorithm design
- Implementation
- Testing
- Maintenance



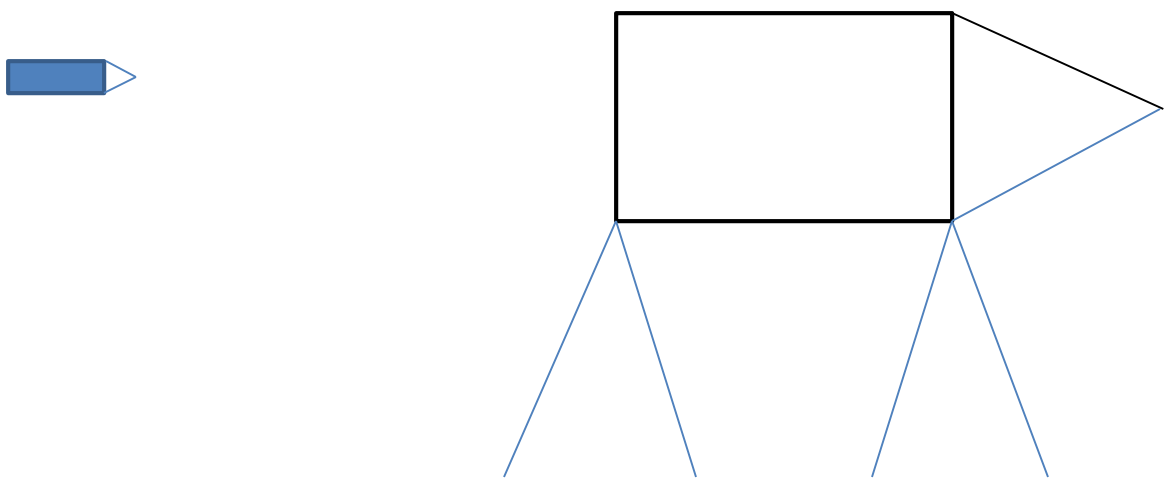
Example:

- Given 3 integer values, write a program to output the maximum.

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
if (a > b && a > c)
    printf("max is %d\n", a);
if (b > a && b > c)
    printf("max is %d\n", b);
if (c > a && c > b)
    printf("max is %d\n", c);
```

Problem Solving

- The animal is formed by 10 sticks.
- Move 2 sticks so that the animal can avoid being hit by the bullet.



Different View of Programming

- Program = Data Structure + Algorithm
- How to store information in computer?
- How to process the information to produce the required result?

Programming Languages

- C/C++, Java, C#
- Syntax (Grammar of the language)
- Semantic (Meaning of the language)

Syntax

- Identifier
  - Must begin with a alphabet or a _
  - Must not have any special character
- Each statement must be terminated by a semi-colon.
- Etc.

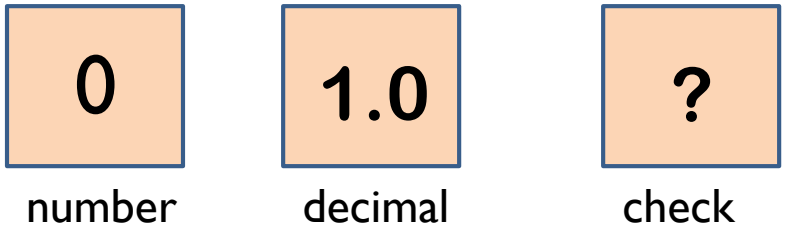
Semantics

- Consider programming to be putting values into boxes.
  - Input statements, assignment statement
- Taking the values out of the boxes and perform some operations on them
  - Using operators such as *, /, +, -, %, ==, <, >, <=, >=, !=, ||, &&
- Output the final results
  - Output statements

Variables: Creating the boxes

- Give an identity to each box.
- Specify what type of value can be put into the box.
- Put an initial value into the box.

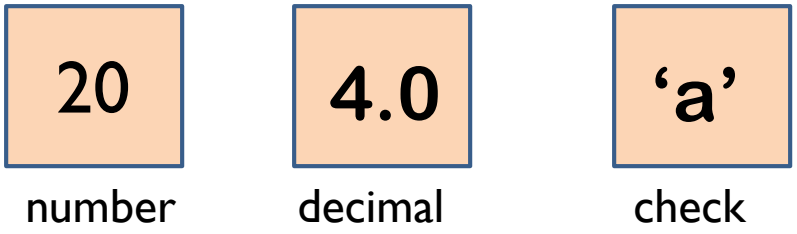
```
int number = 0;
float decimal = 1.0;
char check;
```



Variables: Put values into the boxes

- Assignment statements
- Input statements

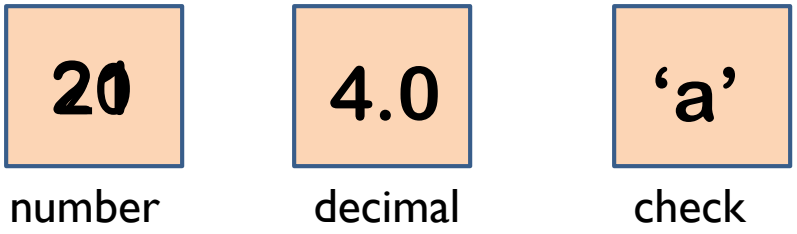
```
number = 20;
decimal = 4.0;
scanf("%c", &check);
```



Variables: Get values out of the boxes

- To do calculations
- To make decisions
- To output the results

```
number = number + 1;
if (sqrt(decimal) == 2.0);
  printf("perfect square");
scanf("%c", &check);
```



Beware of errors that are difficult to discover

Arithmetic: Different from normal Math

- `number = number + 1;`
- `number = number / 10;`
- `number = number % 10;`
- Be careful about the difference between

  `number = 1`

  and

  `number == 1`

Sequential Construct

- Statements are executed sequentially one after another.
- When a function is called, the function must be executed entirely before the statement after the function is executed.
- Compound statement.
  - Compound statement are created by putting many single statements into a pair of braces, '{' and '}'

Conditional Construct

- Making decision
- Each of the conditional construct is considered as one statement.
- You may nest any other valid statements within the construct.
- Simple if statement
- if-else statement

```
if (a == b)
   printf("%d and %d are equal\n", a, b);
```

```
if (a > b && a > c)
    max = a;
else
    if (b > a && b > c)
        max = b;
    else
        max = c;
```

Is this correct?

Conditional Construct

- Be careful with the pairing of if-else, the following has a totally different meaning as what is intended.
- Indentation does not mean the else statement is paired with the first if statement.
- When in doubt, use braces to ensure the pairing

```
if (a > b && a > c)
    max = a;
    if (b > a && b > c)
        max = b;
else
        max = c;
```

```
if (a > b && a > c) {
    max = a;
    if (b > a && b > c)
        max = b;
}
else
        max = c;
```

switch Statement

- Nested if statements are difficult to write and difficult to understand.
- The switch statement are normally used if there are only a limited discrete values for the control variables.

Iterative Constructs

- for loop

```
for (initialisation; condition; modification) {
}
```

- Initialisation: to set an initial value for loop control variable(s). Eg. **j = 0;**
- Condition: The termination condition to terminate the loop when it becomes false. Eg. **j < 10;**
- Modification: modify the control variable so that the termination condition will eventually become true. Eg. **j++;**

Iterative Constructs

- while loop

```
while (condition) {
    <loop body>
}
```

- do-while loop

```
do
    <loop body>
while (condition)
```

- Initialisation of the variables in the conditions are normally done outside the loop.
- Modification of the values for the variables are done in the loop.
- Loop will terminate when condition becomes false.
- While loop may not be executed at all but the do while loop will execute at least one time.

The End