CS1010E TOPIC 1: GETTING STARTED WITH PROGRAMMING

Siau-Cheng KHOO

Block COM2, Room 04-11, +65 6516 6730

www.comp.nus.edu.sg/~khoosc

khoosc@nus.edu.sg

Semester II, 2017/2018

Course Outline

- Introduce the fundamental constructs of programming
- C is the programming language taught
- Basic programming methodology, not just C
- Preparation for CS1020E and other computing courses

- Learning Outcome:
  - Ability to understand basic algorithms and implement them in C

Hardware vs Software

Hardware

Monitor and speaker **(output)**

Houses processor, memory, buses, etc.

Keyboard and mouse **(input)**

Software

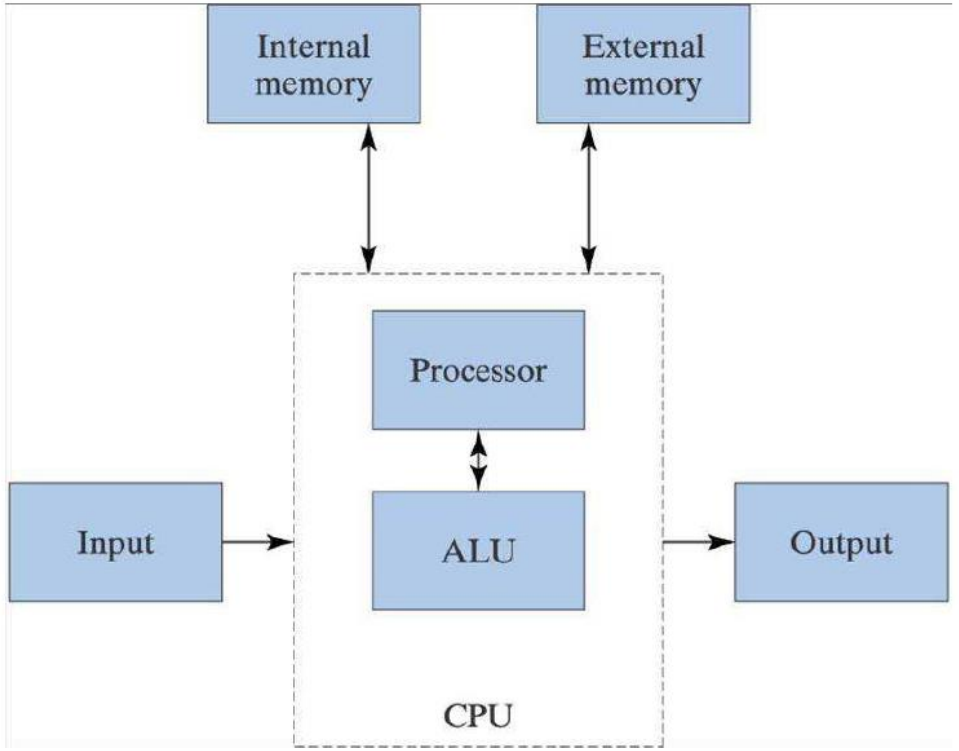
- Set of instructions to perform some tasks
- Programs are software

http://www.tutorialspoint.com/computer_fundamentals/computer_quick_guide.htm

Computer Hardware

- A computer is a machine that is designed to perform tasks that are specified with a set of instructions called a program

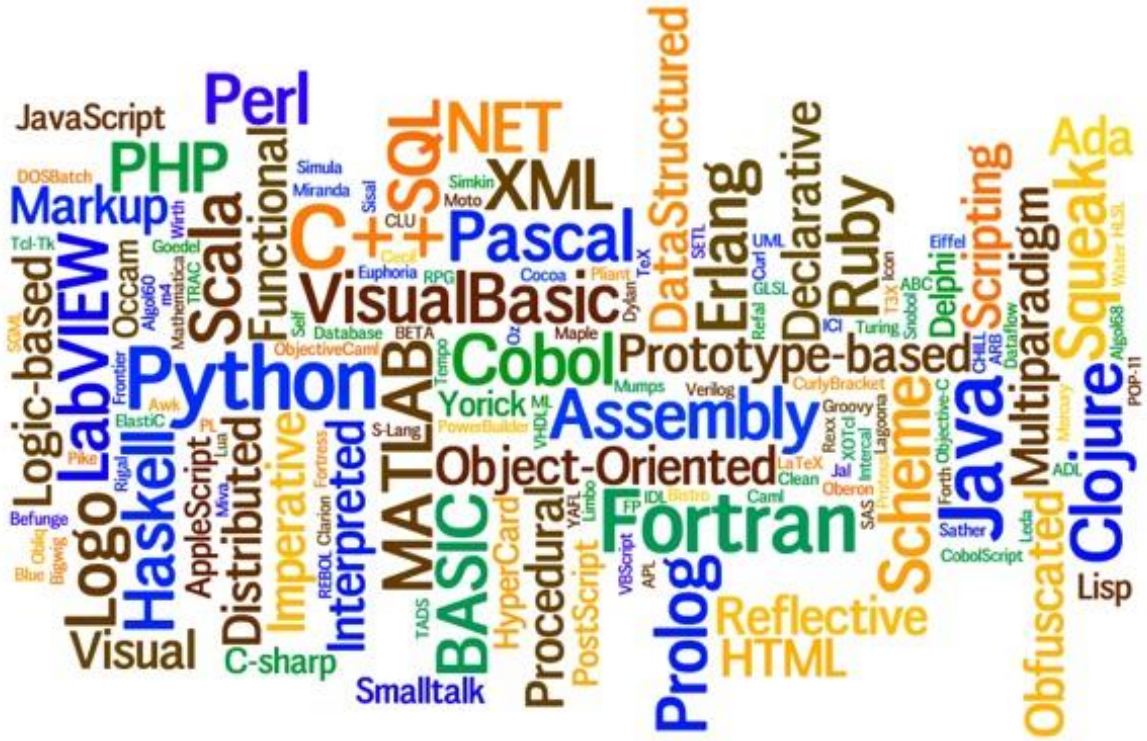
Internal Organization



Programs

- (Computer) Program
  - Sequence of instructions for a computer to execute, thus performing a task
- Programming languages
  - Languages for writing programs

Software



Why C Programming Language?

- Most widely used in engineering fields
- Closest to the computer's memory model
- Covers fundamental features that support understanding toward other programming languages

Lecture Outline

- C Program Structure – First impression
- How C program runs: understanding its memory model
- Problem Solving Process (aka., Computational Thinking)
- Looking into Solving two computational problems

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

Input

Compute

Output

This lecture covers

- Basic structure of C programs
- Basic Memory model used by Running a C program
- Developing C programs
- Maintaining C programs
- Generic Problem Solving Process (Computational Thinking)

Understanding C Programs

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The main function

Understanding C Programs

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

Variables holds values in memory

Understanding C Programs

```
#include <stdio.h>

int main(void)
{
   double a, b, area;

   printf("Er                                    \n");
   scanf("%lf

   area = a*b

   printf("Th                                    :"
          "

   return 0;
}
```

Variables holds values in memory

@ a = 2; @ b is 8; @ area = 16

0	2 **10.0**	4	6	8 **15.0**
10	12	14	16 **150.0**	18
20	22	24	26	28
30	32	34	36	38
40	42	44	46	48

Understanding C Programs

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

Declaration statement declares variables used

Understanding C Programs

```
#include <stdio.h>

int main(void)
{
   double a, b, area;

   printf("Er                                          \n");
   scanf("%lf

   area = a*b

   printf("Th                                          :"
          "


   return 0;
}
```

Declaration statement declares variable used

@ a = 2; @ b is 8; @ area = 16

0	2	4	6	8
10	12	14	16	18
20	22	24	26	28
30	32	34	36	38
40	42	44	46	48

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The printf statement outputs info

Input

Output

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The scanf statement reads in data

Input

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
   double a, b, area;

   printf("Enter the lengths of the sides of the rectangle:  \n");
   scanf("%lf %lf", &a, &b);

   area = a*b

   printf("Th                                                  "
          "

   return 0;
}
```

The scanf statement reads in data

@ a = 2; @ b is 8; @ area = 16

Input

0	2 **10.0**	4	6	8 **15.0**
10	12	14	16	18
20	22	24	26	28
30	32	34	36	38
40	42	44	46	48

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a * b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The * operator performs computation

Common binary operators : + - * /

Compute

A C Program to find the area of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a * b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The = operator assigns RHS to variable at LHS

Assignment statement places value into memory

Compute

A C Program to find the area of a rectangle

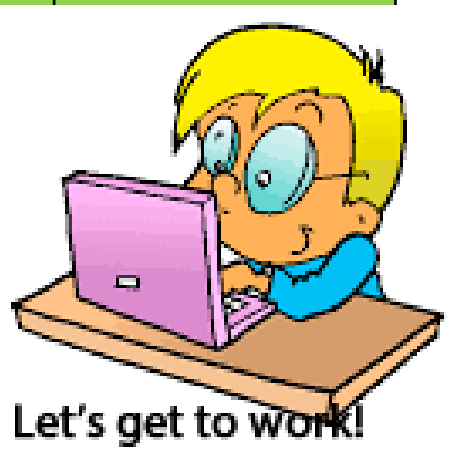
```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle: \n");
  scanf("%lf %lf"

  area = a * b;

  printf("The area
          " is %5

  return 0;
}
```

@ a = 2; @ b is 8; @ area = 16

The = operator assigns computed RHS value to LHS variable

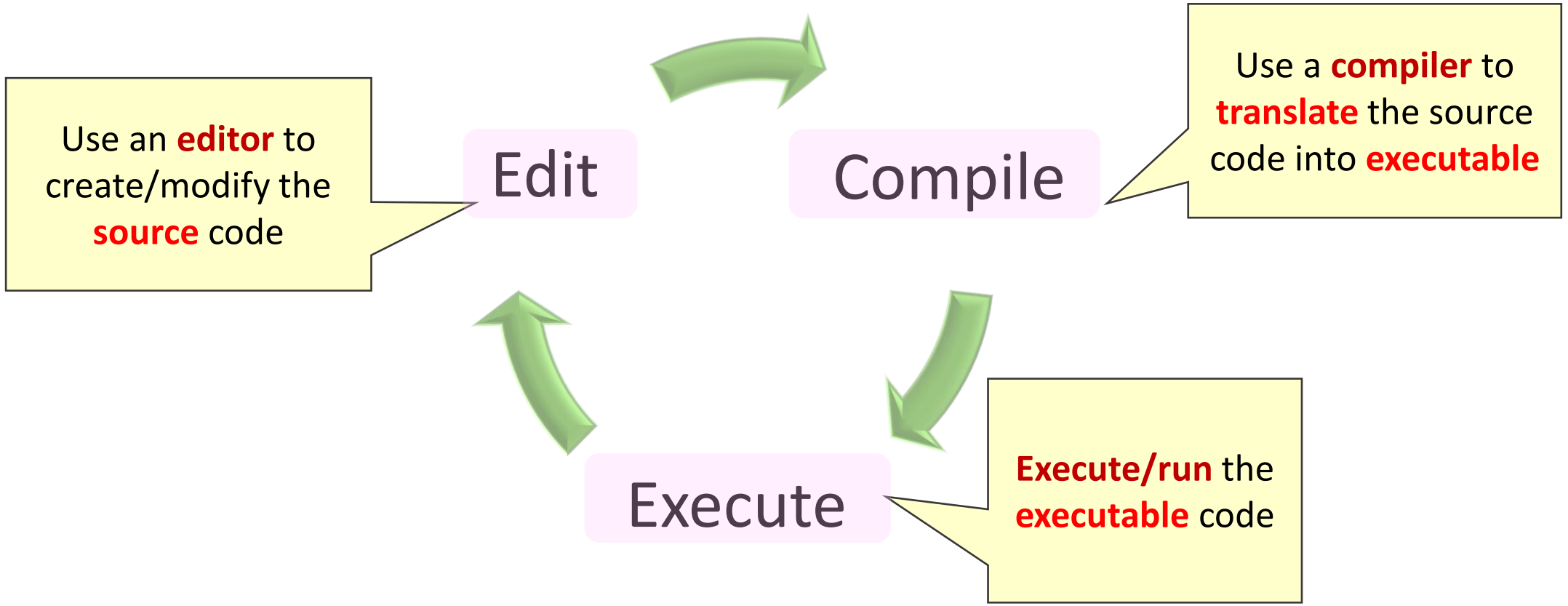
Assignment statement places value into memory

0	2 10.0	4	6	8 15.0
10	12	14	16 150.0	
20	22	24	26	
30	32	34	36	
40	42	44	46	



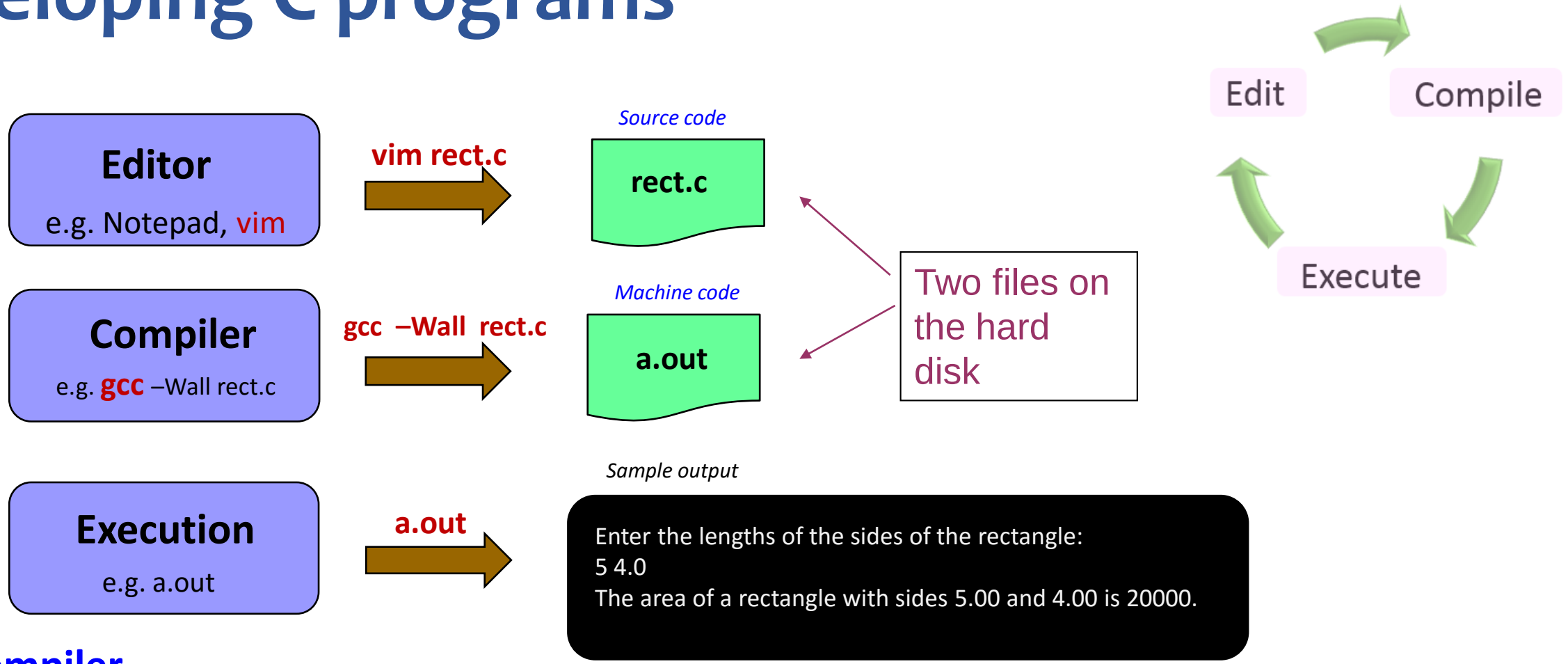
Developing C programs

The Edit, Compile and Execute Cycle



Process is iterative

Developing C programs



- **Compiler**
  - **Translate a program written in a high-level language to a program in a target (low-level) language.**

Programming Environment

SSH/Xshell are programs to allow users to access a remote host over a network.
To download SSH for your home use, go to
http://www.comp.nus.edu.sg/~cs1010/2_resources/online.html

sunfire UNIX server

To login to Sunfire server, you need an SoC **UNIX account username and password**. Create one now at:
https://mysoc.nus.edu.sg/~newacct

Your NUSNET account ends with @nus.edu.sg or @u.nus.edu
Your **UNIX account** ends with **@comp.nus.edu.sg**

Summary on Programming in C

- **Memory model**: Variable → address; address → values
- Retrieving values from memory: Referring to variable names
- Placing values into the memory:
  - Input values from outside: `scanf`
  - Physically put values into memory: **Assignment**
- Reading/understanding/tracing a C program
  - Look for the `main` function
  - Read **sequentially** top-down

Maintaining C programs using comments

```
/*
   Program rect.c
   Version 1  Author: Khoo S.C.   Date: 10 Jan 2018
   This program computes the area of a rectangle
   given the length of its two sides
*/
#include <stdio.h>

int main(void)
{
  double a, b, area;
  /* accept two real numbers as two sides of the rectangle */
  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;                    // compute the area

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area); // output of 2 precisions

  return 0;
}
```

A C Program to find the perimeter of a rectangle

```
#include <stdio.h>

int main(void)
{
  double a, b, area;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  area = a*b;

  printf("The area of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, area);

  return 0;
}
```

The old program

Input

Compute

Output

A C Program to find the perimeter of a rectangle

The new program

```
#include <stdio.h>

int main(void)
{
  double a, b, perimeter;

  printf("Enter the lengths of the sides of the rectangle:  \n");
  scanf("%lf %lf", &a, &b);

  perimeter = (a + b) * 2;

  printf("The perimeter of a rectangle with sides %5.2f and %5.2f"
         " is %5.2f. \n", a, b, perimeter);

  return 0;
}
```

A C Program to find the perimeter of a rectangle

```
#include <stdio.h>

int main(void)
{
   double a, b, perimeter;

   printf("Enter the lengths of t                    r the rectangle:  \n");
   scanf("%lf %lf", &a, &b);

   perimeter = (a

   printf(                 imeter of a rectangle with sides %5.2f and %5.2f"
                    %5.2f. \n", a, b, perimeter);

   return 0;
}
```

The new progr

You can now write and run your own programs!

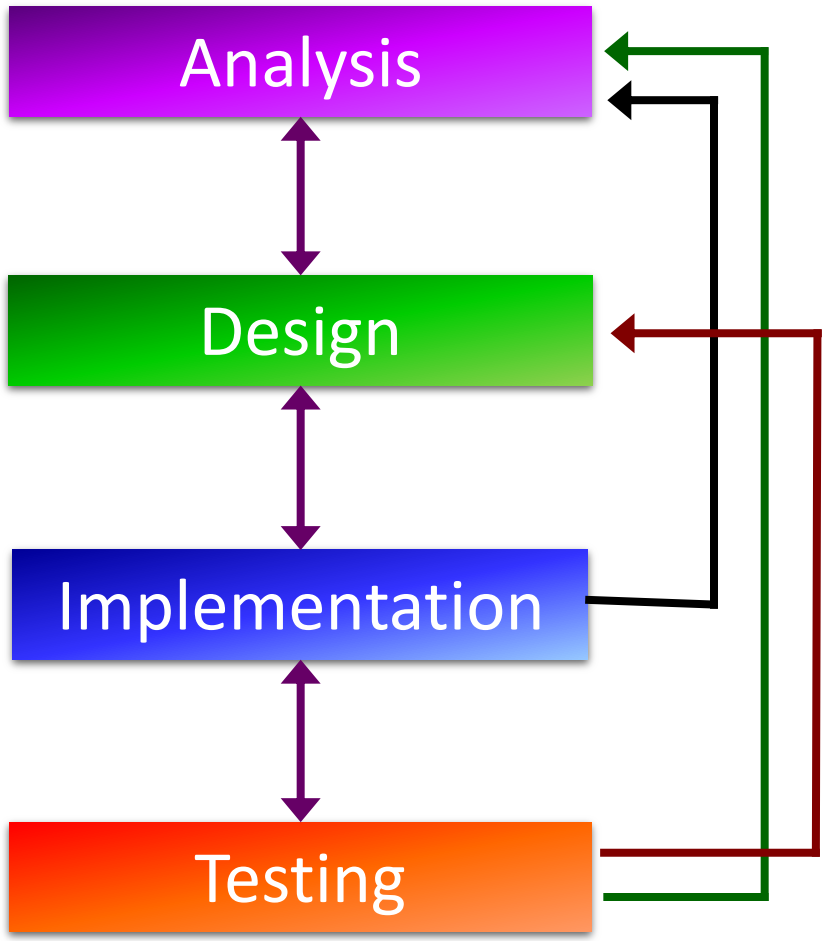
Problem #2

- **Given two non-negative integers *a* and *b*, find the greatest common divisor (GCD) of *a* and *b*.**
- **Examples:**
  - ***a* = 24, *b* = 60**

    ***Greatest common divisor is* 12**
  - ***a* = 1071, *b* = 462**

    ***Greatest common divisor is* 21**

Problem Solving Process



Iterative process

Similar in idea to Computational Thinking

Pólya: How to Solve It (1/5)

A great discovery solves a great problem but there is a grain of discovery in the solution of *any* problem.

Your problem may be modest; but if it challenges your curiosity and brings into play your inventive faculties, and if you solve it by ***your own means***, you may experience the tension and enjoy the triumph of discovery.

Such experiences at a susceptible age may create a taste for mental work and leave their imprint on mind and character for a lifetime.

– George Pólya

Pólya: How to Solve It (2/5)

- **Phase 1: Understanding the problem**
- Phase 2: Devising a plan
- Phase 3: Carrying out the plan
- Phase 4: Looking back
  - What is the unknown? What are the data?
  - What is the condition? Is it possible to satisfy the condition? Is the condition sufficient to determine the unknown?
  - Draw a figure. Introduce suitable notation.

Pólya: How to Solve It (3/5)

- Phase 1: Understanding the problem
- **Phase 2: Devising a plan**
- Phase 3: Carrying out the plan
- Phase 4: Looking back
  - Have you seen the problem before? Do you know a related problem?
  - Look at the unknown. Think of a problem having the same or similar unknown.
  - Split the problem into smaller sub-problems.

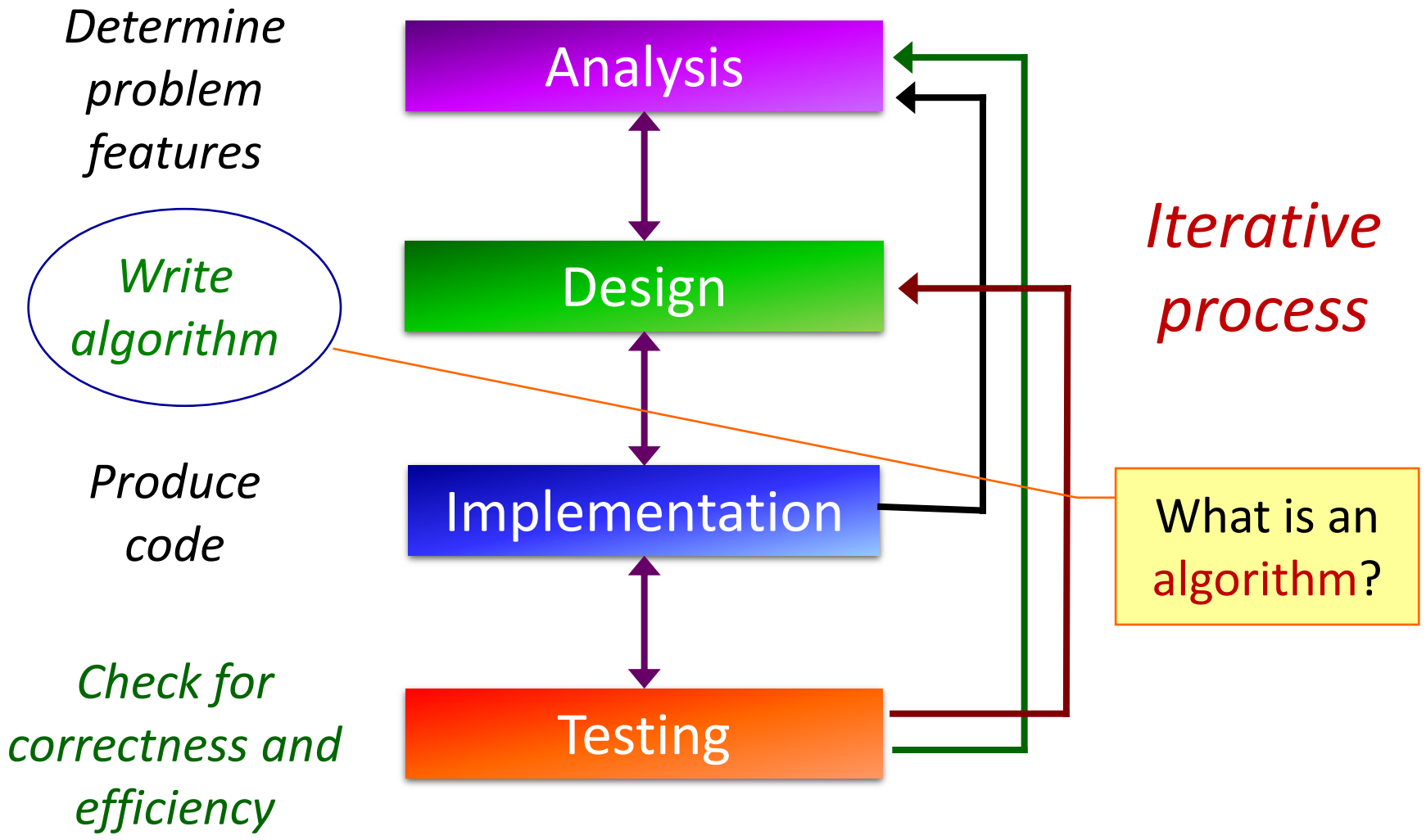
Pólya: How to Solve It (4/5)

- Phase 1: Understanding the problem
- Phase 2: Devising a plan
- **Phase 3: Carrying out the plan**
- Phase 4: Looking back
  - Carry out your plan of the solution. Check each step.
  - Can you see clearly that the step is correct?
  - Can you prove that it is correct?

Pólya: How to Solve It (5/5)

- Phase 1: Understanding the problem
- Phase 2: Devising a plan
- Phase 3: Carrying out the plan
- **Phase 4: Looking back**
  - Can you check the result?
  - Can you derive the result differently?
  - Can you use the result, or the method, for some other problem?

Algorithmic Problem Solving Process



Algorithmic Problem Solving

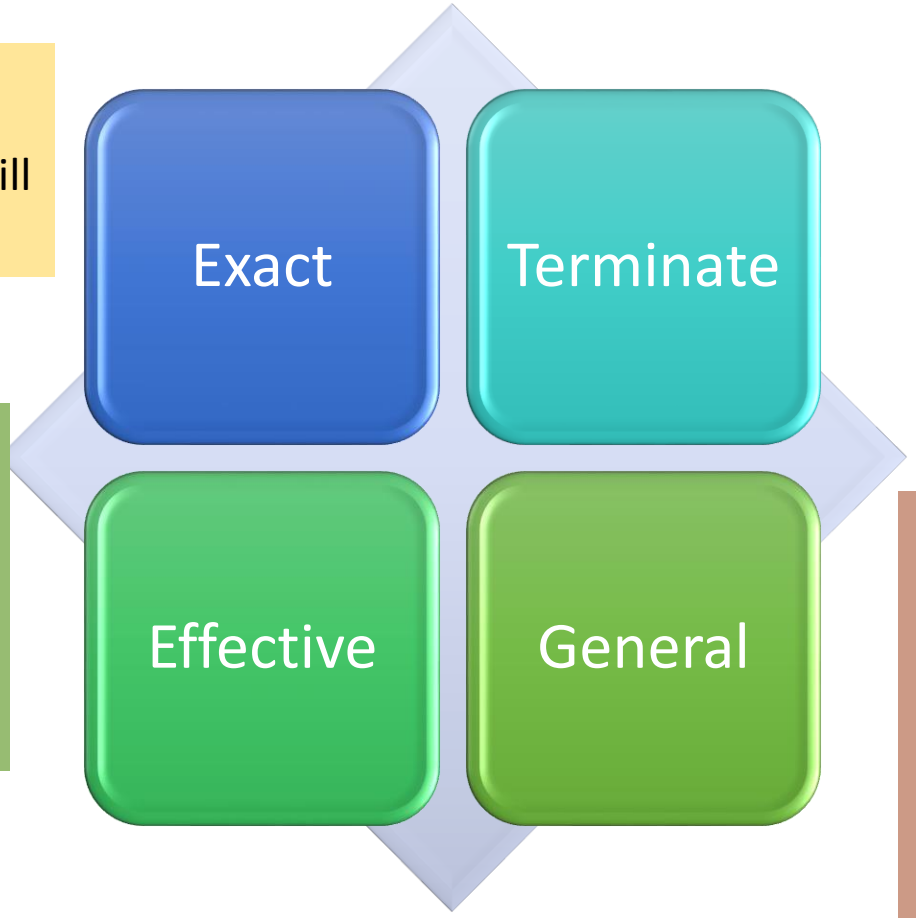
- An **algorithm** is a well-defined computational procedure consisting of *a set of instructions*, that takes some value or set of values, as *input*, and produces some value or set of values, as *output*.

This is the computational step in Computational Thinking

An Algorithm has these properties

Each step must be **exact**. (Or it will not be precise.)

The algorithm must **terminate**. (Or no solution will be obtained.)

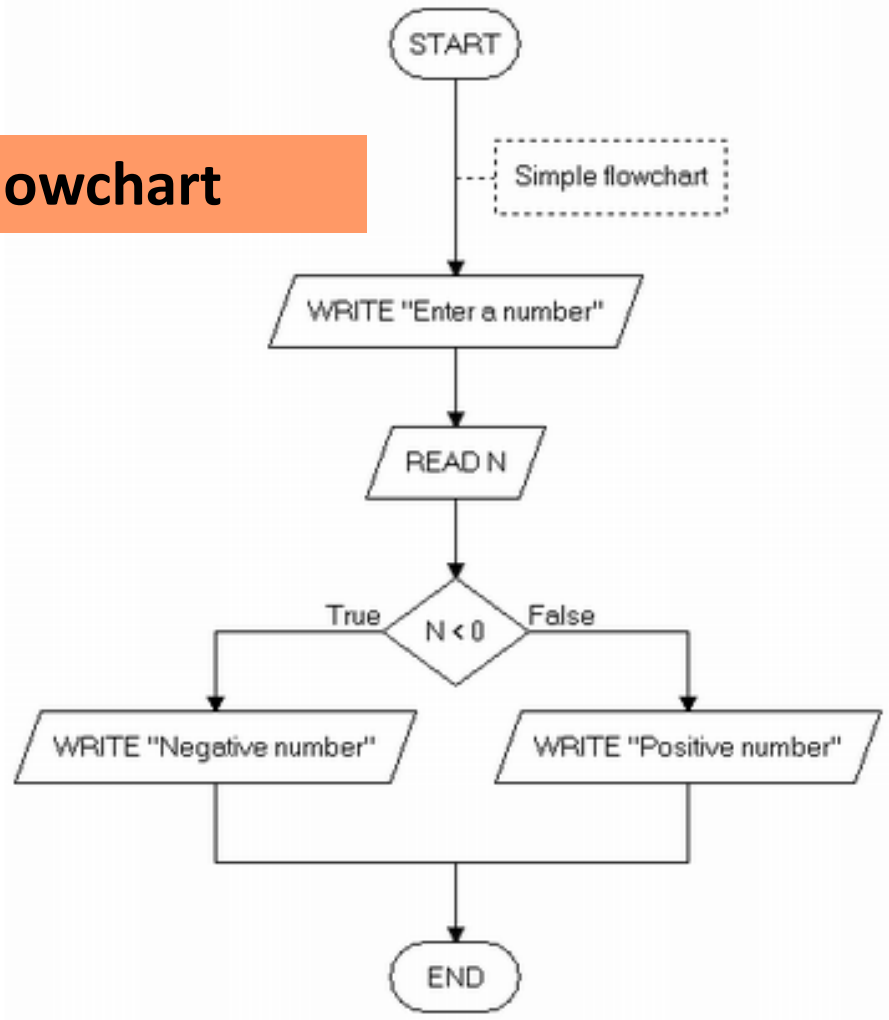


The algorithm must be **effective**. (i.e. it must solve the problem.)

The algorithm must be **general**. (Within the constraints of the system/language.)

Ways of representing an algorithm

Flowchart



Pseudocode

PSEUDOCODE

```
set total to zero

get list of numbers

loop through each number in the list
   add each number to total
end loop

if number more than zero
   print "it's positive" message
else
   print "it's zero or less" message
end if
```

lynda.com

Back to Problem #2

- **Given two positive integers *a* and *b*, find the greatest common divisor (GCD) of *a* and *b*.**
- **Examples:**
  - ***a* = 24, *b* = 60**

    ***Greatest common divisor is* 12**
  - ***a* = 1071, *b* = 462**

    ***Greatest common divisor is* 21**
- **A famous algorithm to compute GCD is called Euclid's Algorithm**

Euclid's Algorithm

- **To compute the greatest common divisor (GCD) of two non-negative integers *a* and *b*.**
- **First documented algorithm by Greek Mathematician Euclid in 300 B.C.**
- **Also called Euclidean Algorithm**

1. Let a and b be integers with $a > b \geq 0$.
2. If $b = 0$, then the GCD is a and algorithm ends.
3. Otherwise, find q and r such that

$$a = q * b + r \qquad \text{where } 0 \leq r < b$$

4. Replace a by b, and b by r. Go to step 2.

$a = 42, \quad b = 12$

$42 = 3 * 12 + 6$

$q = 3, \quad r = 6$

r **is the remainder of *a* / *b***
(*r* is "*a* modulo *b*")

Euclid's Algorithm (Pseudo-code)

```
// Assume A and B are non-negative
// integers, but not both zeroes.

Algorithm GCD(A, B) {
    while (B > 0) {
        r ← A modulo B
        A ← B
        B ← r
    }
    result is A
}
```

Let's trace GCD(12, 42)

(*B* > 0)?	*r*	*A*	*B*
		12	42
true	12	42	12
true	6	12	6
true	0	6	0
false			

Result: **6**

```
/*
 * CS1010E AY2017/18 Semester 2 GCD Computation
 */
// Pre-cond: a and b are non-negative integers, not both zero.
#include <stdio.h>

int main(void) {
	int a, b, rem;			// declaring variables

	printf("Enter two non-negative integers: ");
	scanf("%d %d", &a, &b) ;

	while (b>0) {
		rem = a % b;	// "a % b" is "a modulo b” in C
		a = b;
		b = rem;
	}

	printf(“The result of gcd is %d.\n", a) ;
	return 0;
}
```

Summary

- C Program Structure
  - A basic understanding
- How C program runs
  - Simple understanding of its memory model
- Problem Solving Process (aka., Computational Thinking)
- Looking into Solving two computational problems
  - Finding the area of a rectangle
  - Finding the greatest common divisor of two positive integers

Homework

- Learn the environment you use to develop programs
  - Learn from: http://www.comp.nus.edu.sg/~cs1010e/gettingStarted.html
  - Create your own Sunfire account ASAP
  - Learn to use vim editor while logging in to Sunfire
- Go through the edit-compile-execute loop of program development
- Remember the typical components of a program:
  - Comments, Declaration, input, compute, output
- Important: Remember to attend Tutorial Session in Week 2