PROGRAMMING I

Dr. Worarat Krathu

Asst. Prof. Dr. Chonlameth Arpnikanondt

Rules and Agreements

- Class
  - Lecture on every Tuesday (10.30-12.20)
  - Lab on every Friday (13.30-15.20)
- Scoring
  - Midterm exam (30%)
  - Final exam (30%)
  - Quiz (30%)
  - In-class exercise / attendance / discussion / assignment / project / etc. (10%)
  - Do or Die exam

Course Outline

Wk.	Description	Hrs
1	Course overview and planning for solutions	2
	Lab – logic practices with simple java	2
2	Introduction to Java, data type, expression and operations	2
	Lab	2
3	Selection	2
	Lab	2
4	Repetition	2
	Lab	2
5	Array	2
	Lab	2
6	Quiz	2
	Lab+ Quiz	2
7	Review	2
	Lab	2
8	Midterm Examination	
9-10	Object and class & method, parameter and class reference	4
	Lab	4
11-13	Other basic Java utility	6
	Lab	6
14-15	Introduction to object-oriented concept and object-oriented programming (e.g., inheritance, encapsulation, etc.)	4
	Lab	4
16	Do or Die Exam	4
17	Final Examination	

Resources

- Y. Daniel Liang, ***"Introduction to Java Programming, Comprehensive Version"***, Pearson Education Inc., **2012 (9th edition)**
- Stuart Reges and Marty Stepp, **"Building Java Programs: A Back to Basics Approach",** Peason Education Inc.**, 2017 (4th edition)**
- Paul Deitel and Harvey Deitel, ***"Java How to Program, Late Objects Version"***, Pearson Education Inc.**, 2010 (8th edition)**
- Class materials of INT102 of Dr. Umaporn Supasitthimethe
- http://docs.oracle.com/javase/7/docs/api/
- Class material is in Classroom on Demand

INTRODUCTION TO COMPUTER & PROGRAMS

Computer Systems

- A computer is an electronic device that stores and processes data according to a series of instructions.

- Hardware
  - The physical, tangible parts of a computer
  - Keyboard, monitor, disks, wires, chips, etc.

- Software
  - Programs and data
  - A program is a series of instructions

Software/Programs

- Computer programs, known as ***software***, are instructions to the computer such as Word processors, Internet browsers, Computer games, Spread sheets.

- Software consists of both programs and data
  - Programs are lists of instructions for the processor.
  - Data can be any information that a program needs: character data, numerical data, image data, audio data, and countless other types.

Software/Programs

- Computers do not understand human languages, so you need to use computer languages in computer programs.
- Each CPU executes instructions in its own unique native machine language which is in the form of binary code
- Programming in machine language is a tedious and time-consuming process. Moreover, the programs are highly difficult to read and maintain.
- For example, to add two numbers, might have the form

```
add a, b -->1101101010011011
```

What is Computer Science?

- Computer Science
  - The study of theoretical foundations of information and computation and their implementation and application in computer systems. -- Wikipedia
  - Many subfields
    - Graphics, Computer Vision
    - Artificial Intelligence
    - Scientific Computing
    - Robotics
    - Databases, Data Mining
    - Computational Linguistics, Natural Language Processing ...
- Computer Engineering
  - Overlap with CS and EE; emphasizes hardware

What is Programming?

- **program**: A set of instructionsto be carried out by a computer.

- **program execution**: The act of carrying out the instructions contained in a program.

- **programming language**: A systematic set of rules used to describe computations in a format that is editable by humans.
  - This course teaches programming in a language named Java.

What is Programming?

"A process that leads from an original formulation of a computing problem to executable computer programs"

https://en.wikipedia.org/wiki/Computer_programming

"The process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer"

http://interactivepython.org/runestone/static/pythonds/Introduction/introduction.html

Required Skill

- Problem solving
- System thinking
  - The art and science of making reliable inferences about behavior by developing an increasingly deep understanding of underlying structure.
- Programming

Richmond, Barry. "System dynamics/systems thinking: Let's just get on with it." *International Systems Dynamics Conference, Sterling, Scotland*. 1994.

ALGORITHM

Algorithm

- The way (how) to solve the problem is called “Algorithm”
- An **algorithm** is a procedure or formula for solving a problem or a list of steps for solving a problem.
- A problem can have several ways (algorithms) to solve

http://whatis.techtarget.com/definition/algorithm

Algorithms

- Example algorithm: "Bake sugar cookies"
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.
  - Spread frosting and sprinkles onto the cookies.
  - ...

Problems with algorithms

- *lack of structure*: Many tiny steps; tough to remember.
- *redundancy*: Consider making a double batch...
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven temperature.
  - Set the timer.
  - Place the first batch of cookies into the oven.
  - Allow the cookies to bake.
  - Set the timer.
  - Place the second batch of cookies into the oven.
  - Allow the cookies to bake.
  - Mix ingredients for frosting.
  - ...

Structured algorithms

- **structured algorithm**: Split into coherent tasks.

  1 Make the cookie batter.
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.

  2 Bake the cookies.
  - Set the oven temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.

  3 Add frosting and sprinkles.
  - Mix the ingredients for the frosting.
  - Spread frosting and sprinkles onto the cookies.

  ...

Removing redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy.

1 Make the cookie batter.

- Mix the dry ingredients.
- ...

2a Bake the cookies (first batch).

- Set the oven temperature.
- Set the timer.
- ...

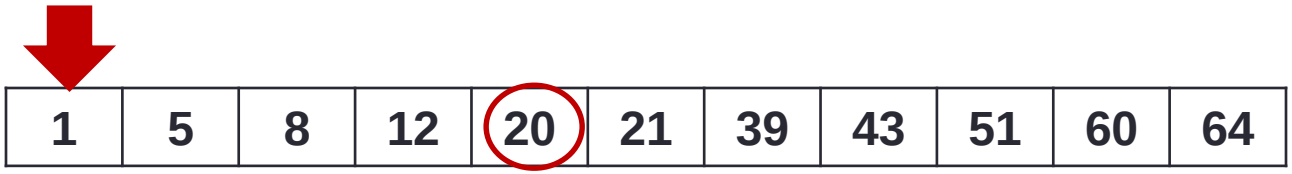
2b Bake the cookies (second batch).

3 Decorate the cookies.

- ...

Example Search Algorithm

- Linear search
  - Find 20 in the sorted list

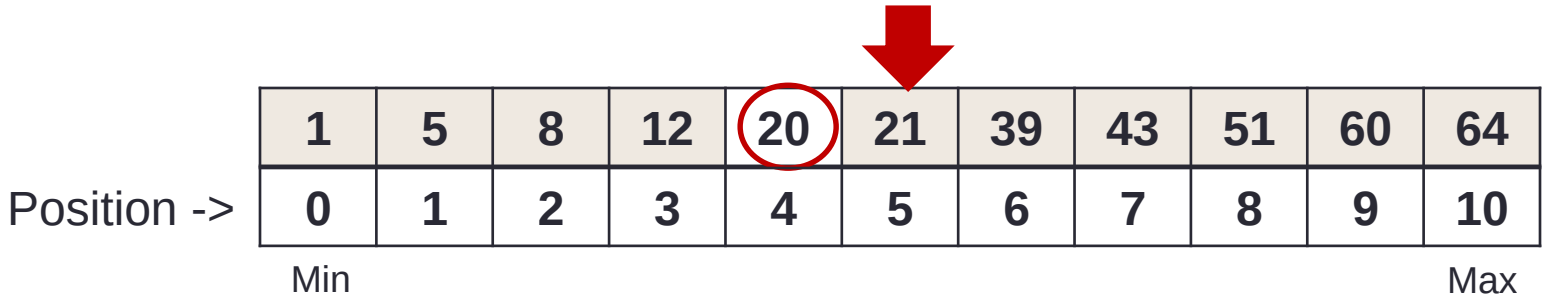



```
LinearSearch(value, list)
        if the list is empty, return -1;
        else
                if the first item of the list has the
desired value, return its location;
                else return LinearSearch(value,
remainder of the list)
```

Set *i* to 1.
Repeat this loop:
    If *i* > *n*, then exit the loop.
    If *A*[*i*] = *x*, then exit the loop.
    Set *i* to *i* + 1.
Return *i*.

Example Search Algorithm

- Binary search
  - Find 20 in the sorted list



1. Let *min* = 0 and *max* = *n*-1.
2. If *max* < *min*, then stop: *target* is not present in array. Return *-1*.
3. Compute *guess* as the average of *max* and *min*, rounded down (so that it is an integer).
4. If array[*guess*] equals *target*, then stop. You found it! Return *guess*.
5. If the *guess* was too low, that is, array[*guess*] < *target*, then set *min* = *guess* + 1.
6. Otherwise, the *guess* was too high. Set *max* = *guess* - 1.
7. Go back to step 3.

Exercise 1.1

- Find sum of

$$1+2+3+4+5+6+\ldots\ldots+n$$

How do you do it (step by step)?

- Iteratively adding number 1 until n
- Using formula n(n+1)/2

EXPRESS ALGORITHM

Express Algorithm

- Programming – writing instructions for computer

- Before programing you can sketch your idea (algorithm)
  - Common (simple) ways to express algorithm
    - Pseudocode
    - Flow chart

Pseudocode

- An informal language to describe algorithm
  - Text-based
  - No strict syntax
  - Commonly uses simple word (in natural language)
  - Structural convention
  - Intended for human read

Pseudocode

- Examples
  - Class scoring

1. Set *stid* = random a student id;
2. If student[*stid*] attends the class but cannot answer the question correctly
   score of student[*stid*] – 1;
3. Else if student[stid] attends the class but doesn't answer the question
   score of student[*stid*] – 2;
4. Else if student[stid] doesn't show up
   score of student[*stid*] – 3;

Pseudocode

- Examples

Linear search

```
LinearSearch(value, list)
    if the list is empty, return -1;
    else
        if the first item of the list has the
desired value, return its location;
        else return LinearSearch(value,
remainder of the list)
```

```
Set i to 1.
Repeat this loop:
    If i > n, then exit the loop.
    If A[i] = x, then exit the loop.
    Set i to i + 1.
Return i.
```

Binary search

1. Let *min* = 0 and *max* = *n*-1.
2. If *max* < *min*, then stop: *target* is not present in array. Return -1.
3. Compute *guess* as the average of *max* and *min*, rounded down (so that it is an integer).
4. If array[*guess*] equals *target*, then stop. You found it! Return *guess*.
5. If the *guess* was too low, that is, array[*guess*] < *target*, then set *min* = *guess* + 1.
6. Otherwise, the *guess* was too high. Set *max* = *guess* - 1.
7. Go back to step 3.

Exercise 1.2

- Write your pseudocode for finding the sum of

  1+2+3+4+5+6+…….+n

```
Set result = 0
Set i = 1
While i < n+1
        result = result + i
        i = i  + 1
Print result
```

```
Set result = n(n+1)/2
Print result
```

Flow Chart

- A formalized graphic representation of a logic sequence of work
- One of the oldest modeling language
- Flowchart uses simple geometric symbols and arrows to define tasks and relationships

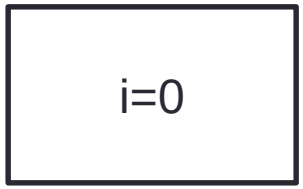
Flow Chart

- An oval denotes the beginning or end of a program

Start program

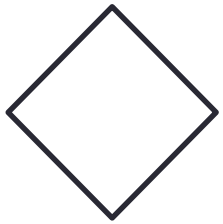
End program

- Rectangle indicates the assignment of a value. The computation is also included

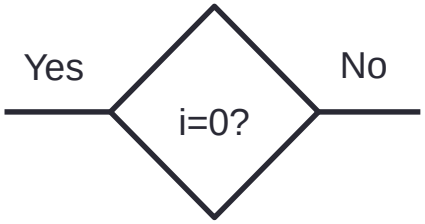


Assign value 0 to variable i

Flow Chart



- A diamond indicates point where a decision is made

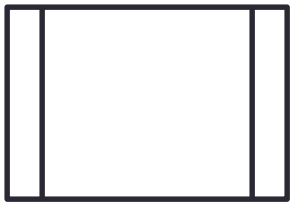


Check if i = 0

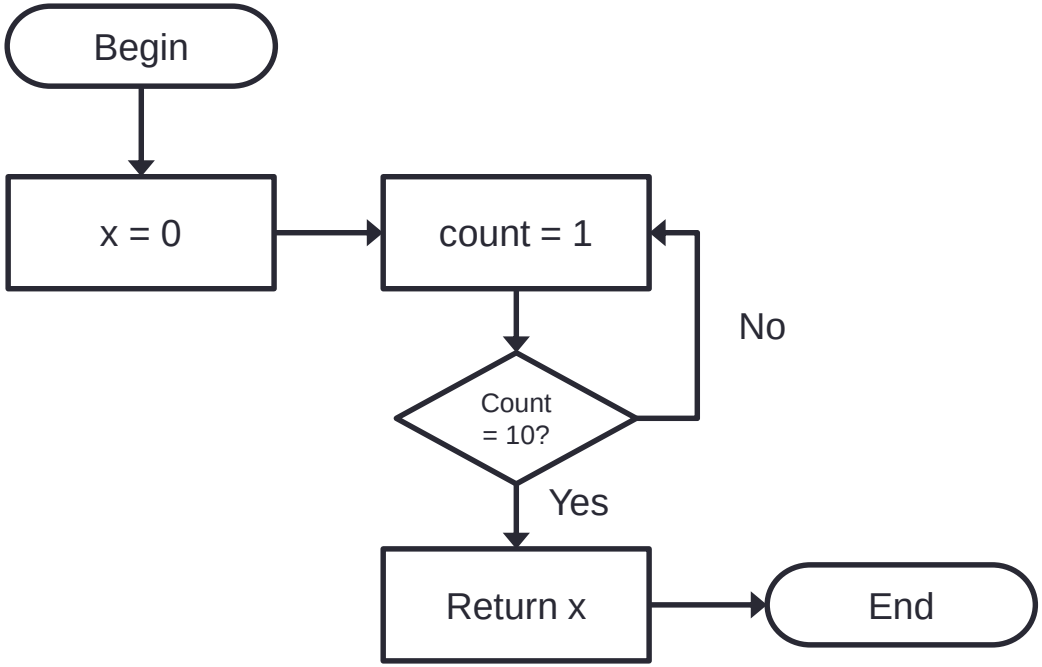
- A parallelogram is a point where there is input to or output from the program

Print value of variable i

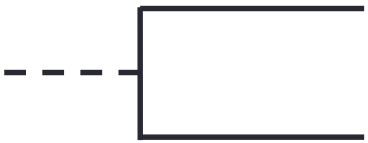
Flow Chart



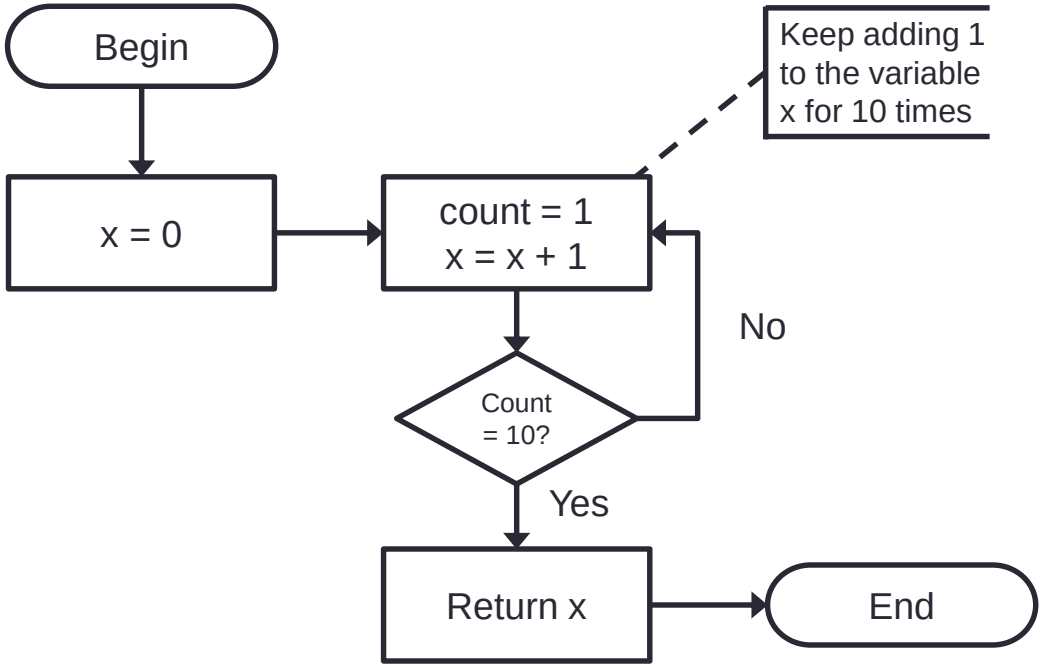
- The double-lined rectangle indicates the use of an algorithm specified out side the program, such as a subroutine



Flow Chart

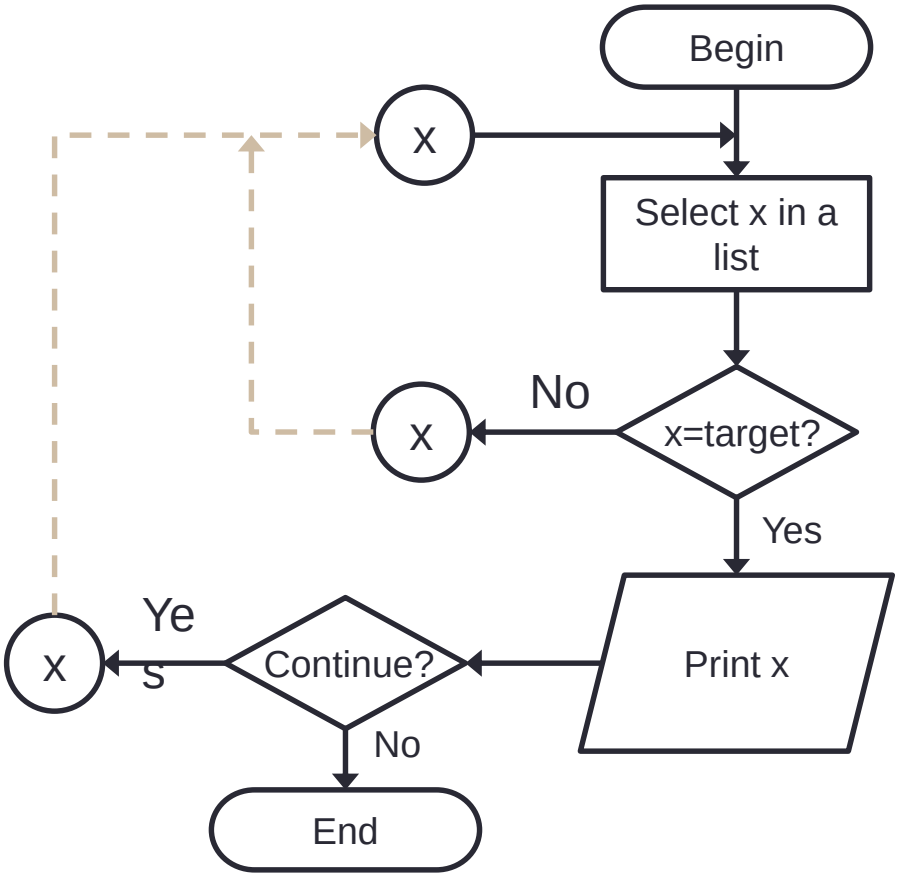


- An open-ended rectangle contains comment statements connected the flow via dashes line

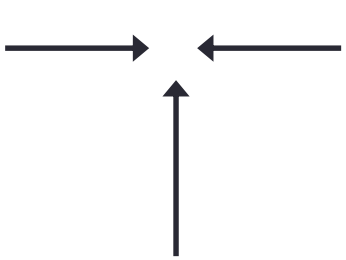


Flow Chart

- Circle can be used to combine flow lines



Flow Chart

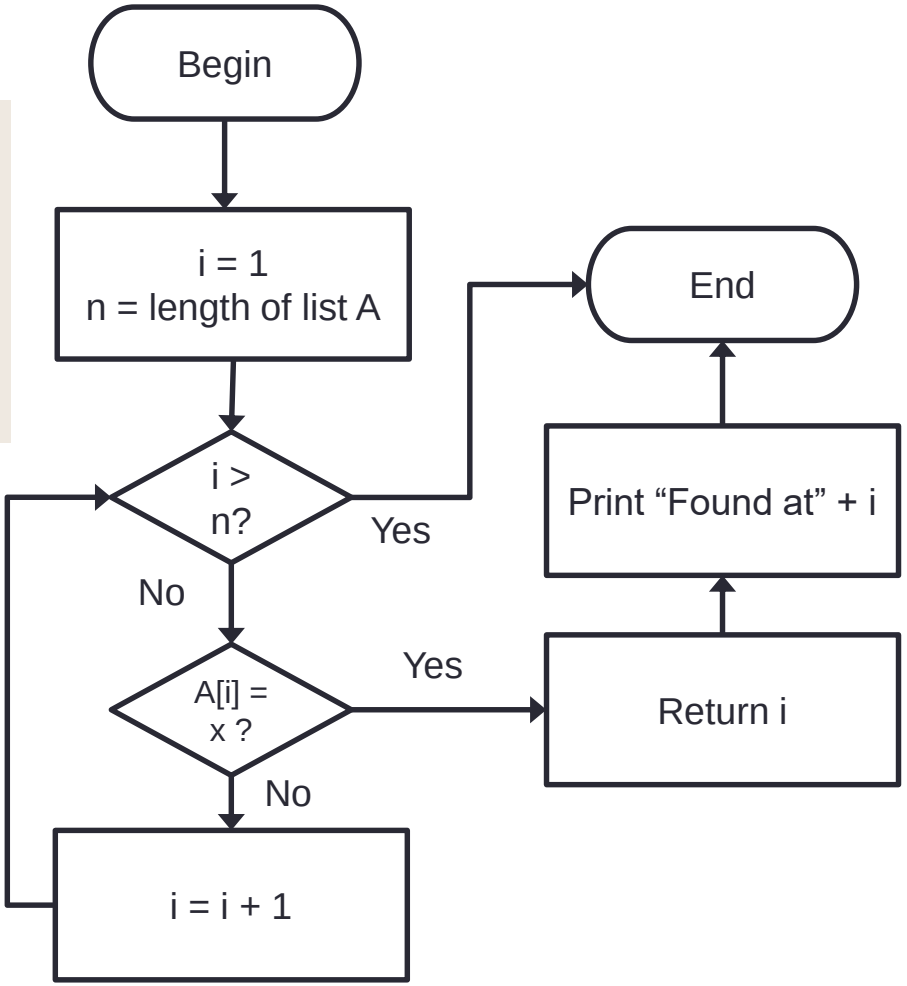


- Arrows indicates the direction and order of program execution
- Standard
  - Left to right 
  - Right to left 
  - Top to bottom 
  - Bottom to top 

Flow Chart Example

- Linear search

Set *i* to 1.
Repeat this loop:
    If *i* > *n*, then exit the loop.
    If *A*[*i*] = *x*, then exit the loop.
    Set *i* to *i* + 1.
Return *i*.

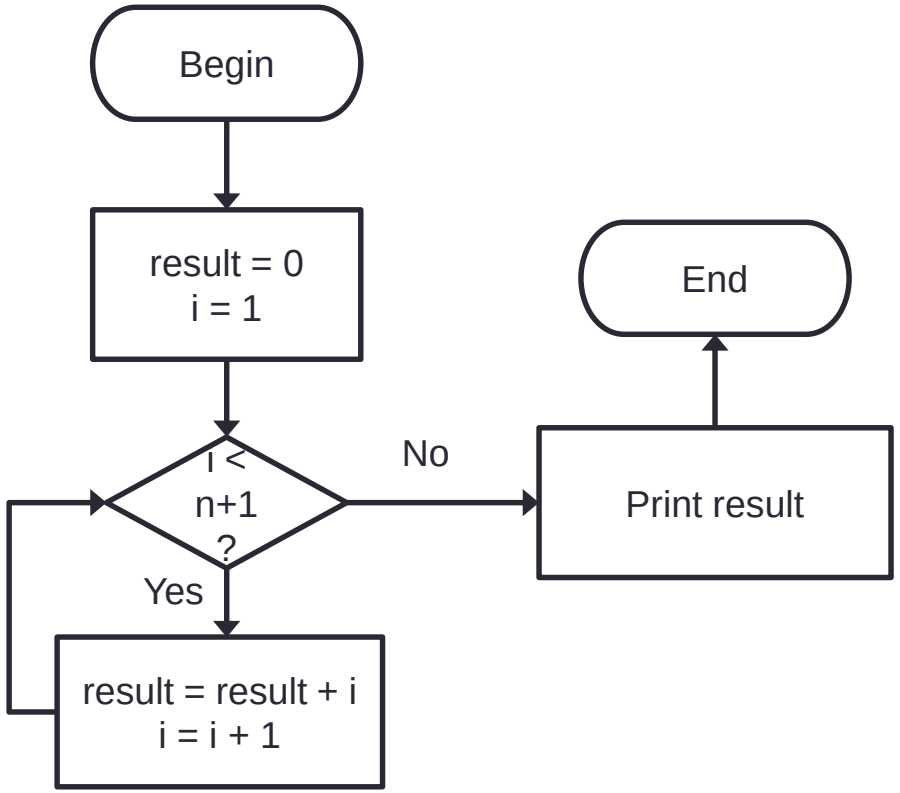


Exercise 1.3

- Draw a flow chart of your algorithm for finding

$$1+2+3+4+5+6+\ldots\ldots+n$$

```
Set result = 0
Set i = 1
While i < n+1
        result = result + i
        i = i  + 1
Print result
```



Exercise 1.4

- Draw a flow chart to calculate a grade according to the following conditions:
  - If the score is below 50 → F
  - 50-54 → D
  - 55-59 → D+
  - 60-64 → C
  - 65-69 → C+
  - 70-74 → B
  - 75-79 → B+
  - More than 80 → A

Express it as Programming Language

- Java

```java
public class Adder {
    public static void main(String[] args){
        int n = 10;
        int result = 0;
        int i = 1;
        while(i<n+1){
            result = result + i;
            i = i + 1;
        }
        System.out.println(result);
    }
}
```

Express it as Programming Language

- Python

test ×

```
n = 10
i = 1
result = 0
while (i < n+1):
    result = result + i
    i = i + 1
print (result)
```

Express it as Programming Language

- C

```c
#include <stdio.h>

void main(void) {
    int n = 10;
    int i = 1;
    int result = 0;
    while( i < n+1 )
    {
      result = result + i;
      i = i +1;
    }
    printf("%d\n", result);

}
```

Exercise 1.5

- What does this algorithm do?
  - Print Fibonacci number of the n^{th}
- What does it print if $n = 2$?
  - 1
- What does it print if $n = 5$?
  - 3
- What does it print if $n = 7$?
  - 8

```
Set result = 1
Set temp1 = 0
Set temp2 = 1
Set count = 3
If n = 1
        result = temp1
Else if n = 2
        result = temp2
Else
        While(count  < n+1)
                result = temp1 + temp2
                temp1 = temp2
                temp2 = result
                count = count + 1
Print result
```