CS221: Computer Programming I

B.Sc in Computer Science

Autumn 2011

Introduction to programming & problem solving

- Transfer of knowledge
- Example of a C++ program
- Structure of a C++ program
- Algorithms and programs

Transfer of knowledge

Example of a C++ program (1)

```
#include <iostream.h>
#include <conio.h>

    int main()
    { int x, n, m, i;
      cout << "Enter size > ";
      cin >> n;
      cout << "Enter first element >";
      cin >> x;
      m  = x;
      i = 2;
      while (i <= n)
            {
                cout << "Enter next element > ";
                cin >> x;
                if (x < m) m = x;
                i = i + 1;
            }
      cout  << m;
    }
```

Example of a C++ program (2)

```
#include <iostream.h>
#include <conio.h>

void main()
 { float x, avg, sum;
   int n, i;
   clrscr();
   cout << "Enter the number of data > ";
   cin >> n;
   sum = 0;
   for (i=1; i<=n; i++) {
     cin >> x;
     sum = sum + x;
   }
   avg = sum/n;
   cout << "The average is " << avg;
 }
```

Example of a C++ program (3)

```
#include <iostream>
#include <conio.h>
using namespace std;

void inpData(float a[100], int & n)
   {
     cout << "Enter the number of data in the list > ";
     cin >> n;
     cout << "Enter the elements of the list one by one \n\n";
     for (int i=0; i<=n-1; i++)
         cin >> a[i];
   }

float searchMin(float a[100], int n)
   { int i;
     float min;
     min = a[0];
     i=2;
     while (i<=n-1) {
          if (a[i] < min) min = a[i];
          i++;
     }
     return min;
   }

int main ()
{ int n;
  float a[100], min;
   inpData(a, n);
   min = searchMin(a, n);
   cout << "Minimum is " << min << endl;
   getch();
   return 0;
}
```

What is an algorithm?

An algorithm is a precise, step-by-step <u>sequence of instructions</u> (steps) accomplishing a well defined task.

Algorithmic Thinking

Algorithmic thinking is the ability to ***understand, execute***, ***evaluate***, and ***create*** algorithms

Preparing a Shiro Wat

1. i”Ÿ<`ƒ ÃŸ}óM
2. i”Ÿ<`ƒ Éeƒ ¨<eØ ÃÚS`“ ¨<H¨<” SÖØ eŸ=ÁÅ`Ó ÃueLM
3. ²Ãƒ ÃÚS`“ ›wa ÃÖudM
4. u`u_ ÃÚS`“ ›wa ÃlLLM
5. ¨<H ÃŸKdM
6. ¨<H c=ðL ia ÃÚS^M
7. ia¨< ŸucK u%EL Ú¨< ÃÚS^M

What can be done with the recipe?

- It is documented for future “preservation”
  - It can be reproduced
- It can be shared to others: Knowledge sharing
- It maybe automated “one day”

Properties of a good Algorithm

- Each instruction/step must be action oriented – (must be an order/<u>executable</u>)
- Each instruction/step must be expressed <u>un-ambiguously</u>
- Each instruction shall be as “reasonably atomic” as possible
- The instructions shall be <u>sequenced</u>
- The algorithm shall come to an end (finite number of executions)

Preparing a Shiro Wat

1. ¨<H ÃŸKdM
2. ¨<H c=ðL ia ÃS)M/ÃÚS^M
3. ia¨< ŸucK u%EL Ú¨< ÃÚS^M
4. i”Ÿ<`ƒ ÃŸ}óM
5. i”Ÿ<`ƒ Éeƒ ¨<eØ ÃÚS`“ ¨<H¨<” SÖØ eŸ=ÁÅ`Ó ÃueLM
6. ²Ãƒ ÃÚS`“ ›wa ÃÖudM
7. u`u_ ÃÚS`“ ›wa ÃlLLM

The power of expression

- “You do not really understand something unless you can explain it to your grandmother.“, Albert Einstein

Income tax calculation

- get salary
- if salary is below 150 then tax = 0
- if salary is between 151 & 650 then tax = (salary - 150)*0.1;
- if salary is between 651 & 1400 then tax = 50 + (salary - 650)*0.15;
- if salary is between 1401 & 2350 then
  tax = 50+112.50+(salary - 1400)*0.2;
- if salary is between 2351 & 3550 then
- tax = 50+112.50+190+(salary - 2350)*0.25;
- if salary is between 3551 & 5000 then
- tax = 50+112.50+190+300+(salary - 3550)*0.3;
- if salary is above 5000
  tax = 50+112.50+190+300+435+(salary - 5000)*0.35;
- Report tax
- Report (salary – tax) as Net income

Algorithm is made of

- Algorithm = instruction (code) + data
- Eg1: preparation of shiro wat
  - Data: “shinkurt”, “shiro”, “berbere”, “chew”
  - Instruction: “mechemer”, “makulalat”, …
- Eg2: Income tax calculation
  - Instruction: computing the deduction
  - Data: salary, income tax, net income

What is accomplished?

- START
- Let n be the number of elements in a list of numbers
- Let x be first element of the list
- Let min x
- Let i 2

* - Let next element of the list is x

- If x < min then min ← x
- i i + 1
- If i <= n then go back & start execution from (*)
- Display min
- STOP

What is accomplished?

Receive two numbers x & y

Let z be 0

As long as x > 0 repeat the following {

if *x* is odd *z* = *z* + *y*

x = *x*/2 // without the fraction

y = *y* + *y*

}

Display z

Structure of an algorithm

- (STARTing & STOPping point)
- Input and Output statements
- Variables and values
- Instructions
  - Sequences
  - Selections
  - Repetitions
- Abstraction Mechanisms
  - Functions, methods, objects

Documenting algorithms

- Plain English, Amharic, etc
- Flowchart
- Pseudo-code (structured English):

Plain English, Amharic, etc

- Natural languages could be used to describe algorithms

- Eg: Searching for the minimum

Take the first number as the minimum and then continue to compare the current minimum with the rest of the list until the list is exhausted

Plain English, Amharic, etc

- Take the first number as the minimum
- If the next number is less than the minimum, take this new number as “a new” minimum
- Continue to compare the minimum with the rest until all the numbers of the list are processed
- Report the minimum

Plain English, Amharic, etc

- Easy to use & understand
- But
  - Ambiguous
  - Data are not adequately represented
  - No obvious structure except the sequence
- Use to describe “seed” algorithm that will be refined using other tools

Exercise

- Describe in Plain English the algorithm that searches both for the minimum and the maximum number in a list of numbers
- Describe an algorithm that counts for the number of negatives from a list of numbers
- Describe an algorithm that checks whether a year is a leap year or not

Leap Year

If a year is a century, then it must be divisible by 400 to be a leap year but otherwise it is enough that it is divisible by 4 to be a leap year.

Leap Year

- If year is a century and is divisible by 400, then it is a leap year
- If year is not a century and is divisible by 4, then it is a leap year
- In all other cases, it is not a leap year

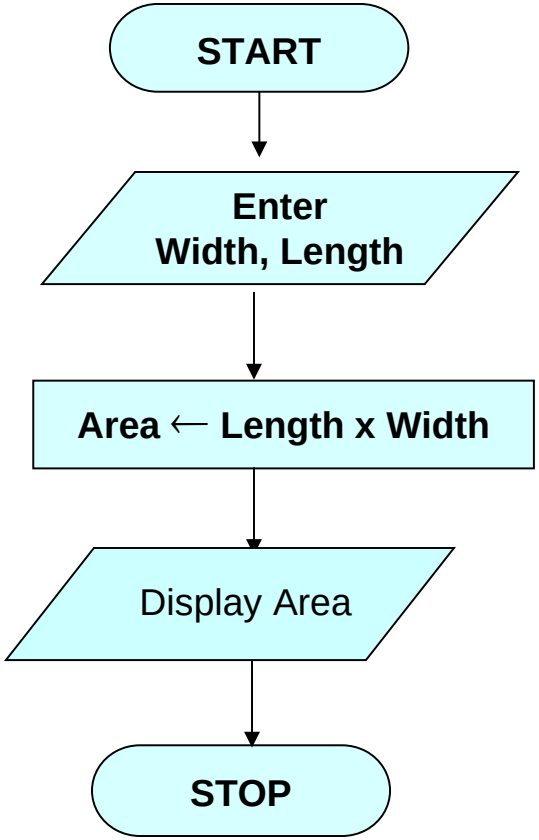
Flowchart

A diagrammatic representation of a sequence of operations, as in a manufacturing process.

Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- control flow from one action to the next

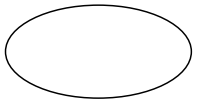
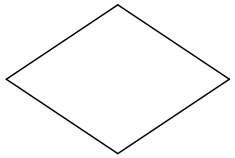
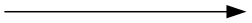
Example

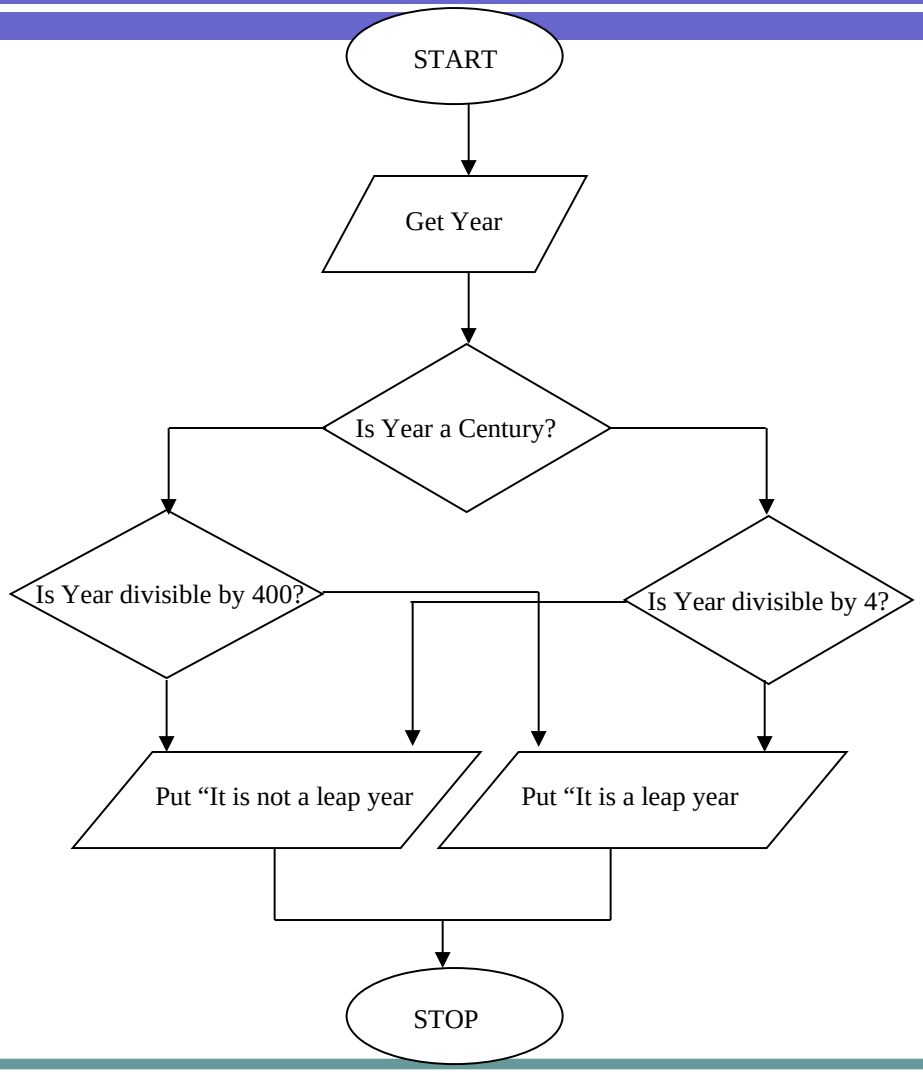

START
Enter
Width, Length
Area ← Length x Width
Display Area
STOP

Four category of instructions

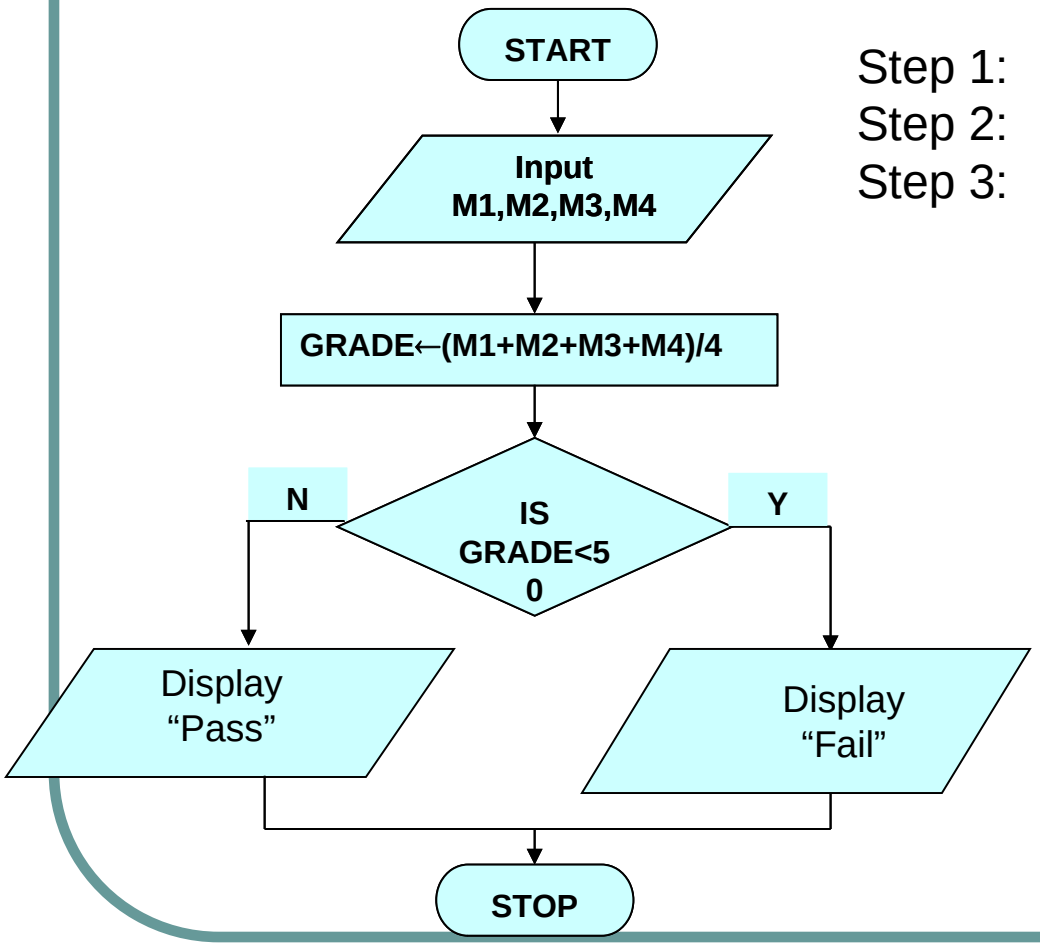
- I/O
- Assignment/Process
- Selection/Branching
- Repetition/Looping

Flowchart Symbols

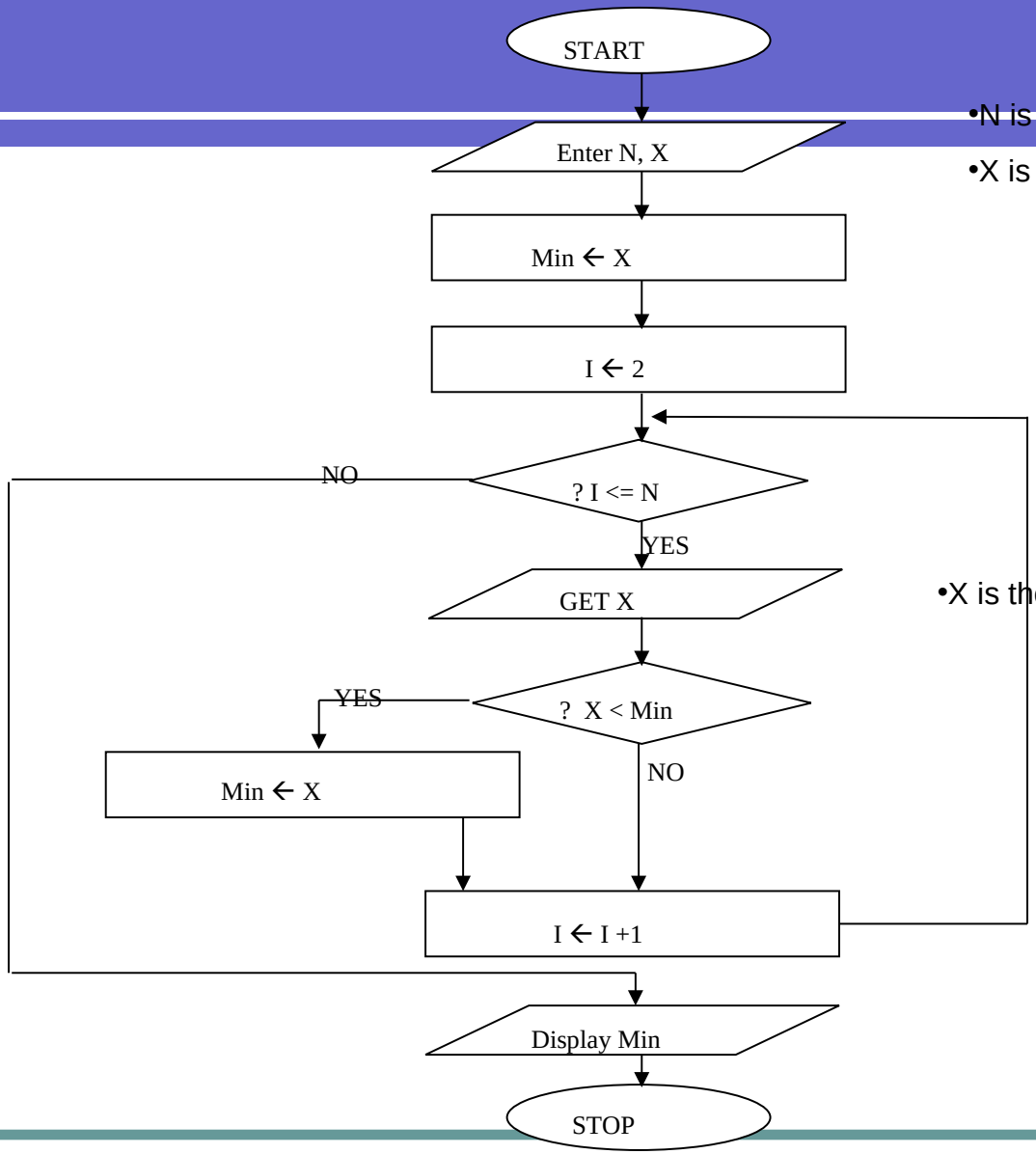
Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input/Output operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Flow line		Denotes the direction of logic flow in the program


START
Get Year
Is Year a Century?
Is Year divisible by 400?
Is Year divisible by 4?
Put “It is not a leap year
Put “It is a leap year
STOP

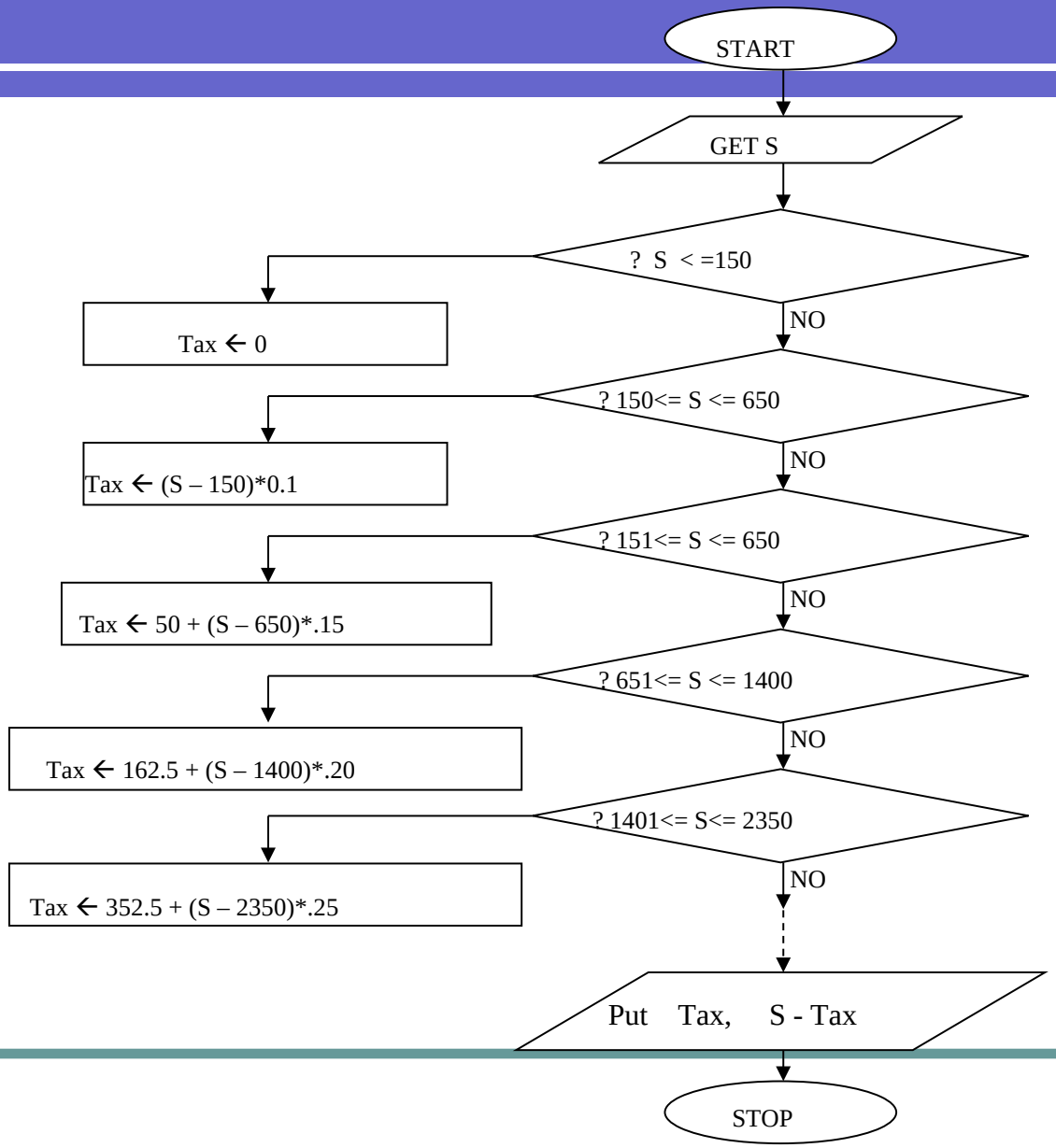
Example

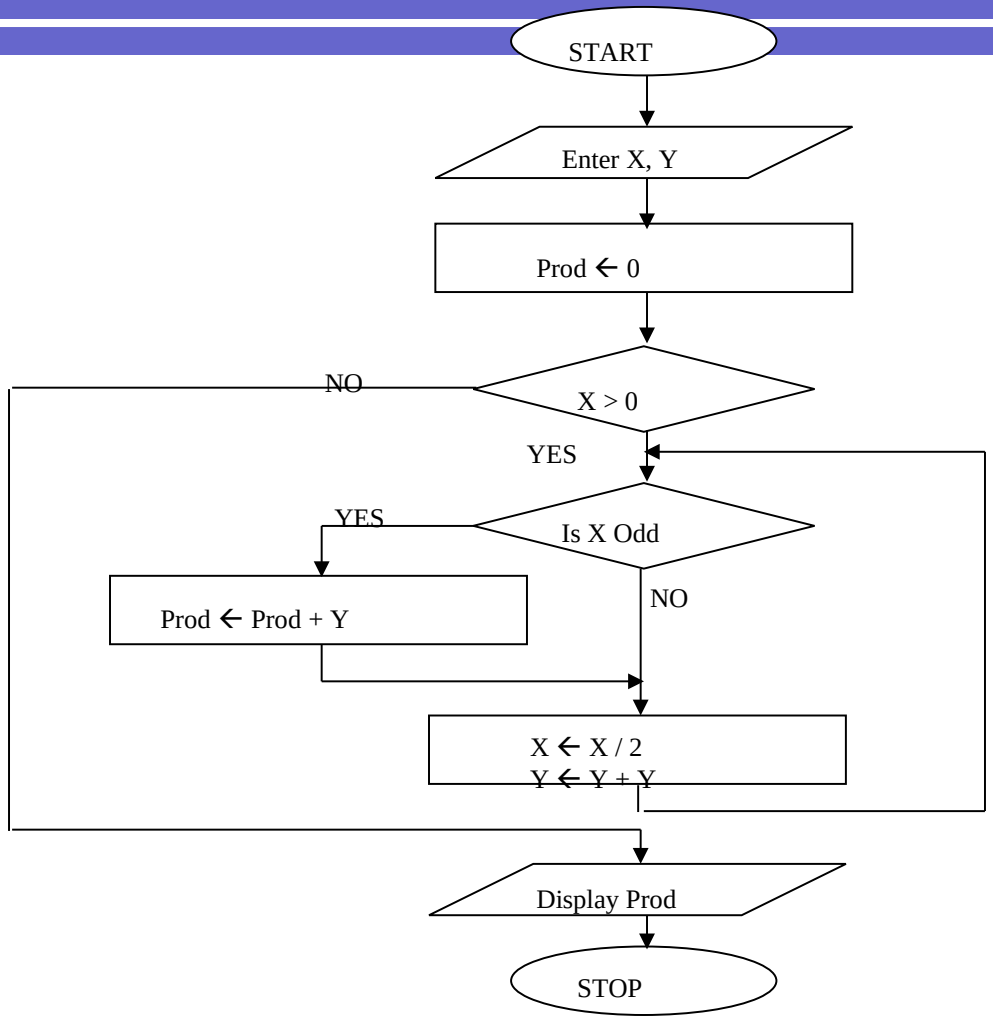


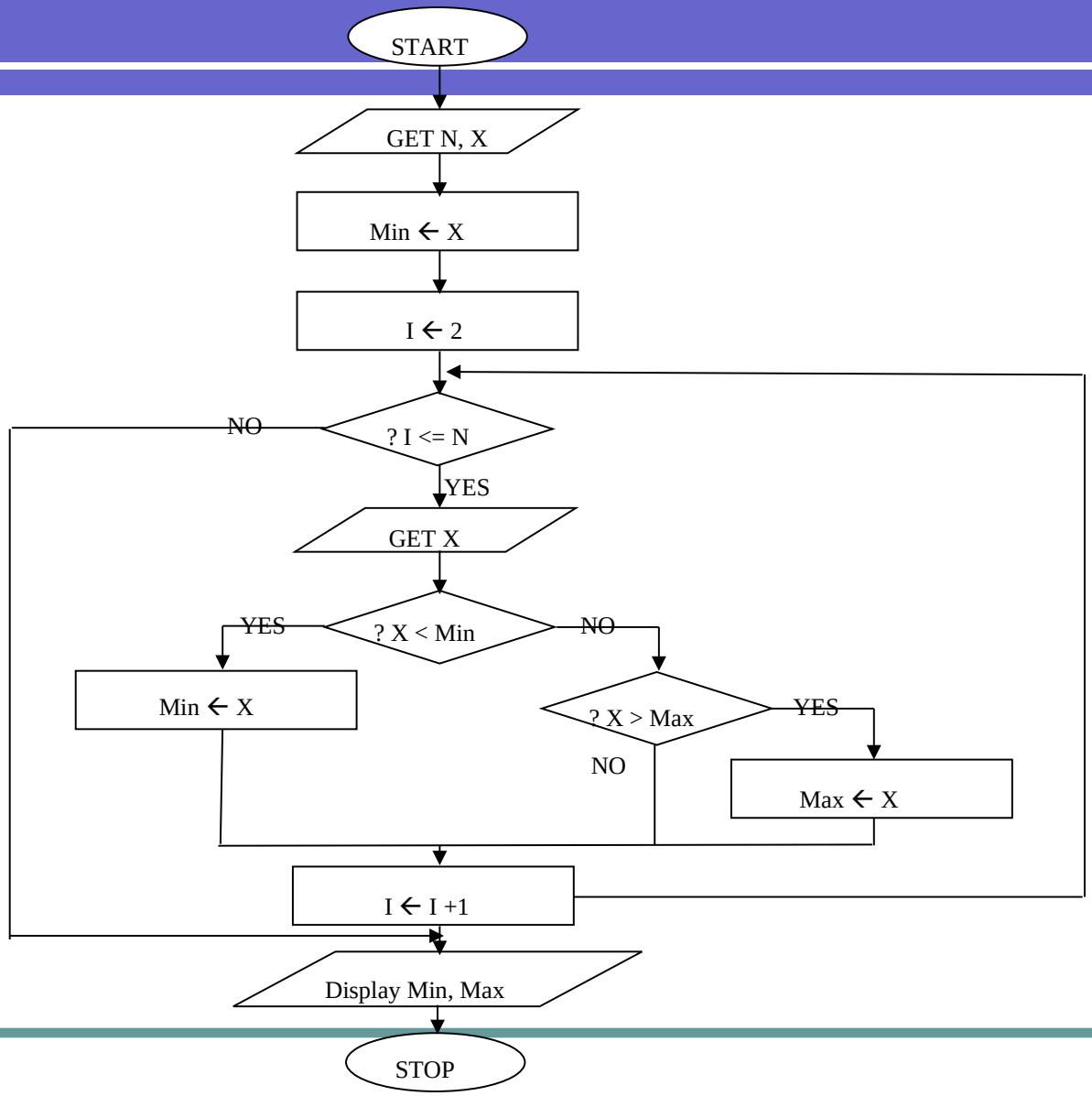
Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE <50) then
Display “FAIL”
else
Display “PASS”

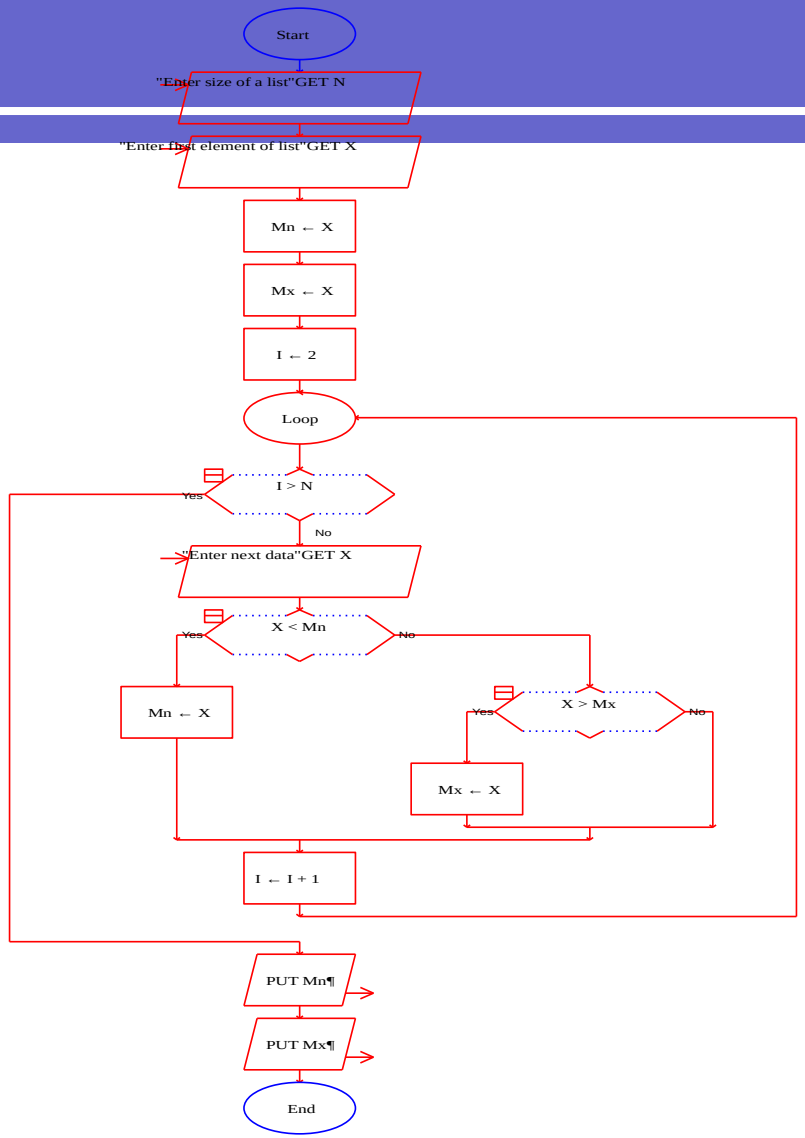


- N is the size of the list
- X is the first element of the list
- X is the next element of the list


START
GET S
? S < =150
Tax ← 0
NO
? 150<= S <= 650
Tax ← (S – 150)*0.1
NO
? 151<= S <= 650
Tax ← 50 + (S – 650)*.15
NO
? 651<= S <= 1400
Tax ← 162.5 + (S – 1400)*.20
NO
? 1401<= S<= 2350
Tax ← 352.5 + (S – 2350)*.25
NO
Put Tax, S - Tax
STOP


START
Enter X, Y
Prod ← 0
X > 0
NO
YES
Is X Odd
YES
NO
Prod ← Prod + Y
X ← X / 2
Y ← Y + Y
Display Prod
STOP


START
GET N, X
Min ← X
I ← 2
? I <= N
NO
YES
GET X
? X < Min
YES
NO
Min ← X
? X > Max
YES
NO
Max ← X
I ← I +1
Display Min, Max
STOP


Start
"Enter size of a list"GET N
"Enter first element of list"GET X
Mn ← X
Mx ← X
I ← 2
Loop
I > N
Yes
No
"Enter next data"GET X
X < Mn
Yes
No
Mn ← X
X > Mx
Yes
No
Mx ← X
I ← I + 1
PUT Mn¶
PUT Mx¶
End

Pseudo-code

- Also called Structured English
- A programming language like constructs
  - If else, case, while, for
- Data represented using variables
- No rigorous grammar➔ gives freedom to think
- Better than English in precision

Data representation

- Data can be numbers, texts or other more complex structures
- Variables such as I, j, x, y, z, p, q …
- Or more readable variables such as size, height, age, numberOfStudents. Etc
- List can be represented as $a_1, a_2, a_3, \ldots a_n$
- A list may contain numbers, texts, etc

Four category of instructions

- I/O
- Assignment/Process
- Selection/Branching
- Repetition/Looping

I/O

- Input: when a data comes from the “outside”
  - Enter the size of the list into x
  - Let x be the size of the list
  - Get the size of the list into x
  - Receive two numbers x & y
  - Read two numbers x & y

I/O

- Output: when a data should go “outside”
  - Display X
  - Return Y
  - Put Y+X/Z
  - Display “The minimum is” Min
  - Write Min Max

Assignment

- When data is generated from within
  - X ← 10
  - X = 10 // this is not usually recommended
  - Let X be 10
  - Assign 10 to X
  - X ← X + 1
  - Min ← X
  - Z ← Z + Y

Selection/Branching/Decision

- When a decision need to be taken in order to choose an action among two or more alternatives
  - Example
    - income tax calculation: if salary is between …
    - searching for the min: if x < min …

Selection/Branching/Decision

- Two varieties:
  1. Do *Action* on condition(s)
     - If <condition(s)> then <action>
  2. Mutual exclusion: Do *Action1* or *Action2* but not both
     - If <condition> then <action1>

       Else <action2>

Specialized action/function

- Relational operators: <, <=, >, >=, ==, !=
- Arithmetic operators: +, -, *, /, %
- Logical operators: and, or, not
- etc

Example: Average Marks

- *Input a set of marks*
- *Compute the average by adding and dividing by the number of marks*
- *if average is below 50*

  Display "FAIL"

  else

  Display "PASS"

More refined

- Input M1,M2,M3,M4
- GRADE ← (M1+M2+M3+M4)/4
- if (GRADE < 50) then

  Display “FAIL”

  else

  Display “PASS”

Repetition / Looping

- Some tasks are executed repeatedly
- Most of the time, the repetition is finite; but it can also be “indefinite”
- Two mandatory components:
  - The repeated task(s)
  - The terminating condition

Repetition/Looping

- While <condition> do <action>
- As long as <condition> do <action>
- Etc
- If the <action> is more than one, then it can be enclosed within
  - { } or
  - Begin end

Example: Multiplication

Receive two numbers x & y

Let z be 0

As long as x > 0 {

if *x* is odd *z* = *z* + *y*

x = *x*/2 // without the fraction

y = *y* + *y*

}

Display z

Example: Searching for Min

- START
- Enter the size of the list into n
- Enter the list: a_1, a_2, a_3, … a_n
- min a_1
- Max← a_1
- i 2
- While (i<=n) do
  - {
    - If a_i < min then min ← a_i
    - Else if (a_i > max) then max ← a_i
    - i ← i + 1
  - }
- Display min, max
- STOP

Problem solving by computers

- Step1: Understanding of the problem
- Step2: Algorithm Development/Design
- Step3: Coding/Program development
- Step4: Testing
- Step5: Maintenance