Formal Definition of Algorithm

An algorithm is a *well ordered* collection of *unambiguous* and *effectively computable* operations that, when executed, *produce a result*, and *halt in finite time.*

Recall *m*-digit addition:

Input: $m \geq 1$ and two m-digit numbers: $a_{m-1}a_{m-2}\ldots a_2a_1a_0$ and $b_{m-1}b_{m-2}\ldots b_2b_1b_0$

Output: Their sum: $c_m c_{m-1}c_{m-2}\ldots c_2c_1c_0$

```
1)  carry ← 0
2)  i ← 0
3)  while i < m  do 4-10
4)      c_i ← a_i + b_i + carry
5)      if  c_i ≥ 10  do 6-7
6)          c_i ← c_i − 10
7)          carry ← 1
8)      else do 9
9)          carry ← 0
10)     i ← i+1
11) c_m ← carry
12) print  c_m c_{m-1} c_{m-2} ... c_2 c_1 c_0
13) stop
```

- To say that a collection of instructions is *well ordered* simply means that it is clearly specified which operation to perform first, and when any operation is completed, which operation to perform next.

Example An algorithm which is not well ordered.

1) do something
2) do something
3) do something
4) repeat

Is m-digit addition well ordered?

- An *unambiguous* operation is one that can be carried out directly by the computing agent with no outside assistance or further simplification. Such an operation is also called a *primitive operation* for the computing agent.

What is primitive for one computing agent may not be primitive for another.

m-digit addition assumes that adding three single-digit numbers is a primitive operation.

If 1-digit addition is *not* primitive, how can we specify an algorithm for it?

The level of detail/abstraction necessary to specify an algorithm depends on the capabilities of the computing agent.

Is adding two 10-digit numbers a primitive operation for your pocket calculator?

- An operation is *effectively computable* if it can be accomplished via some computational process. Essentially this means that it is doable by some computing agent.

Example $a \leftarrow \sqrt{b}$

Example list all prime numbers $p_1, p_2, p_3, \ldots\ldots$

- We say an algorithm *produces a result* rather than an answer because sometimes (i.e. for some inputs) there is no answer. In such a case the result would be an error message.

- To say that an algorithm *halts in finite time* means that only finitely many operations will be executed.

Example The following is example is called an infinite loop.

1) do something
2) do something
3) go to (1)

This is why we always include the stop instruction in our algorithms.

It should always be clear how and under what circumstances execution of an algorithm halts.

In summary we can say the Computer Science is the study of *Algorithmic Problem Solving*.

CHAPTER 2 : ALGORITHM DESIGN

PRECISE
EXACT

AMBIGUOUS
ABSTRACT

COMPUTER LANGUAGES

PSEUDO-CODE

NATURAL LANGUAGES

RECALL : TYPES OF OPERATIONS

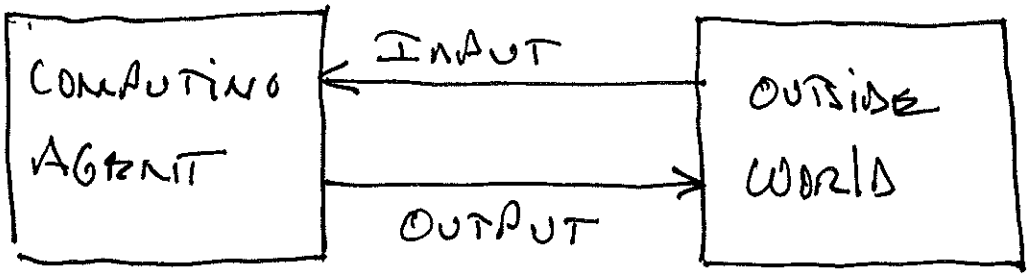
- SEQUENTIAL
- CONDITIONAL (BRANCHING)
- ITERATIVE (LOOPING)

SEQUENTIAL

variable ← expression

EX. $c_i \leftarrow a_i + b_i + carry$

EX. $x \leftarrow \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$

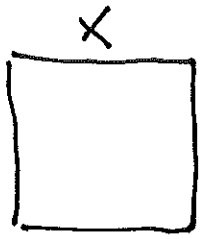


INPUT : get variable

~~Print~~

OUTPUT : Print expression

Ex. get x



Ex. Print x

Ex. Print x+y+7

Ex. Print 'error: division by zero'

Ex. Print 'the value of x is' x

EX. STRAIGHT line ALGORITHM

1.) get a, b, c, d

2.) sum $\leftarrow$ a+b+c+d

3.) average $\leftarrow$ $\frac{\text{sum}}{4}$

4.) Print average

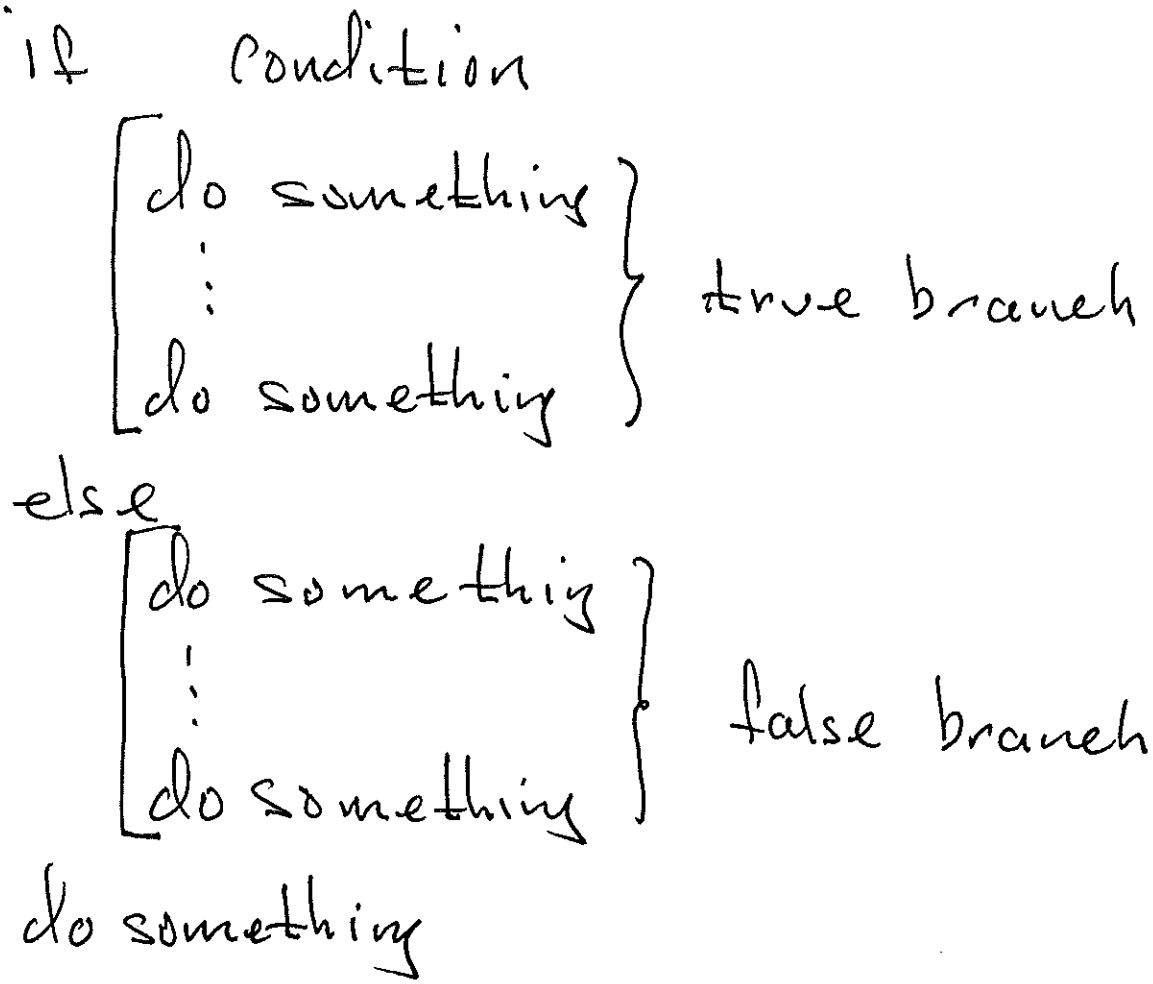
CONDITIONAL OPERATIONS

- if
- if - else

```
if condition
    [do something
     ⋮
     do something]
do something
```

logical expression evaluates to true or false.

'true branch'


if condition
do something
do something
true branch
else
do something
do something
false branch
do something

Ex. 1.) get a, b

2.) if b = 0

3.) Print 'error: division by 0'

4.) else

5.) quotient ← $\frac{a}{b}$

6.) Print quotient

7.) stop

ITERATIVE OPERATIONS

- while
- do-while
- for

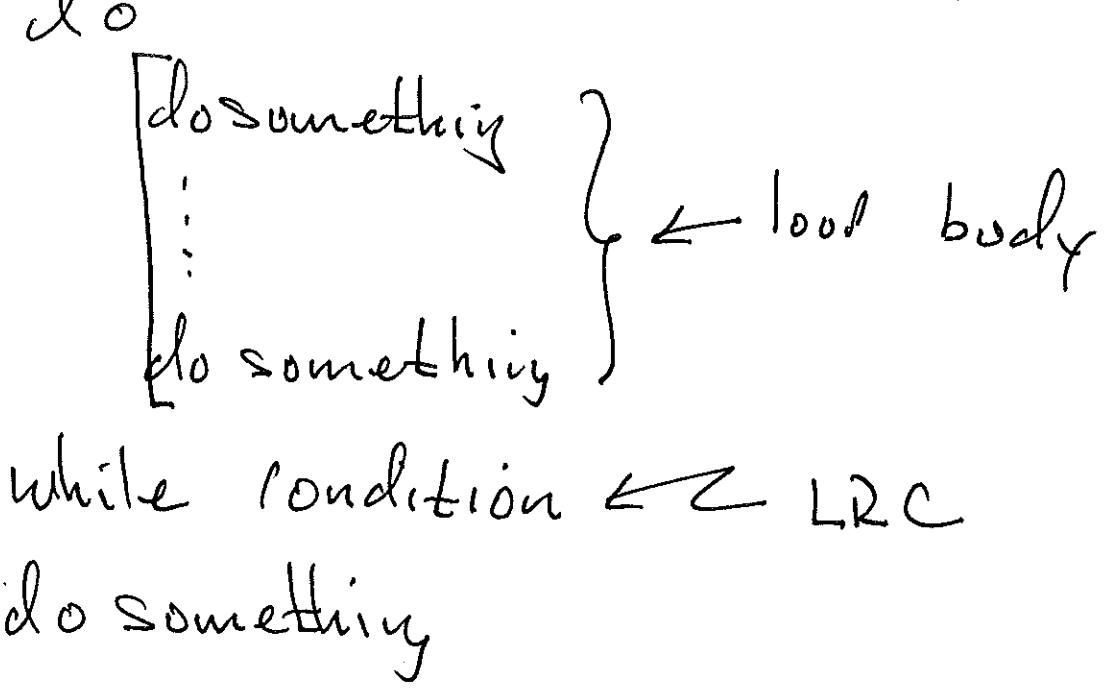
```
while condition
    [ do something
      ⋮
      do something ]
do something
```

loop repetition condition (LRC)

loop body

Ex.

```
1.) response ← 'yes'
2.) while response = 'yes'
3.)     get a, b
4.)     if b=0
5.)         print 'error: div. by 0'
6.)     else
7.)         print a/b
8.)     print 'do you wish to continue?'
9.)     get response
10.) stop
```


do
do something
do something
loop body
while condition
LRC
do something

Ex.

```
1.) do
2.)     get a, b
3.)     if b=0
4.)         print 'error: ....'
5.)     else
6.)         print a/b
7.)     print 'continue ?'
8.)     get response
9.) while response = 'yes'
10.) stop
```