Algorithms and Flowcharts

Algorithms play a vital role in Computer Science. Leaders in the field always remind students that *"Coding is simply translation… The hard work, creativity, and solutions are in the algorithm".*

An algorithm *is a precise description (usually a diagram) that outlines the specific process necessary to solve a problem.*

Algorithms are of more than just a list.

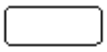
They usually show:

A. ***Sequence*** *of items needed to be done*
B. ***Decisions*** *that need to be made and that lead to different path*
C. ***Repetition*** *of tasks*

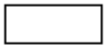
Aside from very simple solutions, the only way to correctly create an **Algorithm** for a computer program is by using flowchart. These flowcharts play a vital role in the communication of solutions to complicated problems

Once the flowchart is drawn, examined and tested, it becomes easy to write the program in *any* high level language. Flowchart algorithms are an essential step for all computer programmers.

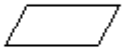
Standard Algorithm Flow Chart Symbols



Start or end of the program



Doing stuff (running motors, displaying stuff, calculating



Input or output operation

Decision making and branching

Connector or joining of two parts of program

We will only use the ones in bold.

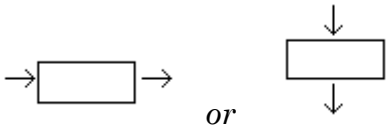
Some Guidelines for Flowchart Algorithms:

1. Must be in a logical order. Linear flow.

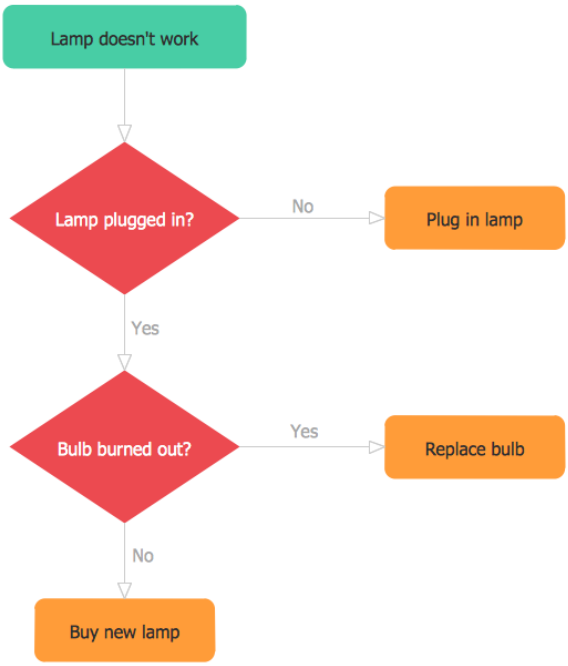
2. The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.

3. The usual direction of the flow of a procedure or system is from left to right or **top to bottom.**

4. Only one flow line should come out from a **process** symbol.



5. A **decision symbol** should have **more than one** *line* coming out of it



6. **Avoid the intersection (crossing) of flow lines.**

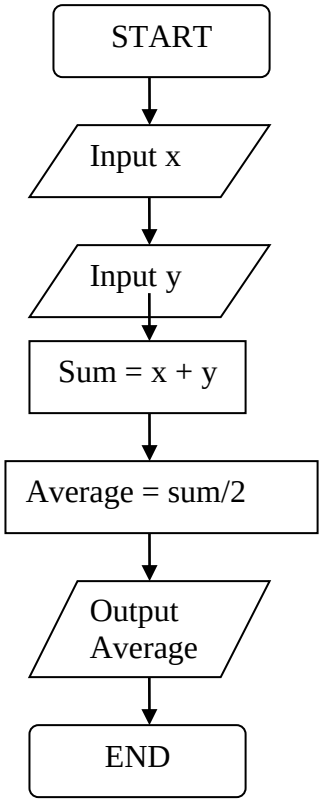
7. Ensure that the flowchart has a *start* and *finish*. ***(not in the case of WHILE (true))***

8. **Test the logic of your flowchart by passing through it with a buddy.**

EXAMPLES:

Example

An algorithm
for finding the average
of two numbers



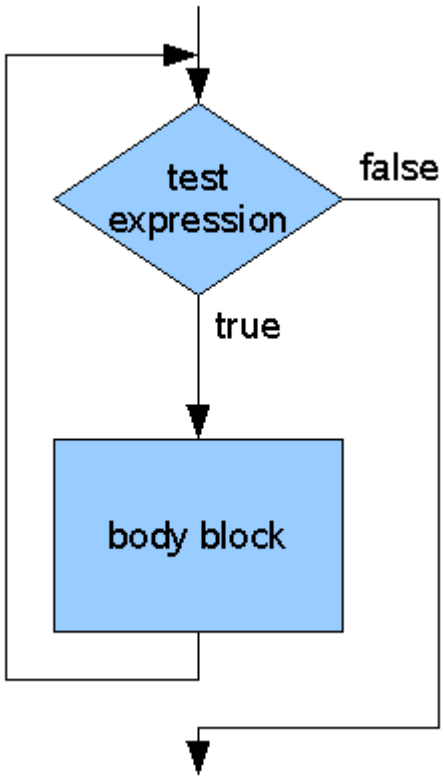
Repeating (Looping) Stuff

Example 2

WHILE LOOP
(Repetition)
The flow-lines show that *if* a specific condition is *true*, the program with do some stuff and then ***"loop back"*** to test the condition again.

The program will be stuck in a repetitive loop until the condition is false

If the condition is false then The program drops out of the loop and on to the **next line** of code.



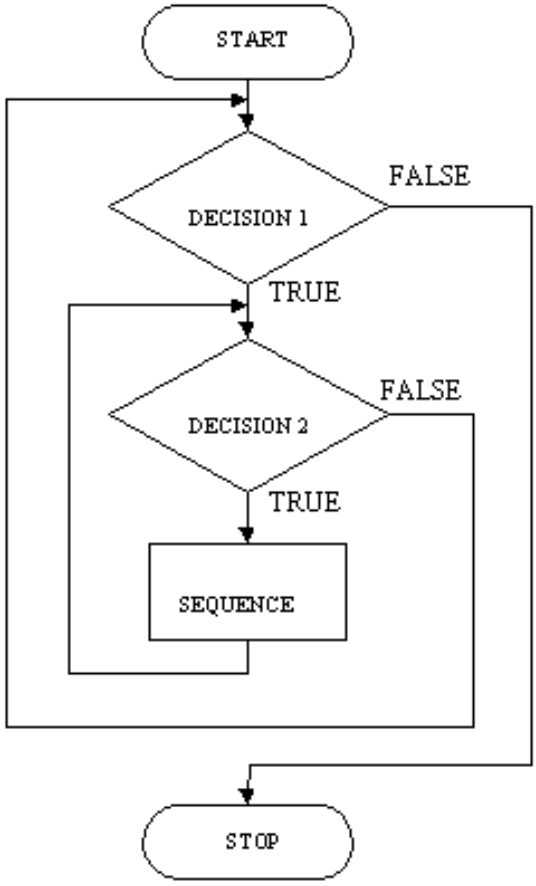
Example 3
(Decision)
NESTED WHILE LOOP

This is one loop that is contained within another loop.

This can be used when you want repeat some code, but only if *TWO* or more conditions are met.

Look carefully at the example diagram.

Tricky…



Decision making (This or That)

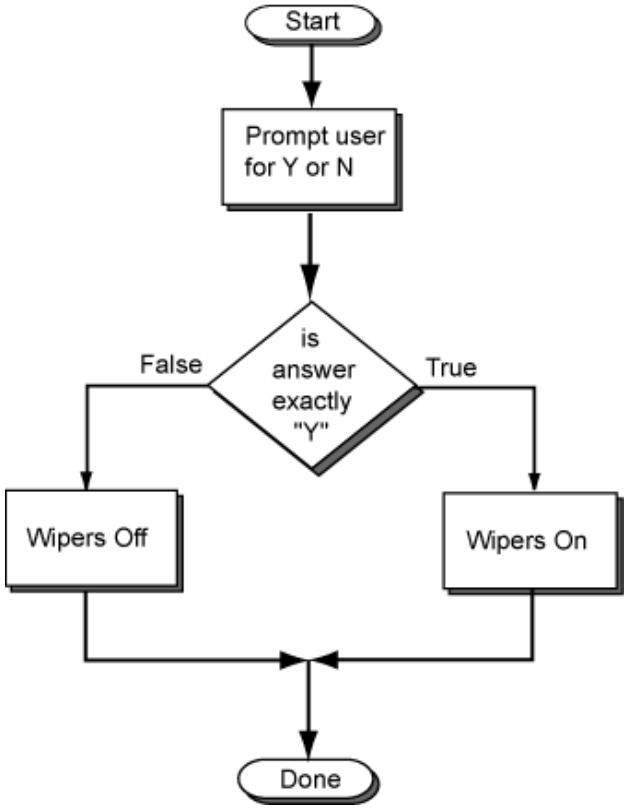
Example 4
(Decision)
If-else STATEMENT

Notice no commands are repeated here.

No looping!

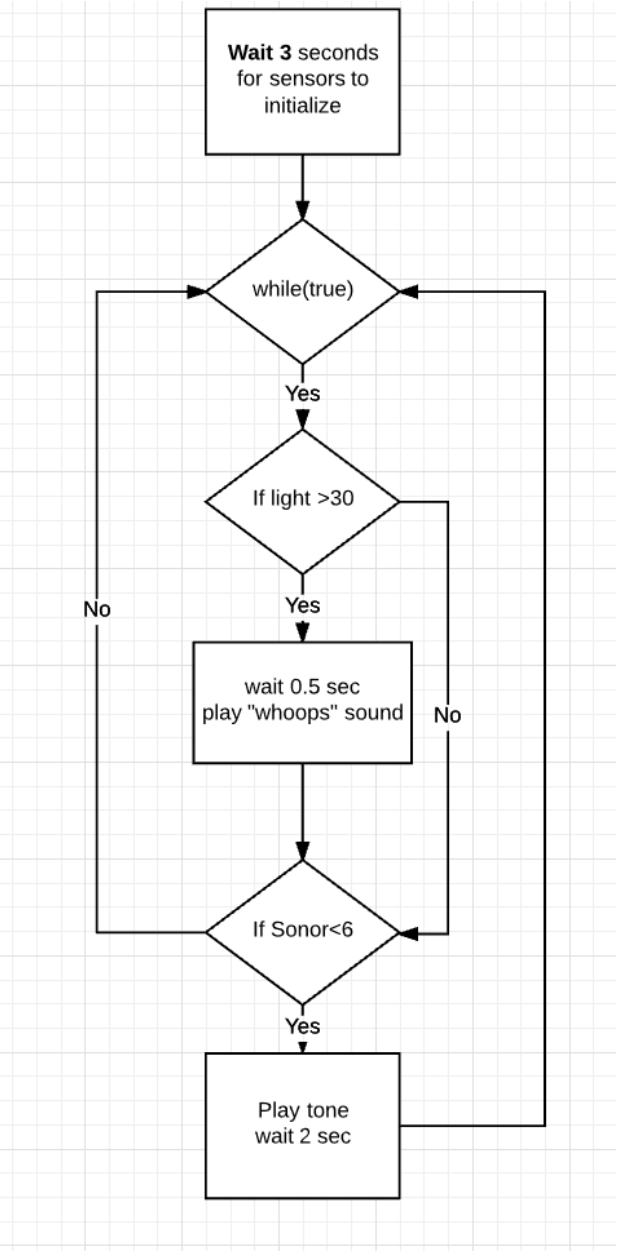
A separate branch in the flow chart is created

No repetition is done during an **if statement.**



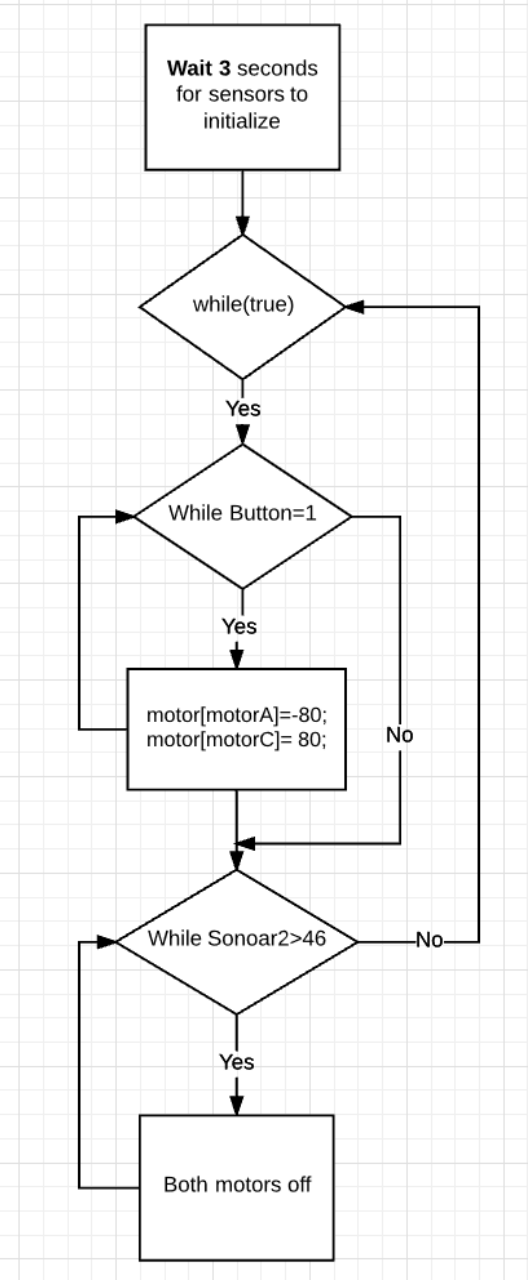
Example 1

```
task main()

{     wait1Msec (3000);

      while(true)
      {
            if (SensorValue[Light]>30)
                  {
                  wait1Msec(500);
                  PlaySoundFile("Woops.rso");
                  }

            if (SensorValue[Sonar]<6)
                  {
                  PlayTone(300, 15);
                  wait1Msec (2000);
                  }
      }
}
```



Example 2

```
task main()
{
  wait1Msec(3000);

  while (true)
  {

        while (SensorValue[button]==1)
                {
                motor[motorA]=-80;
                motor[motorC]= 80;
                }



        while (SensorRaw[SonarValue2]>46)
                {
               motor[motorA]=0;
               motor[motorC]=0;
                }

  }
}
```



Now you try:

Problem #1. Draw a correct algorithm for the code below:

```
task main()
{
     wait1Msec(3000);
     while(true)
     {
          eraseDisplay();
          wait1Msec(900);
          nxtDisplayCenteredBigTextLine(2,"Search for Black”);
          nxtDisplayCenteredBigTextLine(3,"%d",SensorValue[S1]);
          wait1Msec(900);

          if (SensorValue[S1]>25)
          {
               eraseDisplay();
               playTone(100, 10);
          }

     }
}
```

Problem #1. Draw a correct algorithm for the code below:

```
task main()
{

wait1Msec(2000);

     while (true)
     {

          while(SensorValue[Button1]==1)
          {
          eraseDisplay();
          nxtDisplayString(3, "INTRUDER");
          wait1Msec(2000);
          }

          while(SensorValue[Button1]==0)
          {
          eraseDisplay();
          nxtDisplayString(3,"ALL CLEAR - NO INTRUDER");
          wait1Msec(100);
          }

     }
}
```