

Lesson

**1**

---

# Introduction to Jackson Structured Programming (JSP)

---

## Contents

1. *Introduction*
2. *JSP Diagrams and Notation*
  - 2.1. *Sequence*
  - 2.2. *Selection*
  - 2.3. *Iteration*
3. *Passing & returning values to and from Functions*
4. *JSP Recursion*
5. *JSP and pseudo code*
6. *Comparative review of JSP and traditional Flowcharting*
7. *Summary*

## **1. Introduction**

I remember being introduced to JSP back in 1984 whilst at College, though it was a recently new methodology, most students found this to be fairly intuitive. I'm hoping that instructors alike will find this to be a key tool in effectively teaching the concept of logic and design.

The Jackson Program Design Methodology, sometimes called Jackson Structured Programming (JSP), is a method for program design and modelling. It begins with considerations about what is known and develops a program design that becomes more complete as the model is put through continued iterations. It is primarily based on functional top down decomposition and can be read top-down or bottom up. It builds program design from an incomplete model, hence termed as being a "composition type" method.

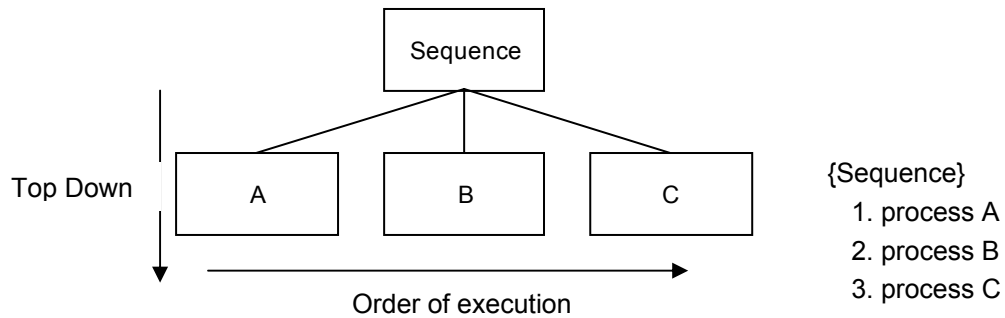
The founder of JSP, Michael Jackson developed this methodology in the early 70's that later emerged in industry and the academic field. By the 80' it was found to be a de-facto in Europe and widely taught in Colleges and Universities.

## 2. JSP Diagrams and Notations

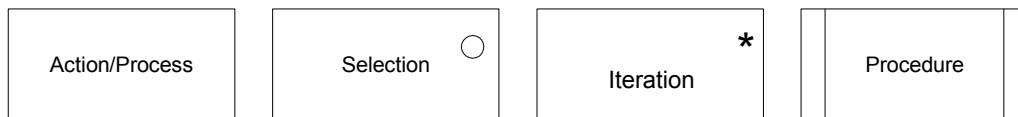
A Jackson Structured Programming diagram is used to explain the inner workings of a program. At a glance they seem similar to algorithm flowcharts, but the likeness is only superficial.

To understand a JSP diagram you must read it properly. With a JSP diagram each step on the same branch is performed top down - left to right.

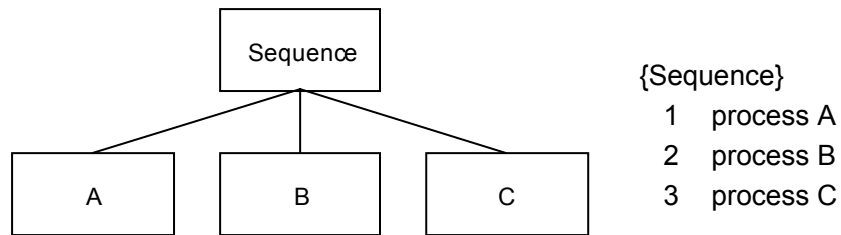
Sequence



With JSP diagrams there are three types of “box” notations used to represent the workings of a program. All programs have three control structures Sequence, Selection and Iteration!



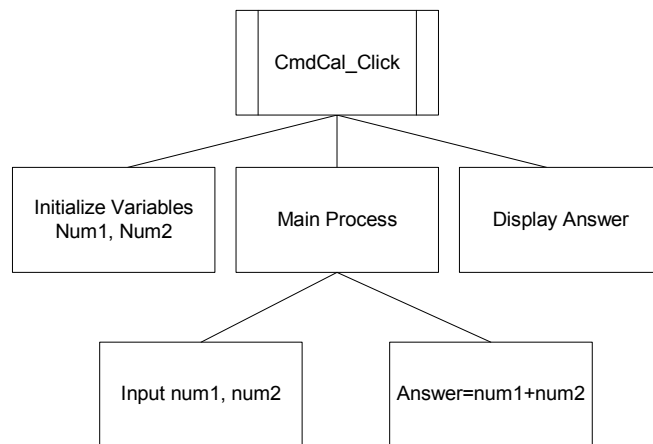
## 2.1. Sequence



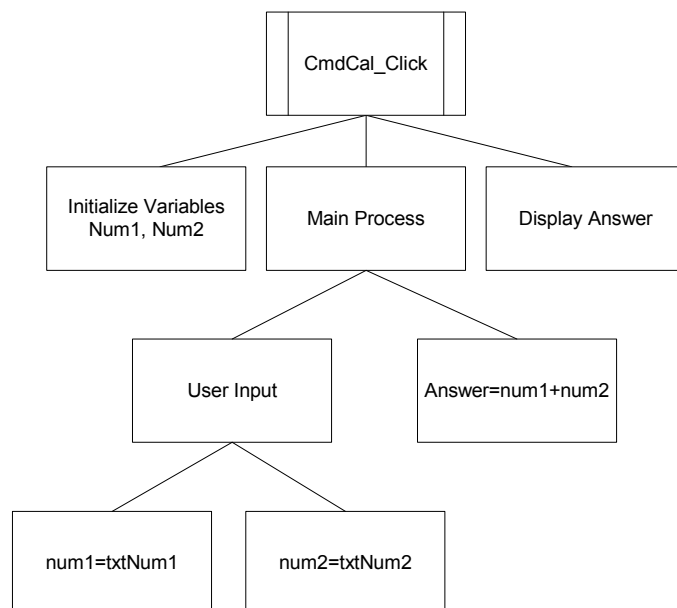
### Example 1

```

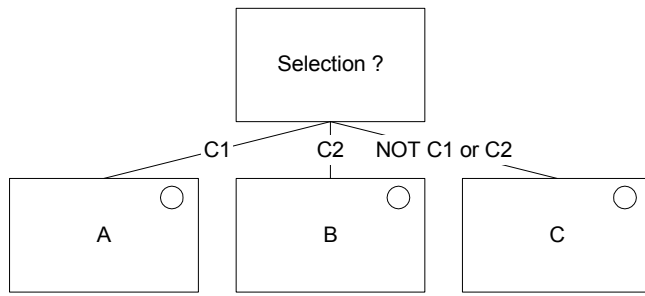
Private Sub cmdCalculate_Click()
  Dim num1 As Byte
  Dim num2 As Byte
  Dim answer As Byte
  num1 = Val(txtNum1)
  num2 = Val(txtNum2)
  answer = num1 + num2
  lblAnswer = CStr(answer)
End Sub
  
```



### Alternative (Refined)



## 2.2. Selection



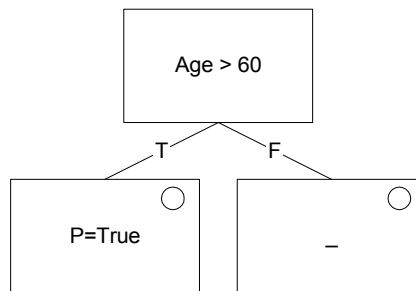
```

{Selection}
1  select
1.1a when c1
1.1b     process A
1.2a when c2
1.2b     process B
1.3a otherwise
1.3b     process C
1
    
```

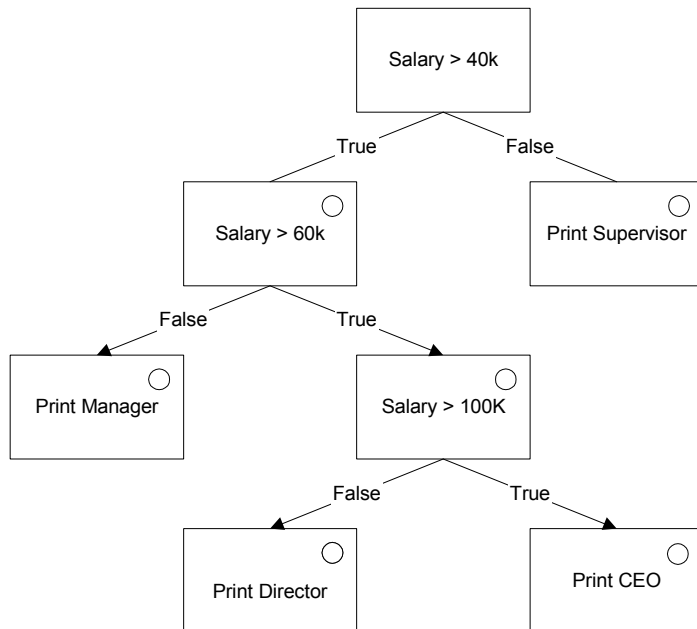
### Example 1 : IF statement

```

If Age > 60 Then
    P = True
Endif
    
```



### Example 2 : IF Then..Else statement

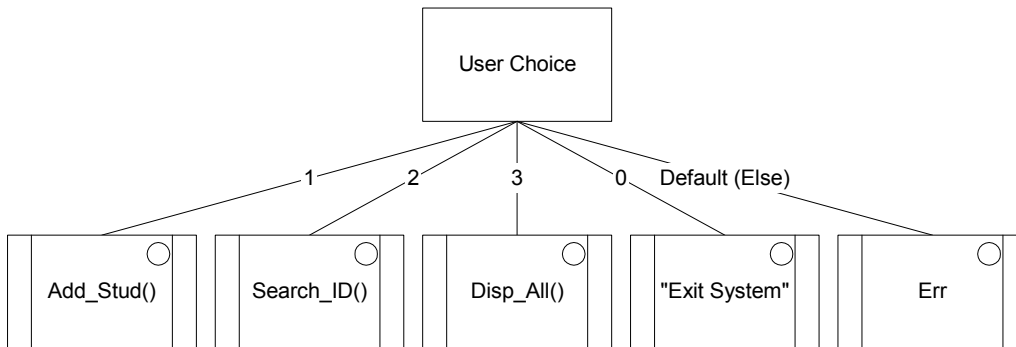


```

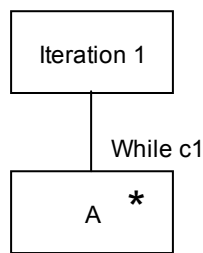
If Sal>40 Then
    if Sal>60 Then
        if Sal>100 Then
            Print "CEO";
        Else
            Print"Director"
        Else
            Print "Manager"
    Else
        Print "Supervisor"
    
```

**Example 3 : CASE statement**

```
switch(ch)
{
  case 1: Add_Stud();      break;
  case 2: Search_ID();    break;
  case 3: Disp_All();     break;
  case 0: cout<<"Exiting Student Management System...!\n\n";exit(1);
  default: cout<<"Invalid choice made....try again!"; break;
}
```

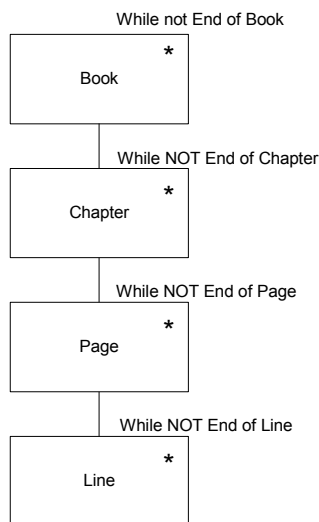


**2.3. Iteration**



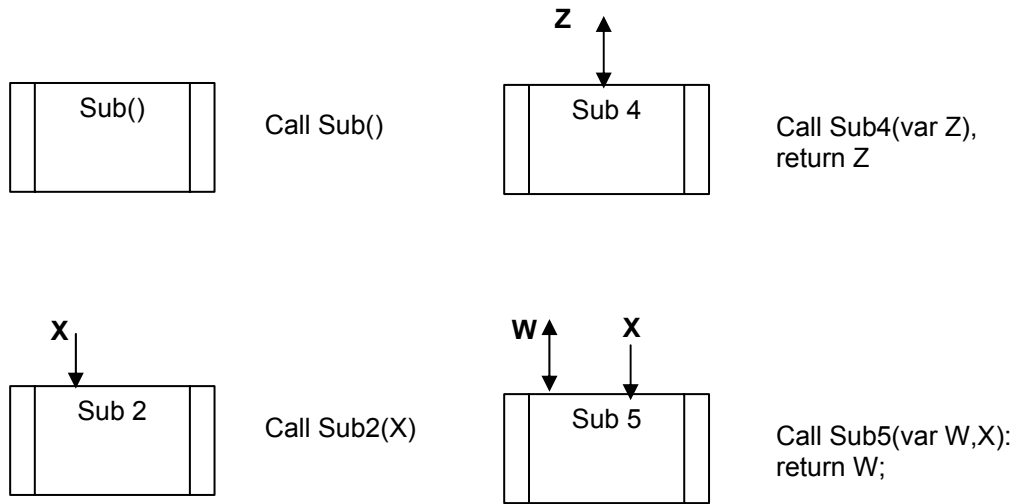
```
{Iteration 2}
1 repeat
1.1 process A
1 until c1
```

**Example to model a book:**

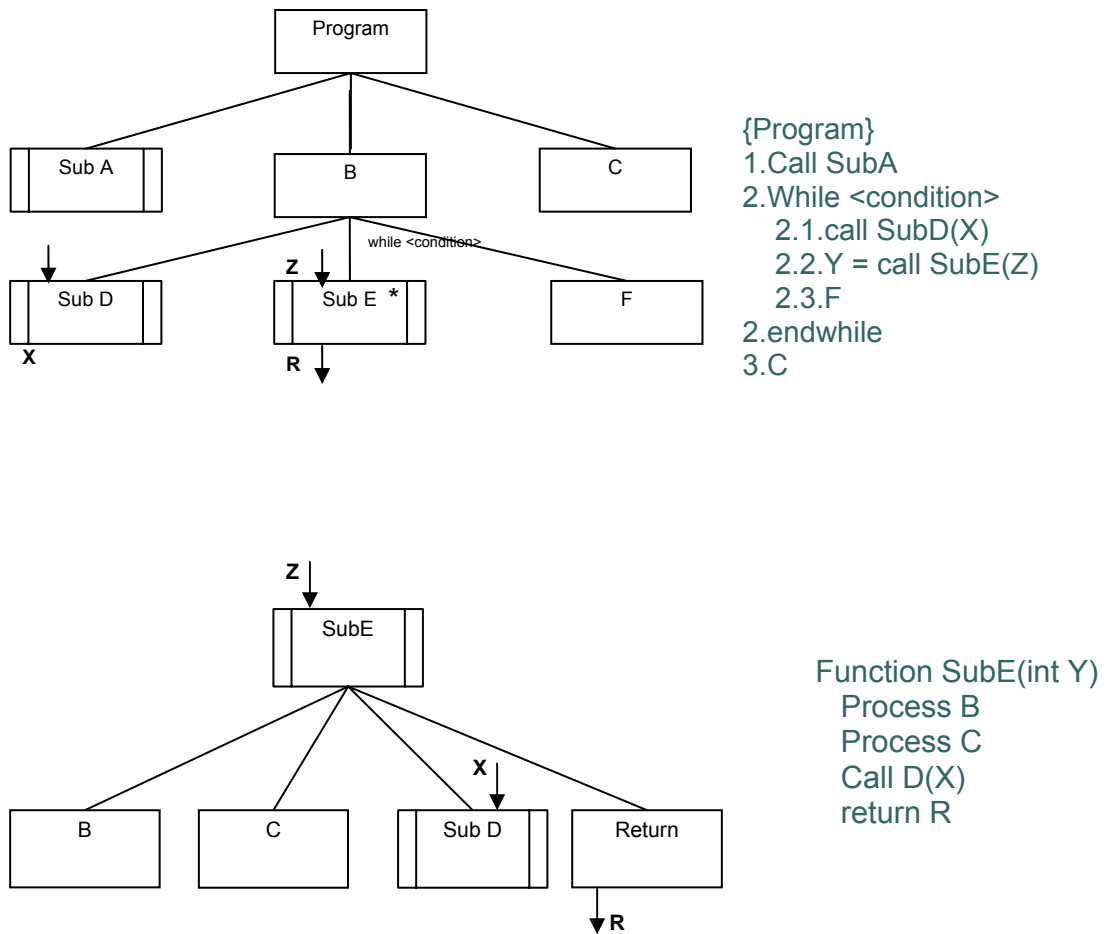


```
While NOT End of Book
  While NOT End of Chapter
    While NOT End of Page
      While NOT End of Line
        .....
      End While
    End While
  End While
End While
```

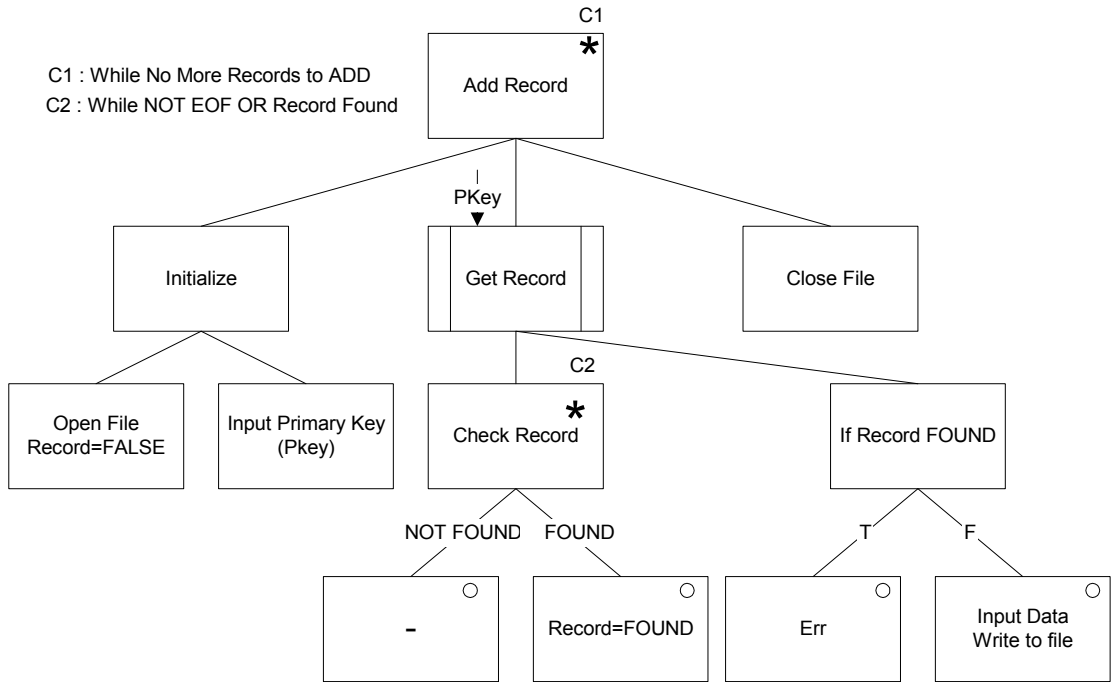
### 3. Passing & returning values to and from Functions



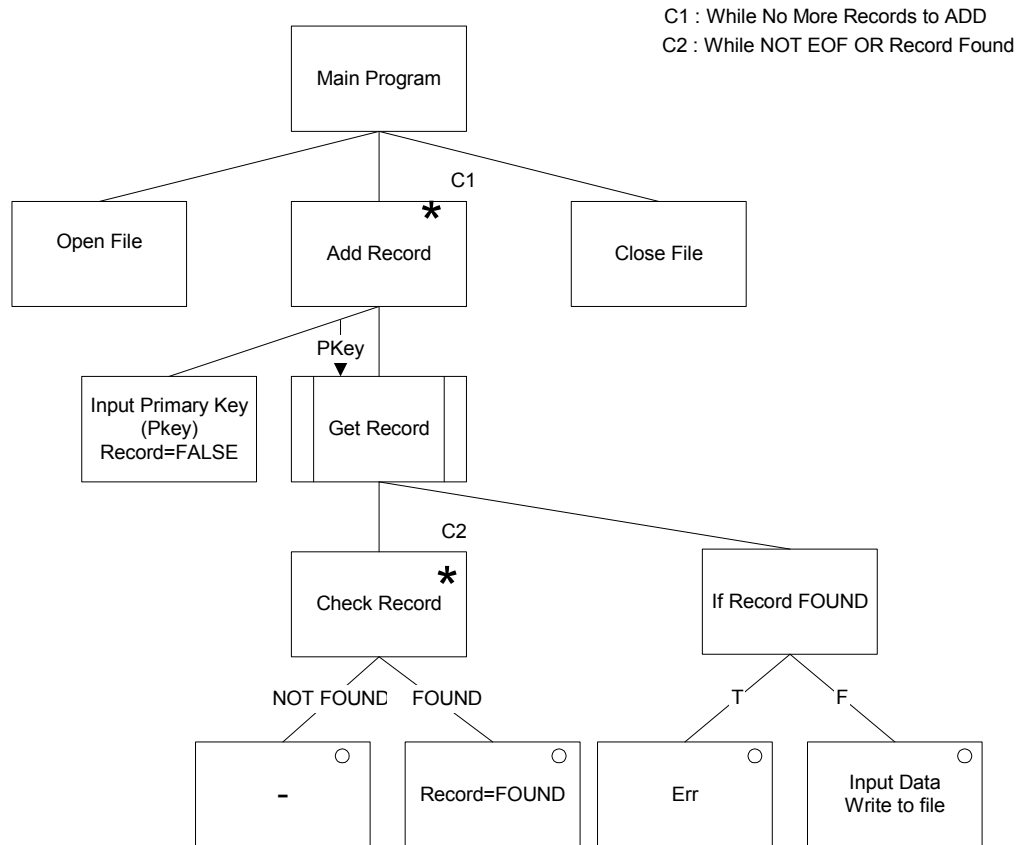
**Example 1:** Demonstrating calling functions with values/parameters.



**Example 2:** Design below demonstrates, sequence, selection and iteration based on a I/O file design (this example closes and opens a file for each add record task)



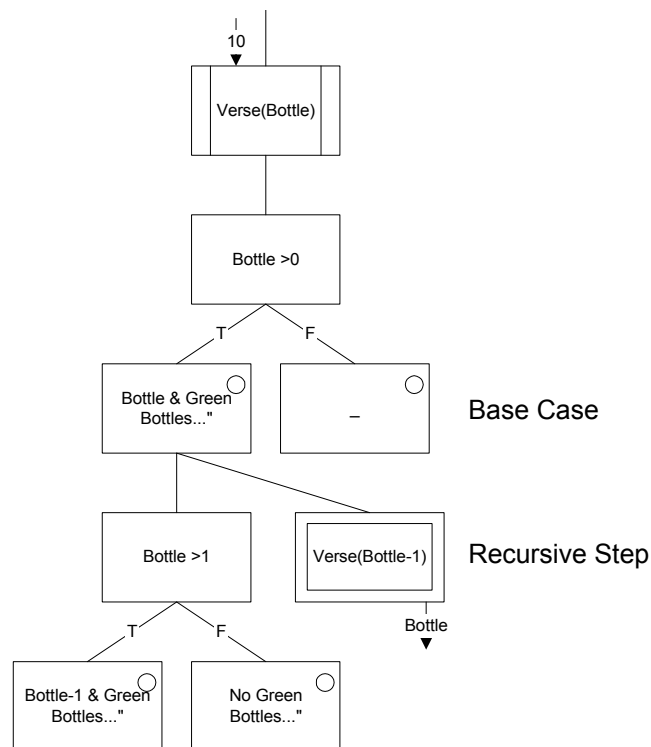
**Example below opens the file at the start of the program, and closes it the end**



## 4. JSP Recursion

A recursive routine is when a procedure calls itself. Each recursive call made allows the problem to get closer to a solution. The Tower of Hanoi is a typical example of recursion. JSP tackles recursion simply by introducing a double line process box that represents a call to itself.

The example below demonstrates the 10 green bottle song (or 10 beers as the Americans would say!) with a recursive call



The recursive component of a JSP object or action design is indicated by a double lined component box. The code corresponding to this design has been included below.

```

main ()
{
    verse(10);           // Call verse
    return 0;
}

int verse(int bottle)
{
    if (bottle > 0 )    // Where n is number of bottles
    {
        cout << bottle << " green bottle";
        cout << " hanging on the wall...\n\n";
        cout << "and if 1 green bottle should accidentally fall? \n";
        if (bottle > 1)
        {
            cout << "There'll be " << (bottle-1) << " green bottle";
            cout << " hanging on the wall\n";
        }
        else
        {
            cout << "There'll be no green bottles hanging on the wall\n\n";
        }
        verse(bottle-1); //Recursive call where bottle=bottle-1 (1 less bottle)
    }
    return 0;
}

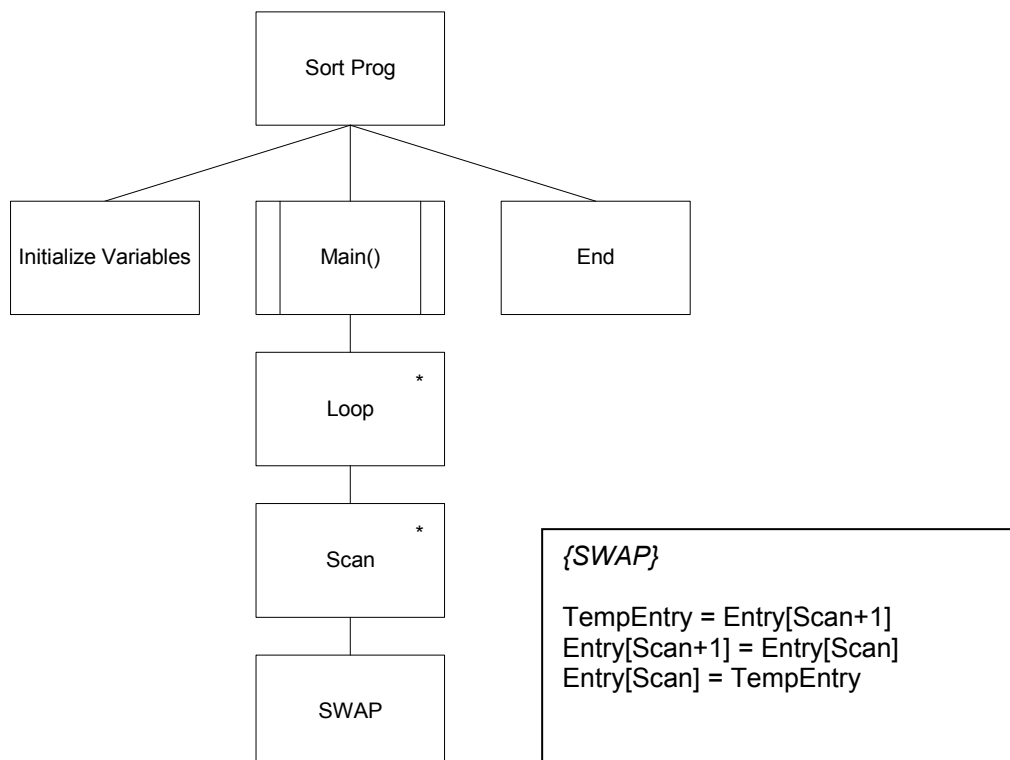
```

## 5. JSP & Pseudo code

Generally a JSP diagram is adequate to describe a program; if more detail is required you can easily combine JSP diagrams and pseudo code. Pseudo code could reflect process boxes in particular where complex algorithms may be difficult to represent as JSP.

The example below uses both JSP and Pseudo code to represent the design of a simple bubble sort program.

```
int main()
{
  for (loop = 0; loop < 6; loop++)
  {
    for (Scan = 0; Scan < 6; Scan++)
    {
      if (Entry[Scan] > Entry[Scan+1])
      {
        TempEntry = Entry[Scan+1];
        Entry[Scan+1] = Entry[Scan];
        Entry[Scan] = TempEntry;
      }
    }
  }
  return 0;
}
```

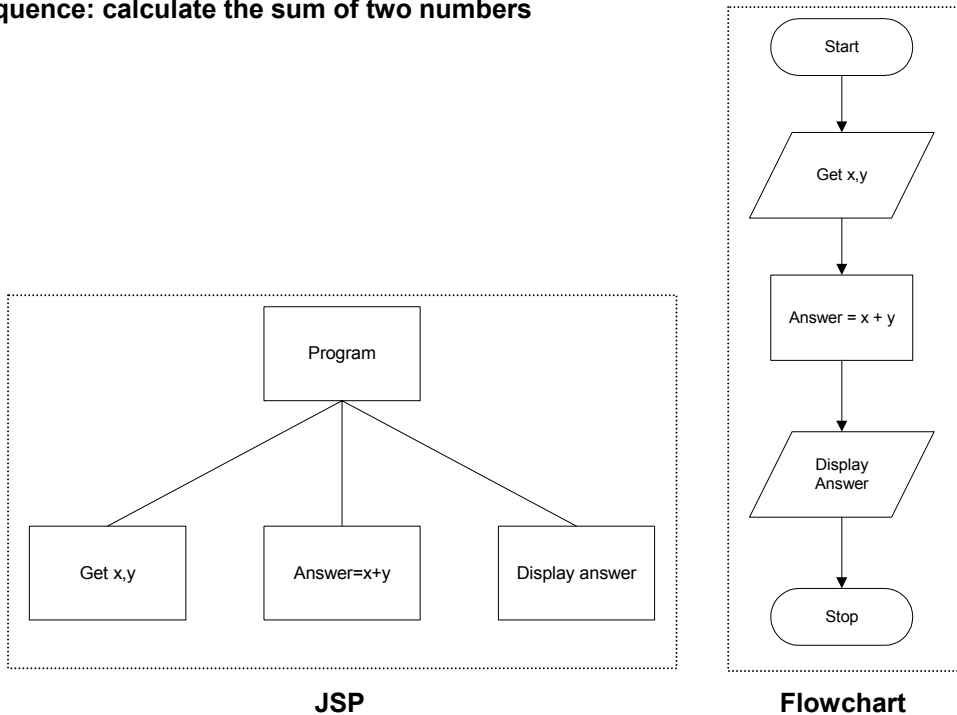


Pseudo code could also be included for each process box, dependant on the application being developed. If this is the case, JSP should still reflect an overall break down of the problem.

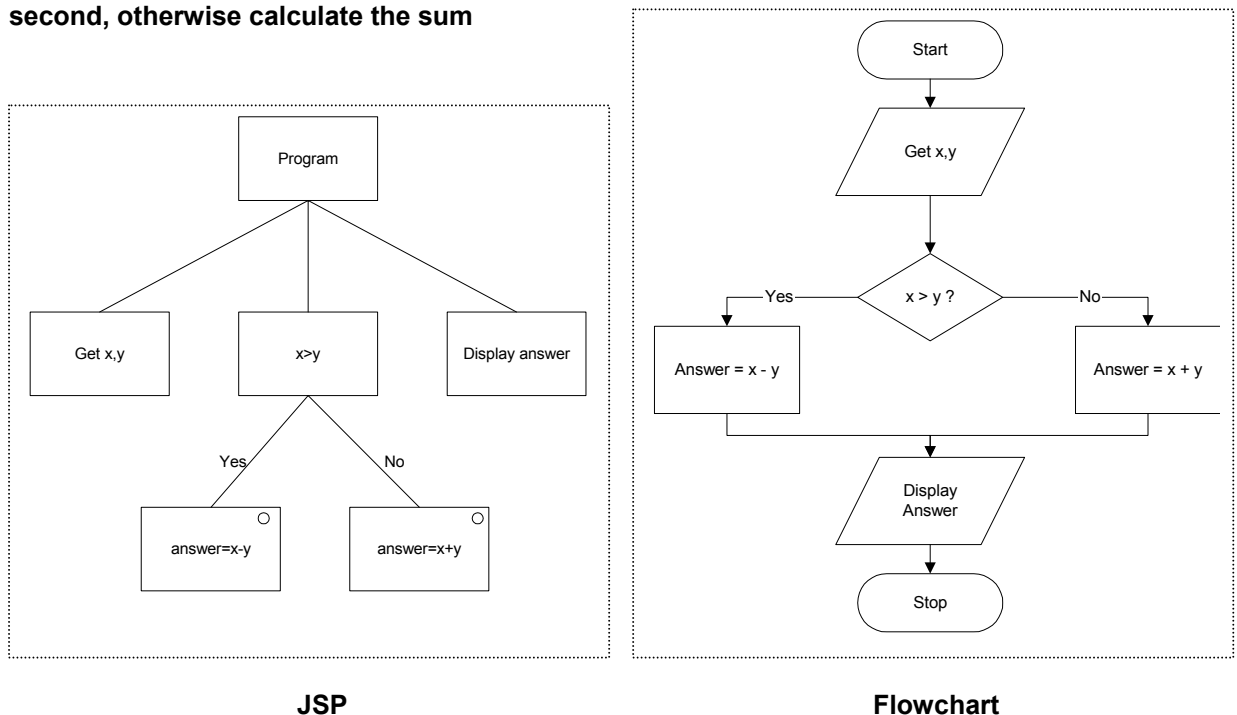
## 6. Comparative review of JSP and traditional Flowcharting

A diagrammatical comparison between traditional flowcharting and JSP methodology is covered in this section. A snap shot picture between the designs clearly shows that JSP is found to be simplistic, clear and avoids the fuzziness shown by the flowchart method. The JSP methodology tends to avoid the fuzziness that can be created by too many notations adopted by flowcharting.

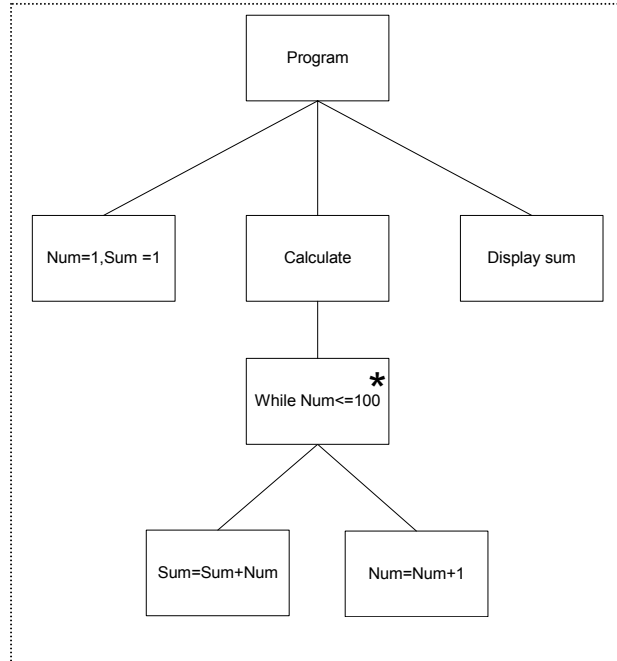
**Sequence: calculate the sum of two numbers**



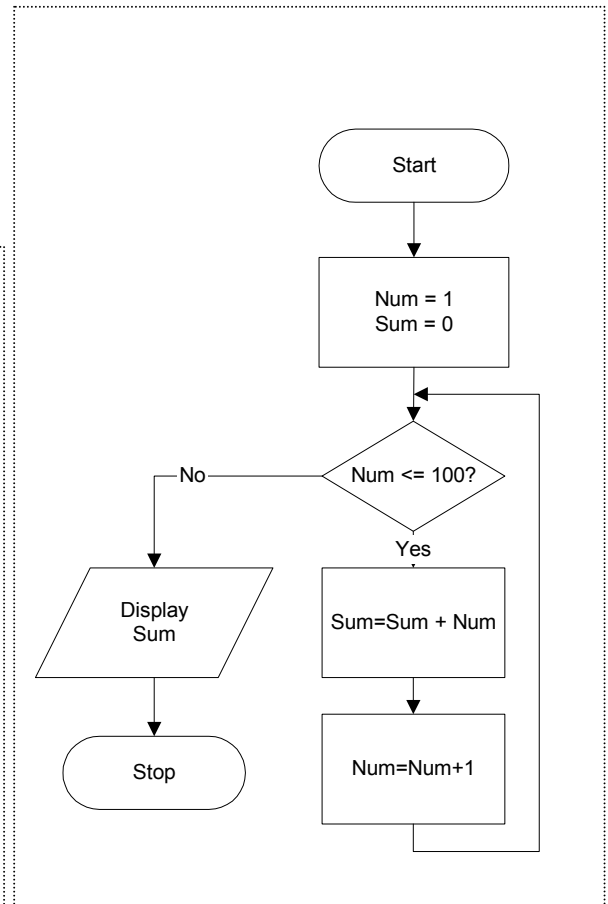
**Selection: Calculate the difference of two numbers if the first is greater than the second, otherwise calculate the sum**



**Iteration: Draw a flowchart to show how to calculate the sum of the numbers 1 to 100**



**JSP**



**Flowchart**

## **7. Summary**

JSP modelling does involve the emergence of input and output that eventually creates the structure of the final design. This chapter has only focussed on how simple problems could be broken down and, solutions designed that are later transformed into code.

This chapter concentrates on allowing students in understanding logic and how to “think logically” prior to coding.

The application of Jackson Program Design include small-scale development of program modules, reports, sequential file processing, and similar areas that do not involve modelling-in-the-large. However, all types of problems can be solved using JSP, and it is a clear, robust system for thinking about problems.

Surveys indicate JSP/JSD (Jackson Structured design) is widely recognized and used in Europe than in the USA. JSP’s intentions have been quite clear in terms of developing structured systems, in particular how modules are integrated to form the whole application. Its primary implementation has close ties with SSADM (Structured Systems Analysis and Design Methodology).

The strength of Jackson Program Design lies in its description of what a program is to do. Its basic characteristics that give it advantages in its systematic approach include:

- Rational- based on reasonable principles, well developed and well defined: structured programming (iteration, selection, sequence).
- Teachable- it has a well set of defined steps, it is possible to explain and show how this system is applied and how it works.
- CONSISTENT- given a single problem for analysis, different programmers or analysts will come up with more or less the same program design.
- SIMPLE- the method does not involve overly complex rules, cumbersome procedures or difficult notations to remember.
- UNIVERSAL- specifications are produced that can be implemented in the language the developer desires.

### Other Resources

JAVA JSP Editor Creates C/Pascal Code from design

<http://flightline.highline.edu/rkang/>

OR

<http://www.ida.his.se/ida/~henrike/JSP/example.shtml>