



Flowchart to code



FEBRUARY 13, 2017

SUPERVISOR

Prof. Yousef B. Mahdy

Project team members:

- **Omar Mohamed Ahmed**
- **Kareem Khaled Mohllal**
- **Mohamed Hosny Abd-Allah**

Chapter 1: INTRODUCTION TO FLOWCHARTING

Concepts of chapter

1.1- Introduction
1.2- Flowcharts
1.3- Types of Flowcharts
1.4 Flowchart Symbols
1.5 Advantages of Flowcharts.....
1.6 Limitations of Using Flowcharts
1.7 When to Use a Flowchart
1.8 Developing Flowcharts.....
1.9 Techniques
1.10 Summary

Chapter 2: Application

Concepts of chapter:

2.1 Introduction
2.2 Shapes.....

Chapter 3: Implementation

Concepts of chapter:

3.1 Project Save
3.2 Project Saver Class...
3.3 Nodes.....

3.4 ConnectorNode Class

3.5 Sample.....

Chapter 4: FrameWork

Concepts of chapter:

4.1 Important Classes Used

Chapter 1: INTRODUCTION TO FLOWCHARTING

Concepts of chapter1.

1- Introduction	6
1.2- Flowcharts	9
1.3- Types of Flowcharts	10
1.4 Flowchart Symbols	12
1.5 Advantages of Flowcharts.....	12
1.6 Limitations of Using Flowcharts	
1.7 When to Use a Flowchart	
1.8 Developing Flowcharts.....	12
1.9 Techniques	12
1.10 Summary	

Chapter One

INTRODUCTION TO FLOWCHARTING

1.1- Introduction:

Computers are capable of handling various complex problems which are tedious and routine in nature. In order that a computer solve a problem, a method for the solution and a detailed procedure has to be prepared by the programmer. The problem solving Involves :

- Detailed study of the problem

- Problem redefinition
- Identification of input data, output requirements and conditions and limitations
- Alternative methods of solution
- Selection of the most suitable method
- Preparation of a list of procedures and steps to obtain the solution
- Generating the output.

The preparation of lists of procedures and steps to obtain the result introduces the algorithmic approach to problem solving. The algorithm is a sequence of instructions designed in such a way that if the instructions are executed in a specific sequence the desired results will be obtained. The instructions should be precise and concise and the result should be obtained after a finite execution of steps. This means that the algorithm should not repeat one or more instructions infinitely. It should terminate at some point and result in the desired output. An algorithm should possess the following characteristics :

- Each and every instruction should be precise and clear
- Each instruction should be performed a finite number of times
- The algorithm should ultimately terminate - When the algorithm terminates the desired result should be obtained.

1.2 FLOWCHARTS:

Before you start coding a program it is necessary to plan the step by step solution to the task your program will carry out. Such a plan can be symbolically developed using a diagram. This diagram is then called a flowchart. Hence a flowchart is a symbolic representation of a solution to a given task. A flowchart can be developed for practically any job. Flowcharting is a tool that can help us to develop and represent graphically program logic sequence. It also enables us to trace and detect any logical or other errors before the programs are written.

1.3 TYPES OF FLOWCHARTS:

Computer professionals use two types of flowcharts viz :

- Program Flowcharts.
- System Flowcharts

1.3.1 Program Flowcharts :

These are used by programmers. A program flowchart shows the program structure, logic flow and operations performed. It also forms an important part of the documentation of the system. It broadly includes the following:

- Program Structure.

- Program Logic.
- Data Inputs at various stages.
- Data Processing
- Computations and Calculations.
- Conditions on which decisions are based.
- Branching & Looping Sequences.
- Results.
- Various Outputs.

The emphasis in a program flowchart is on the logic.

1.3.2 System Flowcharts :

System flowcharts are used by system analyst to show various processes, sub systems, outputs and operations on data in a system.

1.4 FLOWCHART SYMBOLS

Normally, an algorithm is expressed as a flowchart and then the flowchart is converted into a program with the programming language. Flowcharts are independent of the programming language being used. Hence one can fully concentrate on the logic of the problem solving at this stage. A large number of programmers use flowcharts to assist them in the development of computer programs. Once the flowchart is fully ready, the programmer then write it in the programming language. At this stage he need not concentrate on the logic but can give more attention to coding each instruction in the box of the flowchart in terms of the statements of the programming language selected.

A flowchart can thus be described as the picture of the logic to be included in the computer program. It is always recommended for a beginner, to draw flowcharts prior to writing programs in the selected language. Flowcharts are very helpful during the testing of the program as well as incorporating further modifications.

Flowcharting has many standard symbols. Flowcharts use boxes of different shapes to denote different types of instructions. The actual instruction is written in the box. These boxes are connected with solid lines which have arrowheads to indicate the direction of flow of the flowchart. The boxes which are used in flowcharts are standardized to have specific meanings. These flowchart symbols have been standardized by the American National Standards Institute. (ANSI).

While using the flowchart symbols following points have to be kept in mind:

- The shape of the symbol is important and must not be changed.
- The size can be changed as required.

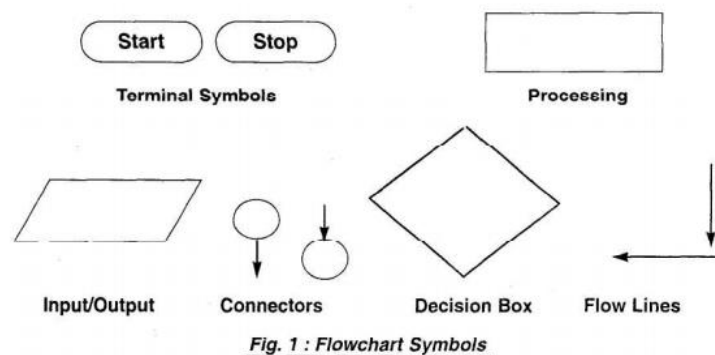
- The symbol must be immediately recognizable.
- The details inside the symbol must be clearly legible.
- The flow lines, as far as possible, must not cross.

Terminal Symbol:

Every flowchart has a unique starting point and an ending point. The flowchart begins at the start terminator and ends at the stop terminator. The Starting Point is indicated with the word START inside the terminator symbol. The Ending Point is indicated with the word STOP inside the terminator symbol. There can be only one

Input/output Symbol :

This symbol is used to denote any input/output function in the program. Thus if there is any input to the program via an input device, like a keyboard, tape, card reader etc. it will be indicated in the flowchart with the help of the Input/output symbol. Similarly, all output instructions, for output to devices like printers, plotters, magnetic tapes, disk, monitors etc. are indicated in the Input/output symbol.



Process Symbol :

A process symbol is used to represent arithmetic and data movement instructions in the flowchart. All arithmetic processes of addition, subtraction, multiplication and division are indicated in the process symbol. The logical process of data movement from one memory location to another is also represented in the process box. If there are more than one process instructions to be executed sequentially, they can be placed in the same process box, one below the other in the sequence in which they are to be executed.

Decision Symbol :

The decision symbol is used in a flowchart to indicate the point where a decision is to be made and branching done upon the result of the decision to one or more alternative paths. The criteria for decision making is written in the decision box. All the possible paths should be accounted for. During execution, the appropriate path will be followed depending upon the result of the decision.

Flow lines :

Flow lines are solid lines with arrowheads which indicate the flow of operation. They show the exact sequence in which the instructions are to be executed. The normal flow of the flowchart is depicted from top to bottom and left to right.

Connectors :

In situations, where the flowcharts becomes big, it may so happen that the flow lines start crossing each other at many places causing confusion. This will also result in making the flowchart difficult to understand. Also, the flowchart may not fit in a single page for big programs. Thus whenever the flowchart becomes complex and spreads over a number of pages connectors are used. The connector represents entry from or exit to another part of the flowchart. A connector symbol is indicated by a circle and a letter or a digit is placed in the circle. This letter or digit indicates a link. A pair of such identically labeled connectors are used to indicate a continued flow in situations where flowcharts are complex or spread over more than one page. Thus a connector indicates an exit from some section in the flowchart and an entry into another section of the flowchart. If an arrow enters a flowchart but does not leave it, it means that it is an exit point in the flowchart and program control is transferred to an identically labeled connector which has an outlet. This connector will be connected to the further program flow from the point where it has exited. Connectors do not represent any operation in the flowchart. Their use is only for the purpose of increased convenience and clarity.

1.5 ADVANTAGES OF FLOWCHARTS :

There are a number of advantages when using flowcharts in problem solving. They provide a very powerful tool to programmers to first represent their program logic graphically and independent of the programming language.

- Developing the program logic and sequence. A macro flowchart can first be designed to depict the main line of logic of the software. This model can then be broken down into smaller detailed parts for further study and analysis.

- A flowchart being a pictorial representation of a program, makes it easier for the programmer to explain the logic of the program to others rather than a program .
- It shows the execution of logical steps without the syntax and language complexities of a program.
- In real life programming situations a number of programmers are associated with the development of a system and each programmer is assigned a specific task of the entire system. Hence, each programmer can develop his own flowchart and later on all the flowcharts can be combined for depicting the overall system. Any problems related to linking of different modules can be detected at this stage itself and suitable modifications carried out. Flowcharts can thus be used as working models in design of new software systems.
- Flowcharts provide a strong documentation in the overall documentation of the software system.
- Once the flowchart is complete, it becomes very easy for programmers to write the program from the starting point to the ending point. Since the flowchart is a detailed representation of the program logic no step is missed during the actual program writing resulting in error free programs. Such programs can also be developed faster.
- A flowchart is very helpful in the process of debugging a program. The bugs can be detected and corrected with the help of a flowchart in a systematic manner.
- A flowchart proves to be a very effective tool for testing. Different sets of data are fed as input to program for the purpose.
 - Communication: Flowcharts are better way of communicating the logic of a system to all concerned.
 - Effective analysis: With the help of flowchart, problem can be analyzed in more effective way.
 - Proper documentation: Program flowcharts serve as a good program documentation, which is needed for various purposes.
 - Efficient Coding: The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
 - Proper Debugging: The flowchart helps in debugging process.
 - Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flowchart.
 - It helps the programmer to put efforts more efficiently on that part.

1.6 Limitations of Using Flowcharts:

Although a flowchart is a very useful tool, there are a few limitations in using flowcharts which are listed below:

- Complex logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
- Alterations and Modifications: If alterations are required the flowchart may require re-drawing completely.

1.7 When to Use a Flowchart:

1. To communicate to others how a process is done.
2. A flowchart is generally used when a new project begins in order to plan for the project.
3. A flowchart helps to clarify how things are currently working and how they could be improved.
4. It also assists in finding the key elements of a process, while drawing clear lines between where one process ends and the next one starts.
5. Developing a flowchart stimulates communication among participants and establishes a common understanding about the process.
6. Flowcharts also uncover steps that are redundant or misplaced.
7. Flowcharts are used to help team members, to identify who provides inputs or resources to whom, to establish important areas for monitoring or data collection, to identify areas for improvement or increased efficiency, and to generate hypotheses about causes.
8. It is recommended that flowcharts be created through group discussion, as individuals rarely know the entire process and the communication contributes to improvement.
9. Flowcharts are very useful for documenting a process (simple or complex) as it eases the understanding of the process.

10. Flowcharts are also very useful to communicate to others how a process is performed and enables understanding of the logic of a process.

1.8 DEVELOPING FLOWCHARTS

In developing the flowcharts following points have to be considered:

- Defining the problem.
- Identify the various steps required to form a solution.
- Determine the required input and output parameters.
- Get expected input data values and output result.
- Determine the various computations and decisions involved. With this background of flowcharts and flowchart symbols let us now draw some sample flowcharts. First we shall write the steps to prepare the flowchart for a particular task and then draw the flowchart.

Example : To prepare a flowchart to add two numbers. (Fig. 2a.)

The steps are :

1. Start.
2. Get two numbers N1 and N2.
3. Add them.
4. Print the result.
5. Stop.

Example : To prepare a flowchart to determine the greatest of two numbers. Here we use the decision symbol. We also combine the two reads for numbers A and B in one box.

The steps are :

1. Start
2. Get two number A and B.
3. If $A > B$ then print A else print B.
4. Stop.

Note that in the first example, we have used two separate input/output boxes to read the numbers N1 and N2. In the second example, both the numbers a and b are read in the same box. Thus if more than one instructions of the same kind follow one another then they can be combined in the same box.

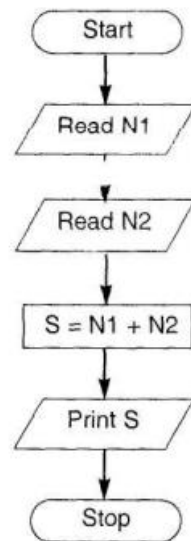


Fig. 2a) Flowchart to add two numbers N1 and N2

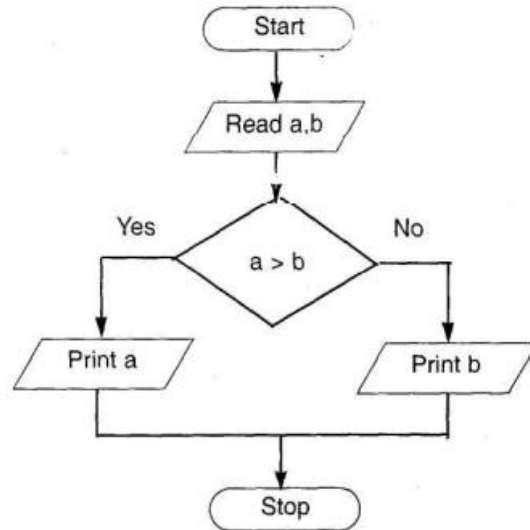


Fig. 2b) Flowchart to determine the greater of two numbers a and b

1.9 TECHNIQUES

In this section we shall cover the various flowcharting techniques viz.

- flowcharts for computations
- flowcharts for decision making
- flowcharts for loops
- Predefined Process

1.9.1 Flowcharts for Computations :

Computers are used to perform many calculations at high speed. When you develop a program it also involves several calculations.

The general format of the flowcharting steps for computations is :

- Create members variables used in calculations and read operation
- Get required data input using members variables.
- Perform the necessary calculations.
- Print the result.

Programming considerations while using computation techniques : Most languages have provision for creating members variables. The exact syntax depends on the language used. In most cases (but not all) your programs have to create and initialize the members variables before you can use them.

The following examples show the usage of flowcharts in computations. The flowcharts are shown in Fig.3a and Fig. 3b.

Example : Flowchart for a program that converts temperature in degrees Celsius to degrees Fahrenheit. First let us write the steps involved in this computation technique.

1. Start.
2. Create memvars F and C (for temperature in Fahrenheit and Celsius).
3. Read degrees Celsius into C.
4. Compute the degrees Fahrenheit into F.
5. Print result (F).
6. Stop.

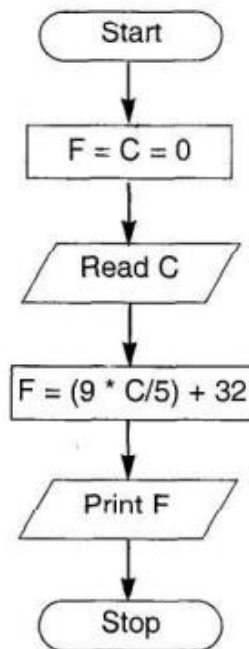


Fig. 3a) Flowchart to convert temperature from Celsius to Fahrenheit

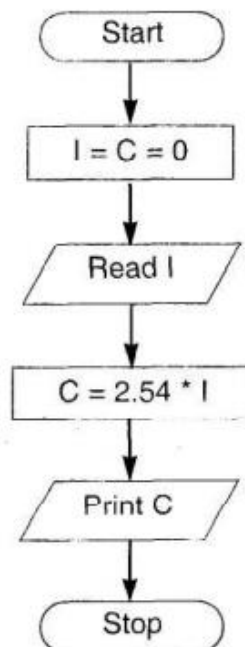


Fig. 3b) Flowchart to convert inches to centimetres

Example : Flowchart for a program that converts inches to centimeters First let us write the steps involved in this computation technique.

1. Start.
2. Create members variables C and I (for Centimeters and Inches respectively).
3. Read value of Inches into I
4. Compute the Centimeters into C.
5. Print result (C).
6. Stop.

1.9.2 Flowcharts for decision making :

Computers are used extensively for performing various types of analysis. The decision symbol is used in flowcharts to indicate it.

The general format of steps for flowcharting is as follows:

- Perform the test of the condition.
- If condition evaluates true branch to Yes steps.
- If condition evaluates false branch to No steps.

Programming Considerations :

Most programming languages have commands for performing test and branching. The exact commands and syntax depends on the language used. Some of the conditional constructs available in programming languages for implementing decision making in programs are as follows:

- If
- If
- else
- endif
- If
- elseif
- endif
- Do case - endcase.
- Switch.

All languages do not support all of the above constructs. The operators available for implementing the decision test are as follows:

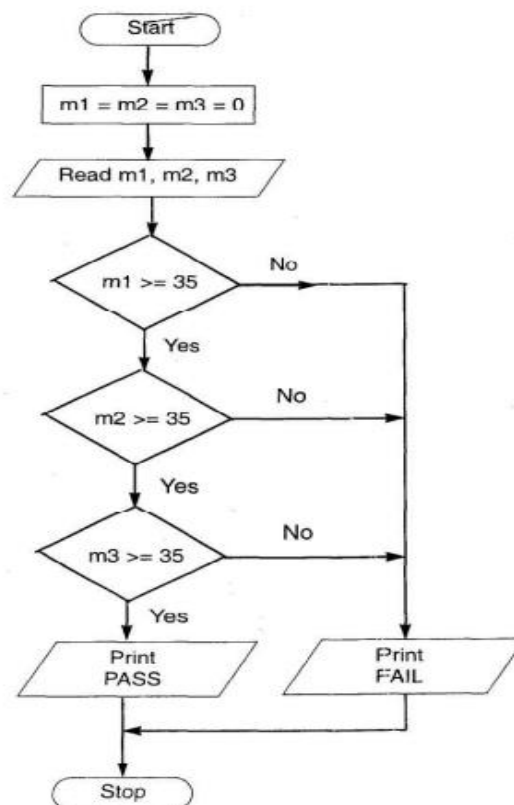
- Relational Operators (which determine equality or inequality)
- Logical Operators, (useful for combining expressions) The branching to another set of commands can be implemented by using functions, procedures etc.

Example:

Flowchart to get marks for 3 subjects and declare the result. If the marks ≥ 35 in all the subjects the student passes else fails. The steps involved in this process are :

1. Start.
2. Create members variables m1, m2, m3.
3. Read marks of three subjects m1, m2, m3.
4. If m1 ≥ 35 goto step 5 else goto step 7
5. If m2 ≥ 35 goto step 6 else goto step 7
6. If m3 ≥ 35 print Pass. Goto step 8
7. Print fail

8. Stop



The flowchart is shown in Fig. 4.

An alternative method is the one in which you can combine all the conditions with the AND operator.

The steps then would be :

1. Start
2. Create members variables m1, m2, m3.
3. Read marks of three subjects into m1, m2 and m3.
4. If m1 >= 35 and m2 >= 35 and m3 >= 35 print Pass. Otherwise goto step 5.
5. Print Fail.
6. Stop Developing this flowchart is left as an exercise to the student.

1.9.3 Flowcharts for loops :

Looping refers to the repeated use of one or more steps. i.e. the statement or block of statements within the loop are executed repeatedly. There are two types of loops. One is known as the fixed loop where the operations are repeated a fixed number of times. In this case, the values of the variables within the loop have no effect on the number of times the loop is to be executed. In the other type which is known as the variable loop, the operations are repeated until a specific condition is met. Here, the number of times the loop is repeated can vary.

The loop process in general includes :

- Setting and initializing a counter

- execution of operations
 - testing the completion of operations
 - incrementing the counter
- The test could either be to determine whether the loop has executed the specified number of times, or whether a specified condition has been met.

Programming considerations :

Most of the programming languages have a number of loop constructs for efficiently handling repetitive statements in a program. These include :

- do-while loop
- while loop
- for loop
- for-next loop

In most of the looping situations, we make use of counters. In situations where the loop is to be repeated on the basis of conditions, relational operators are used to check the conditions.

Example : To find the sum of first N numbers. This example illustrates the use of a loop for a specific number of times. Fig. 5a. The steps are :

1. Start
2. Create members variables S , N, I
3. Read N
4. Set S (sum) to 0
5. Set counter (I) to 1. 6. $S = S + I$
7. Increment I by 1.
8. Check if I is less than or equal to N. If no, go to step 6.
9. Print S
10. Stop

The flowchart is shown in Figure 5a.

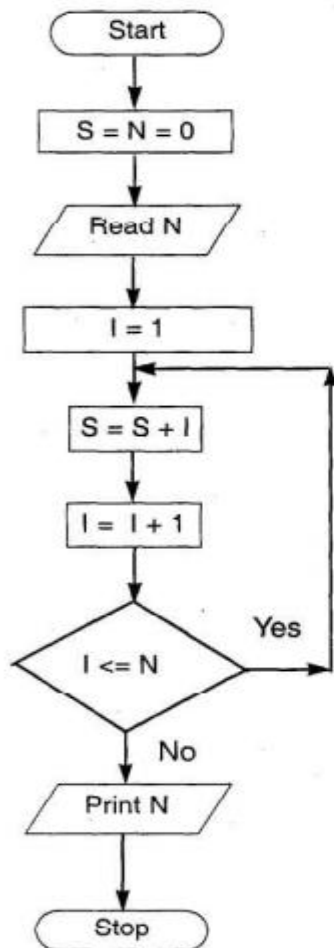


Fig.5a Flowchart to find sum of first N numbers

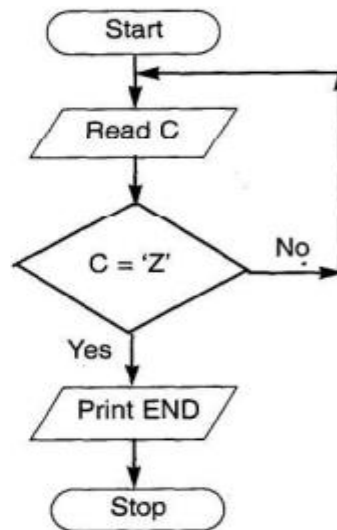


Fig.5b Flowchart to read a character till Z is input

Chapter 2: Application

Concepts of chapter:

2.1 Introduction	17
2.2 Shapes	17
2.3 Data Types	21
2.4 Identifiers	22
2.5 Shapes Properties	24

Chapter 2 Application

2.1 Introduction

FlowchartToCode is a free application that helps you create programs using simple flowcharts.

Typically, programs are written using a text editor. Depending on the programming language, this can be either easy or quite difficult for a beginning programmer. Many languages require you to write lines upon lines of confusing code just to get it to display "Hello, world!".

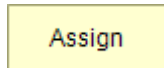
However, using FlowchartToCode, you can use shapes to represent the different actions that you want your program to perform. So, you can concentrate on the algorithm rather than all the nuances and details of a typical programming language.

You can execute your programs directly in FlowchartToCode. But, if you want to learn a high-level programming language, FlowchartToCode can convert your flowchart to many popular languages. These include: C# and C++ for Applications.

2.2 Shapes

- Assignment Shape

Default Appearance

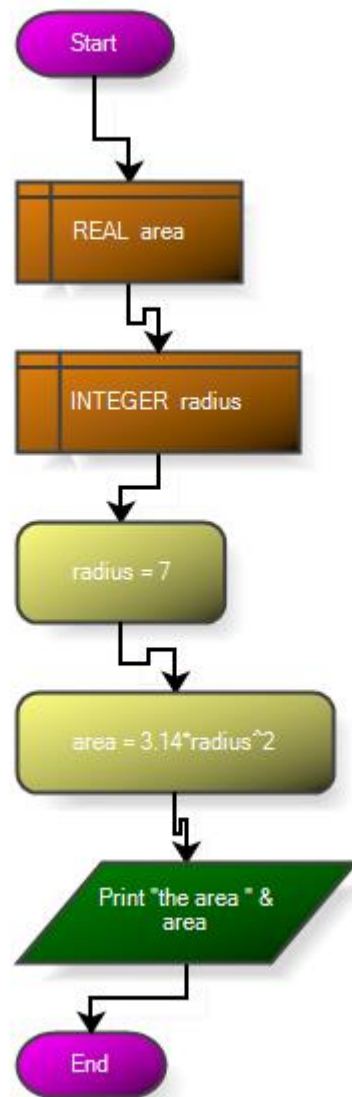


What it Does

The Assignment shape is used to store the result of a calculation into a variable. This is one of the most common tasks found in programs.

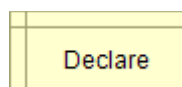
Example

The example, to the right, declares two variables: area (which stores real numbers) and radius (which stores integers). It then uses an Assignment Statement to set the 'radius' to 7. Finally, it computes the area of a circle and stores the result in 'area'.



- Declare Shape

Default Appearance

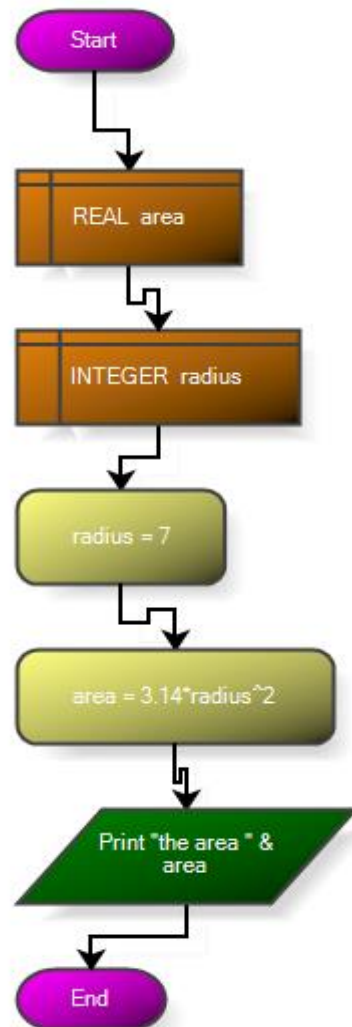


What it Does

A Declare Statement is used to create variables and arrays. These are used to store data while the program runs.

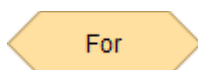
Example

The example, to the right, declares two variables: area (which stores real numbers) and radius (which stores integers). It then uses an Assignment Statement to set the 'radius' to 7. Finally, it computes the area of a circle and stores the result in 'area'.



- For Shape

Default Appearance



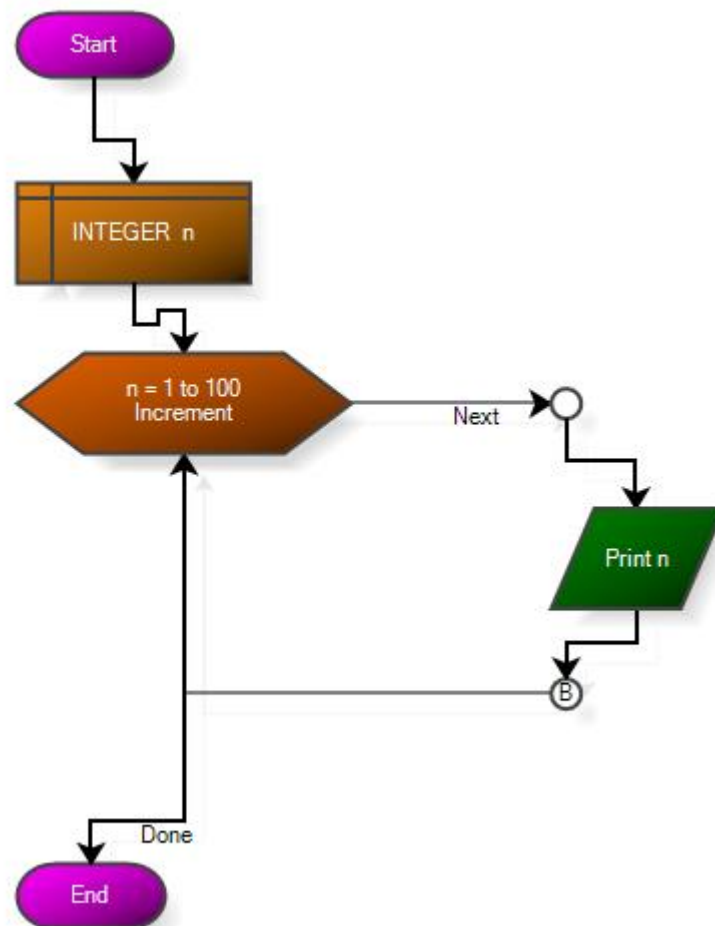
What it Does

For Loops increment a variable through a range of values. This is a common, useful, replacement for a While Statement.

Example

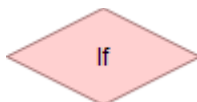
The example, to the right, prints the numbers from 1 to 100. The loop executes 100 times. The value of 'n' starts at 1 and increases by 1 each time the loop executes. The loop ends when 'n' reaches 100.

....



- If Shape

Default Appearance



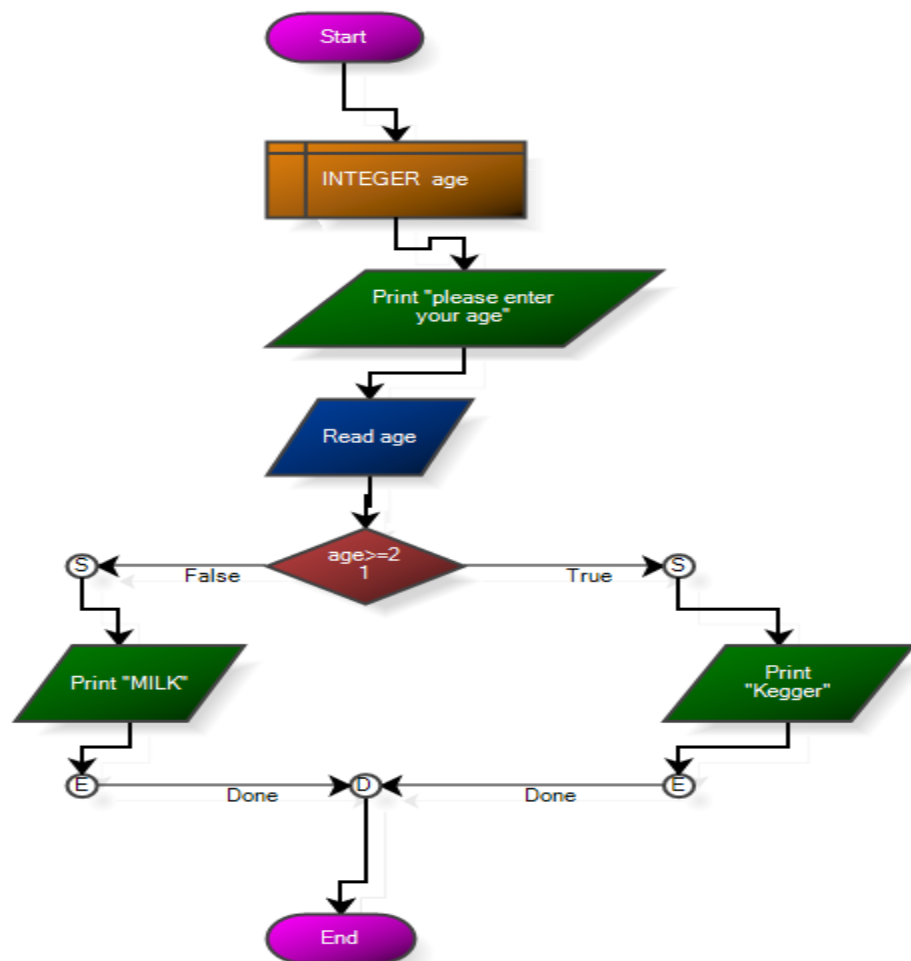
What it Does

An If Statement checks a Boolean expression and then executes a true or false branch based on the result.

Example

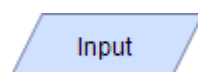
The example, to the right, declares an integer called 'age'. It then reads the age from the keyboard.

Finally, an If Statement checks if the age is greater than or equal to 18. Based on this, it either takes the false branch and displays "Sorry, not yet", or takes the true branch and displays "Go vote!"



- Input Shape

Default Appearance

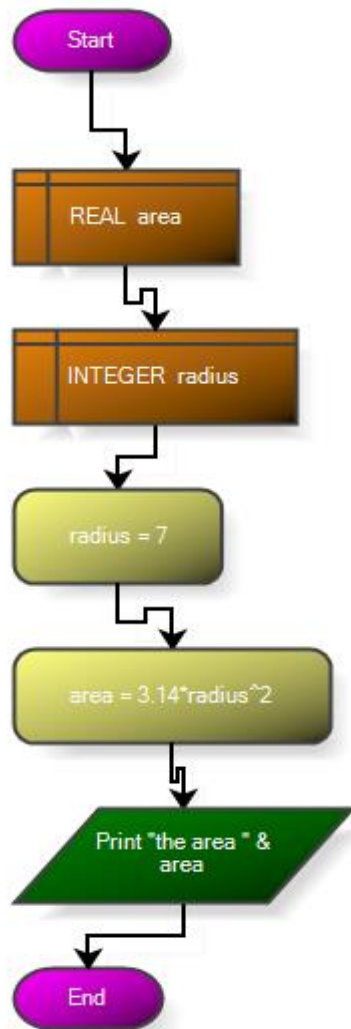


What it Does

An Input Statement reads a value from the keyboard and stores the result in a variable.

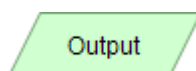
Example

The example, to the right, creates two variables: 'area' and 'radius'. It then uses an Input Statement to read the radius from the keyboard. A final Output Statement then displays the result.



- Output Shape

Default Appearance

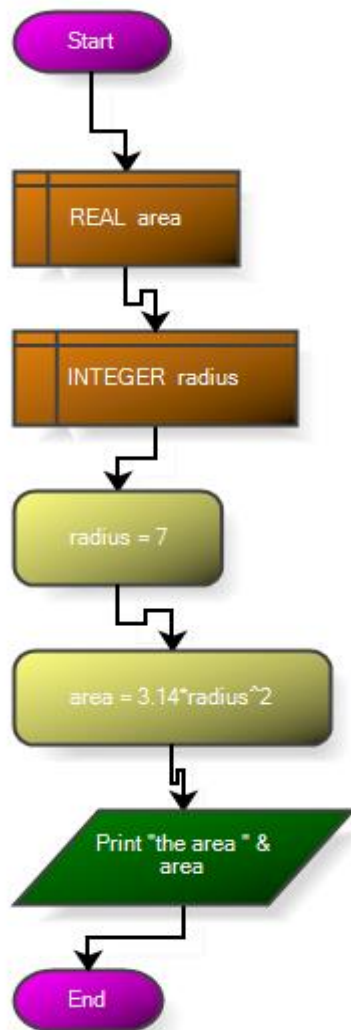


What it Does

An Output Statement evaluates an expression and then displays the result on the screen.

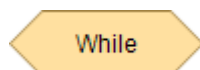
Example

The example, to the right, creates two variables: 'area' and 'radius'. It then uses an Input Statement to read the radius from the keyboard. A final Output Statement then displays the result.



- While Shape

Default Appearance

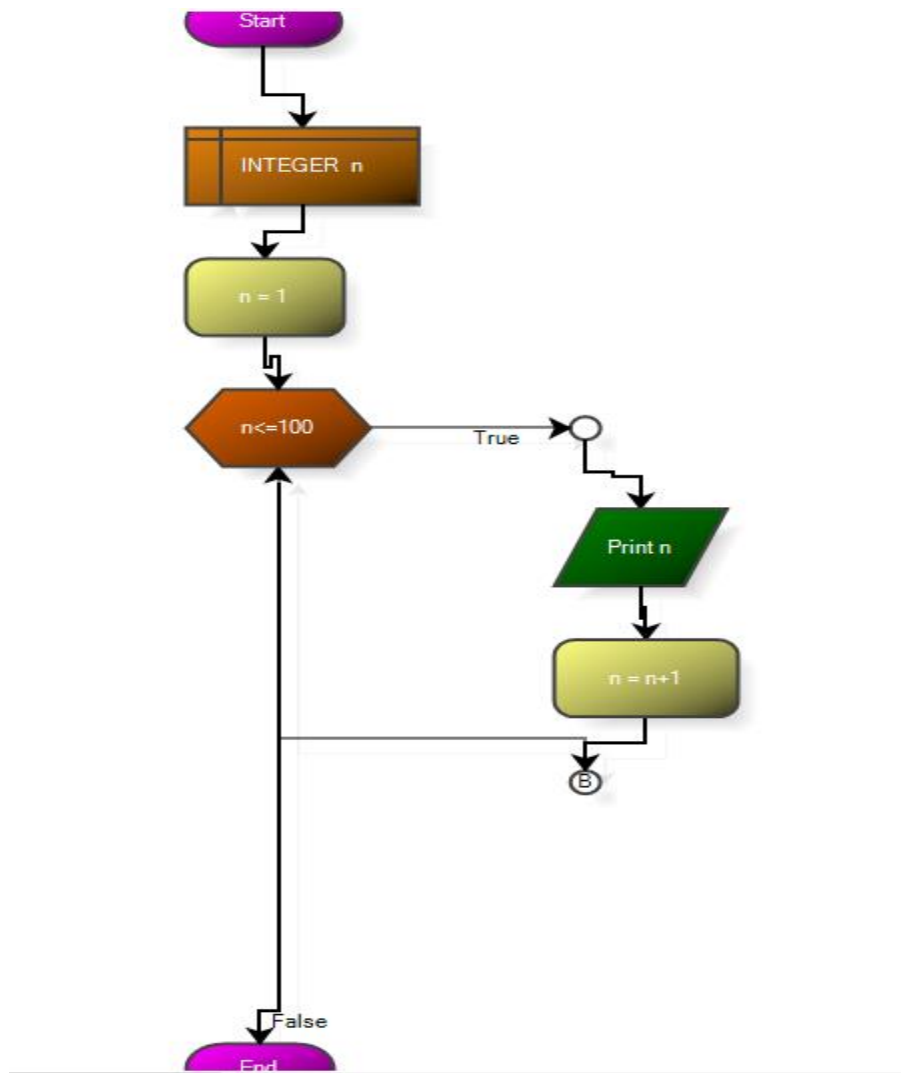


What it Does

A While Loop evaluates a Boolean expression and, if true, executes statements. It rechecks the expression and loops until it is false.

Example

The example, to the right, prints the numbers from 1 to 100. The assignment statement " $n = n + 1$ " increments the variable 'n' by 1 for each iteration of the loop.



2.3 Data Types

- Integer Data Type

The Integer data type is one of the most commonly used types in programming. An integer can store a positive or negative whole number, but can't store fractional values. So, it can store values such as 5, 42, 1947, but can't store numbers such as 3.2, 4.5, etc...

If a number with a fractional value is stored into a integer, the fractional value will be discarded. Hence, if 3.2 is stored into an integer, it will only retain 3.

Integer
1947

- Real Data Type

The Real data type can store any number - both whole numbers and ones with fractional values. In many languages, this is called a "double" after the implementation standard known as "double-precision floating point".

Real
1.618

- String Data Type

The String data type is used to store any textual data. This includes words, letters, or anything else you would send in a text message. In programming, the text is delimited with double quotes. For example: "CSU, Sacramento", "computer", and "Year 1947" are all strings.

String
Sacramento State

- Boolean Data Type

The Boolean Data Type can store either "true" or "false". These are the basis of decision making in a computer program.

Boolean
True

Summary Chart

Data Type	Notes
-----------	-------

Boolean	Stores either Boolean true or false
Real	Stores a real number.
Integer	Stores an integer number.
String	Stores textual data.

1.4 Identifiers

Any time you define a function or variable, it is given a unique name called an "identifier". To prevent identifiers from being confused with other items in an expression, they must follow a naming convention. Every programming language has one and it is fairly consistent from language to language.

In FlowchartToCode, identifiers must adhere to the following rules:

- They must start with a letter.
- After the first letter, the identifier can contain addition letters or numbers.
- Spaces are not allowed.
- They cannot be reserved words or words already defined by FlowchartToCode

Also note:

- Languages such as Visual Basic and C also allow the underscore character "_". FlowchartToCode, however, does not allow it.
- Identifiers are not case-sensitive.

The following are some simple example identifiers.

Valid Identifiers	Notes
X	Identifiers can be single letter.
Name	
noun2	Numbers are allowed after the first letter

2.5 Shapes Properties

When click on assign shape show you this form.

Assign Properties

Assign

An input statement reads a value from the keyboard and stores the result in a variable.

Variable name:

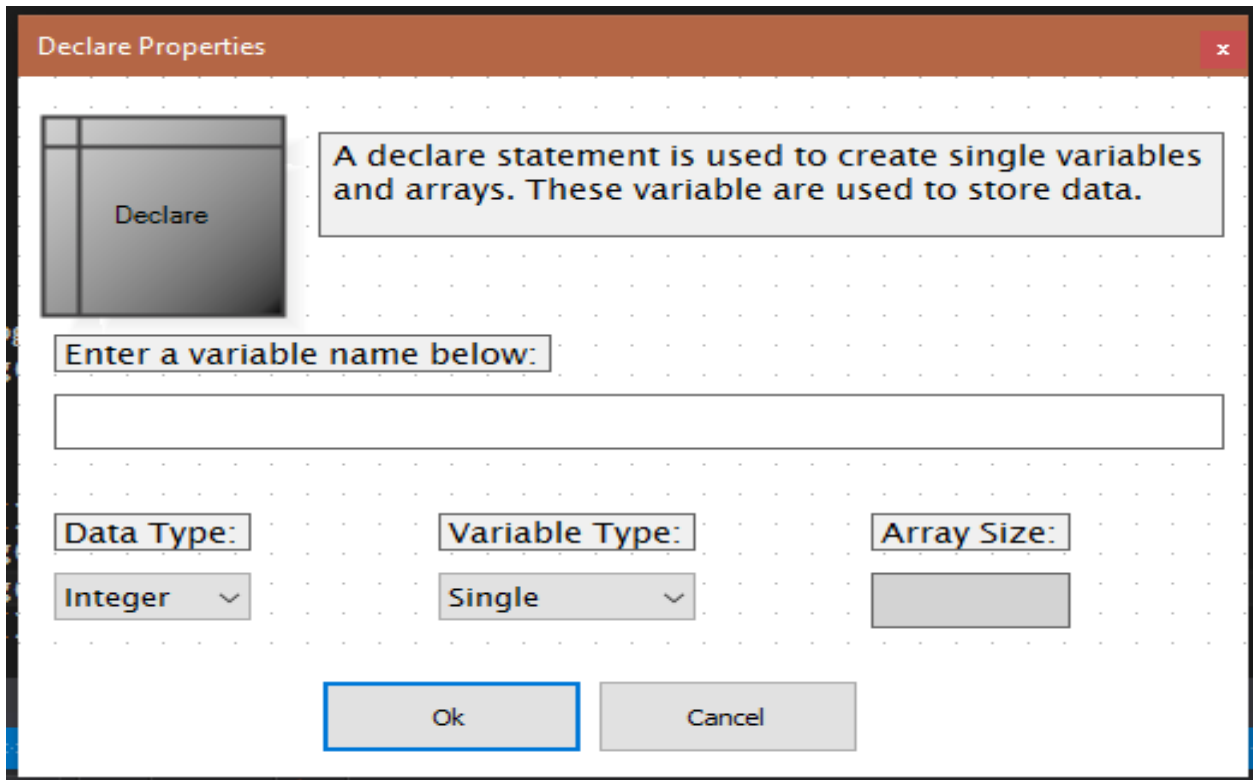
=

Value expression:

Ok

Cancel

When click on Declare shape show you this form



The 'Declare Properties' dialog box features a title bar with a close button. On the left is a 'Declare' shape icon. A text box explains: 'A declare statement is used to create single variables and arrays. These variable are used to store data.' Below this is a label 'Enter a variable name below:' followed by a text input field. Further down are three sections: 'Data Type:' with a dropdown menu showing 'Integer', 'Variable Type:' with a dropdown menu showing 'Single', and 'Array Size:' with an empty text input field. At the bottom are 'Ok' and 'Cancel' buttons.

Declare Properties

Declare

A declare statement is used to create single variables and arrays. These variable are used to store data.

Enter a variable name below:

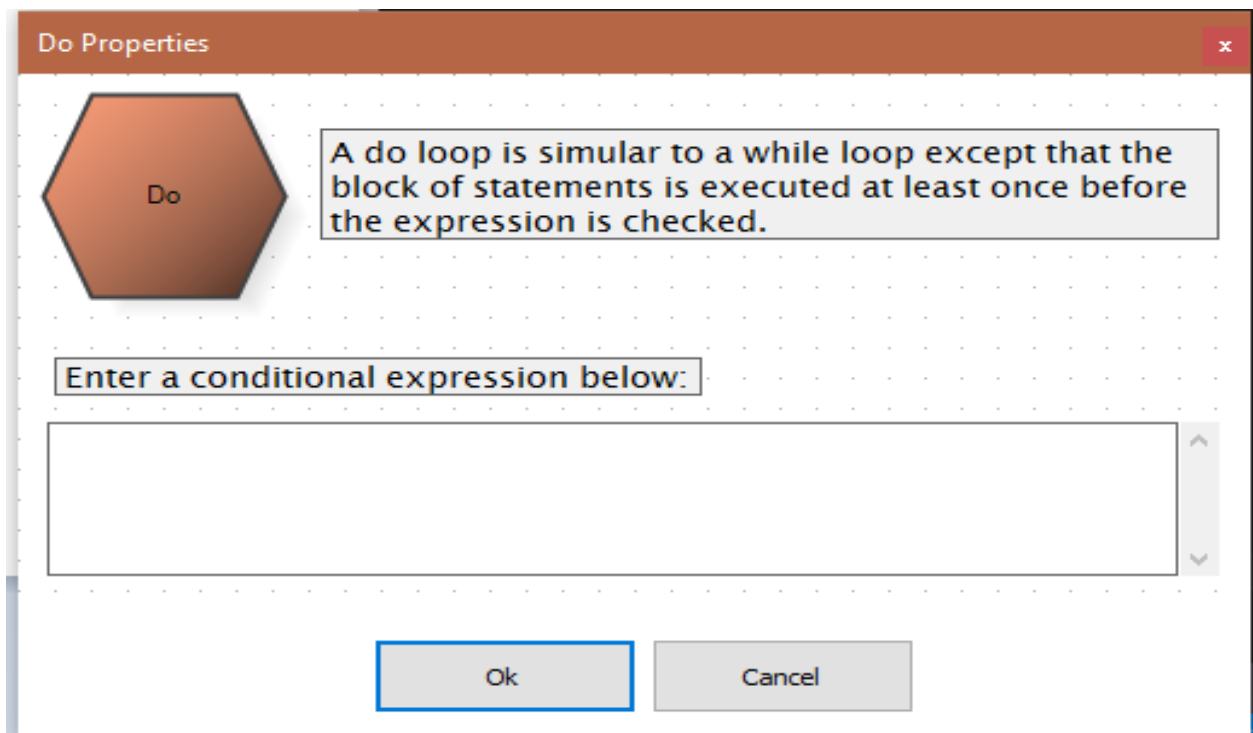
Data Type: Integer

Variable Type: Single

Array Size:

Ok Cancel

When click on Do shape show you this form.



The 'Do Properties' dialog box has a title bar with a close button. On the left is a 'Do' shape icon. A text box explains: 'A do loop is similar to a while loop except that the block of statements is executed at least once before the expression is checked.' Below this is a label 'Enter a conditional expression below:' followed by a large text input field with a vertical scrollbar. At the bottom are 'Ok' and 'Cancel' buttons.

Do Properties

Do

A do loop is similar to a while loop except that the block of statements is executed at least once before the expression is checked.

Enter a conditional expression below:

Ok Cancel

When click on For shape show you this form.

Decision Properties

For

A for loop increments or decrements a variable through a range of values.

Enter a loop variable name below:

Enter a start value below:

Enter an end value below:

Step by:

Behaviour:

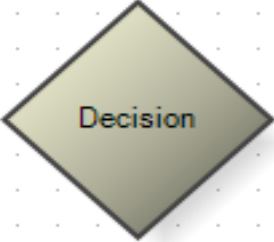
Increment

Ok

Cancel

When click on IF shape show you this form.

Decision Properties



An if statement checks a boolean expression then executes a true or false branch based on the result.

Enter a conditional expression below:

Ok

Cancel

When click on Input shape show you this form.

Input Properties

Input

An input statement reads a value from the keyboard and stores the result in a variable.

Enter a variable name below:

Ok

Cancel

When click on Output shape show you this form.

Output Properties

Output

An output statement evaluates an expression and then displays the result to the screen.

Enter an expression below:

Ok

Cancel

When click on While shape show you this form.

While

A while loop checks a boolean expression and if true it executes a block of statements . It rechecks the expression and loops until it is false.

Enter a conditional expression below:


Ok

Cancel

Chapter 3: Implementation

Project Save Feature

- **Our Program** can save the project you're working on by clicking on the save

icon  a Folder Picker Dialog opens to choose the desired folder to save your project, the project will be saved in an XML format

- **Xml Format ::**

The Xml Version Used is version="1.0"

The Encoding is "utf-8"

The Xml Start with tag that have the name <FlowChart>

All the project Nodes are encapsulated between this tag

Nodes start with the start node and followed by the nodes in the

Main block if nodes contained in another block like loop block

These Nodes are encapsulated inside this node

The Nodes are written one by one and each one has it's on Attributes.

- **The Nodes ::**

Terminal Nodes (Start , End) ::

Tags

<Start (Attributes)> , <End (Attributes)>

Attributes

- o Location Attribute :: indicates the node location

(Location Attribute)

The location Attribute is written in this way "x,y" => "50,200" to indicate the coordinates of the node on the Model

Encapsulated Tags

None

Assignment Node ::

Tag

< Assign (Attributes)>

Attributes

- o Location Attribute :: indicates the node location
- o Statment Attribute :: include the statment

Encapsulated Tags

None

Input Node ::

Tag

< Input(Attributes)>

Attributes

- Location Attribute :: indicates the node location
- Statment Attribute :: include the statment

Encapsulated Tags

None

Output Node ::

Tag

< Output(Attributes)>

Attributes

- Location Attribute :: indicates the node location
- Statment Attribute :: include the statment

Encapsulated Tags

None

While Node ::

Tag

< While (Attributes)>

Attributes

- Location Attribute :: indicates the node location
- Statment Attribute :: include the statment
- True_End_Location :: Indicates where the true block ends

Encapsulated Tags

- True Tag :: this tag includes all the other nodes that are drawn inside the true track of the while node

<True>

<Xnode>.....</Xnode>

</True>

For Node ::

Tag

< For (Attributes)>

Attributes

- Location :: indicates the node location
- Statment :: include the statment
- True_End_Location :: Indicates where the true block ends
- Variable :: indicates the name of counter variable
- StartVal :: the value that the variable begin with
- EndVal :: the value that the variable should reach to
- Direction :: indicates wether the variable is increasing or decreasing
- StepBy :: Indicates the step value that the counter variable increse or decrease by

Encapsulated Tags

- True Tag :: this tag includes all the other nodes that are drawn inside the true track of the for node

<True>

<Xnode>.....</Xnode>

</True>

IF Node ::

Tag

< If (Attributes)>

Attributes

- Location :: indicates the node location
- Statment :: include the statment
- True_End_Location :: Indicates where the true block ends

Encapsulated Tags

- True Tag :: this tag includes all the other nodes that are drawn inside the true track of the if node

<True>

<Xnode>.....</Xnode>

</True>

IFElse Node ::

< IfElse (Attributes)>

Attributes

- Location :: indicates the node location
- Statment :: include the statment
- True_End_Location :: Indicates where the true block ends
- False_End_Location :: Indicates where the true block ends
- Middle_Location :: Indicates where should both true and false tracks meets

Encapsulated Tags

- True Tag :: this tag includes all the other nodes that are drawn inside the true track of the while node

<True>

<Xnode>.....</Xnode>

</True>

- False Tag :: this tag includes all the other nodes that are drawn inside the false track of the If Else node

< False >

<Xnode>.....</Xnode>

</False >

Declare Node ::

Tag

< Declare (Attributes)>

Attributes

- Location Attribute :: indicates the node location
- Statment Attribute :: include the statment
- Variable_Name :: the name of the declared varaible
- Single :: if the variable is single or array
- Size :: size of the array if it's not a single variable
- Variable_Type :: the data type of the declared variable

Encapsulated Tags

None

Project_Saver Class ::

Description::

This Class is responsible for saving project on Xml Format An Object of this class is instantiated and called by the Controller Class

Class Diagram::

F

Important Methods ::

GetBlockXml

This Method Takes Two Nodes the node to start with and the node to end with because each Block on the Diagram has A node to begin with and another to end with for example the main block

Starts with the Start Node and ends with the End Node

It loops between those two nodes and checks for the node type and then it calls putAttribute method to put the proper attributes

```
private string getBlockXML(BaseNode node, BaseNode endBlockNode)
{
    indentation += 4;
    StringBuilder sb = new StringBuilder("\r\n");

    while (node != endBlockNode)
    {
        if (!(node is HolderNode )) {
            sb.Append("\r\n");
            sb.Append(' ', indentation);
            sb.Append("<" + node.Name + putAttributes(node)+ ">");
        }
        if (node is HolderNode );
        else if (node is DecisionNode)
        {
            if (node is IfElseNode)
            {
                IfElseNode ifNode = (IfElseNode)node;
                indentation += 4;
                sb.Append("\r\n");
                sb.Append(' ', indentation);
                sb.Append("<True>");
                sb.Append(getBlockXML(ifNode.TrueNode, ifNode.BackNode));
                sb.Append("</True>");
                sb.Append("\r\n");
                sb.Append(' ', indentation);
                sb.Append("<False>");
                sb.Append(getBlockXML(ifNode.FalseNode, ifNode.BackfalseNode));
                sb.Append("</False>");
                indentation -= 4;
            }
        }
    }
}
```

```

        else
        {
            DecisionNode loopNode = (DecisionNode)node;
            indentation += 4;
            sb.Append("\r\n");
            sb.Append(' ', indentation);
            sb.Append("<True>");
            sb.Append(getBlockXML(loopNode.TrueNode, loopNode.BackNode));
            sb.Append("</True>");
            indentation -= 4;
            sb.Append("\r\n");
        }
    }
    else
    {
        sb.Append("\r\n");
    }

    if (!(node is HolderNode )) {
        sb.Append(' ', indentation);
        sb.Append("</" + node.Name + "> \n");
    }
    node = node.OutConnector.EndNode;

}
sb.Append("\r\n");
indentation -= 4;
sb.Append(' ', indentation);

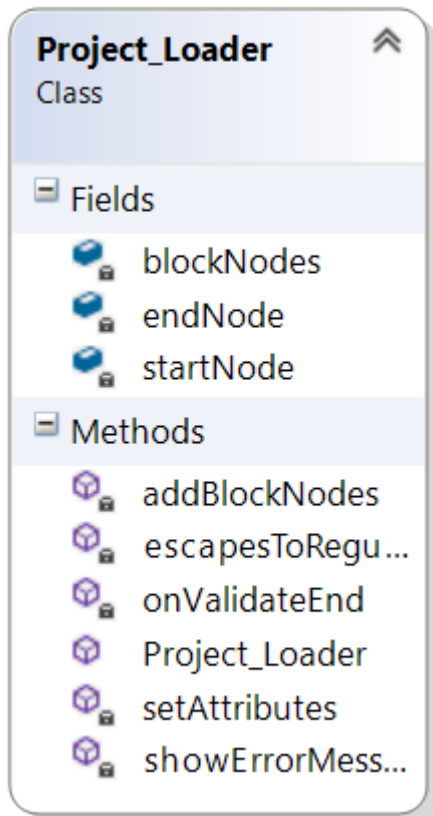
return sb.ToString();

```

PutAttributes Method

This Method is responsible for adding the attributes that we discussed before to each tag

Project Loader Class



Description ::

This Class used to load any previously saved Project by getting the file path it starts to read all the XML code and convert each tag to it's correspondent Node

Important Methods :::

AddBlockNodes Method

This Method is responsible for replasing tags in the same block to a block of nodes it takes a list of XmlNodeNodes

```

private void addBlockNodes(XmlNodeList list, BaseNode lastNode, BaseNode parentNode)
{
    foreach (XmlNode node in list)
    {
        BaseNode newNode = null;
        if (node.Name.Equals("End"))
        {
            newNode = endNode;
        }
        else if (node.Name.Equals("Start"))
        {
            newNode = startNode;
        }
        else if (node.Name.Equals("Assign"))
        {
            newNode = new AssignNode();
        }

        }
        else if (node.Name.Equals("Declare"))
        {
            newNode = new DeclareNode();

            else if (node.Name.Equals("For"))
            {
                newNode = new ForNode();
                setAttributes(node, newNode);
                addBlockNodes(node.FirstChild.ChildNodes, ((DecisionNode)newNode).TrueNode, newNode);
            }
            else
            {
                showErrorMessage("Error not valid file");
            }
        }

        if (!(node.Name.Equals("Start") || node.Name.Equals("End")))
        {
            BaseNode oldNode = lastNode.OutConnector.EndNode;
            lastNode.OutConnector.EndNode = newNode;
            newNode.OutConnector.EndNode = oldNode;
            newNode.addToModel();
            newNode.ParentNode = parentNode;
        }

        lastNode = newNode;
        blockNodes.Add(new Pair(node, newNode));
        setAttributes(node, newNode);
    }
}

```

SetAttributes Method

This Method is responsible for adding the attributes that we discussed before from each tag to the actual node in the Model

```

private static void setAttributes(XmlNode node, BaseNode newNode)
{
    string statement = node.Attributes["Statment"]?.InnerText;
    statement = escapesToRegular(node.Attributes["Statment"]?.InnerText);
    string location = node.Attributes["Location"]?.InnerText;
    string[] true_end_location = node.Attributes["True_End_Location"]?.InnerText.Split(',');
    string[] false_end_location = node.Attributes["False_End_Location"]?.InnerText.Split(',');
    string[] mid_location = node.Attributes["Mid_Location"]?.InnerText.Split(',');

    newNode.Statement = statement;

    if (location != null)
    {
        string[] points = location.Split(',');
        newNode.NodeLocation = new System.Drawing.PointF((float)Double.Parse(points[0]), (float)Double.Parse(points[1]));
    }
    if (newNode is IfElseNode)
    {
        IfElseNode ifElseNode = (IfElseNode)newNode;
        ifElseNode.BackNode.NodeLocation = new System.Drawing.PointF((float)Double.Parse(true_end_location[0]), (float)Double.Parse(true_end_location[1]));
        ifElseNode.BackfalseNode.NodeLocation = new System.Drawing.PointF((float)Double.Parse(false_end_location[0]), (float)Double.Parse(false_end_location[1]));
        ifElseNode.MiddleNode.NodeLocation = new System.Drawing.PointF((float)Double.Parse(mid_location[0]), (float)Double.Parse(mid_location[1]));
    }
}
else if (newNode is DeclareNode)
{
    string variable_name = node.Attributes["Variable_Name"].InnerText;
    string variable_type = node.Attributes["Variable_Type"].InnerText;
    string single = node.Attributes["Single_Variable"].InnerText;
    string size = node.Attributes["Size"].InnerText;
    DeclareNode declareNode = (DeclareNode)newNode;
    declareNode._Var.VarName = variable_name;
    declareNode._Var.Size = Int32.Parse(size);
    declareNode._Var.Single = Boolean.Parse(single);
    switch (variable_type)
    {
        case "INTEGER":
            declareNode._Var.VarType = DeclareNode.Variable.Data_Type.INTEGER;
            break;
        case "REAL":
            declareNode._Var.VarType = DeclareNode.Variable.Data_Type.REAL;
            break;
        case "BOOLEAN":
            declareNode._Var.VarType = DeclareNode.Variable.Data_Type.BOOLEAN;
            break;
        case "STRING":
            declareNode._Var.VarType = DeclareNode.Variable.Data_Type.STRING;
            break;
    }
}
}

```

Nodes

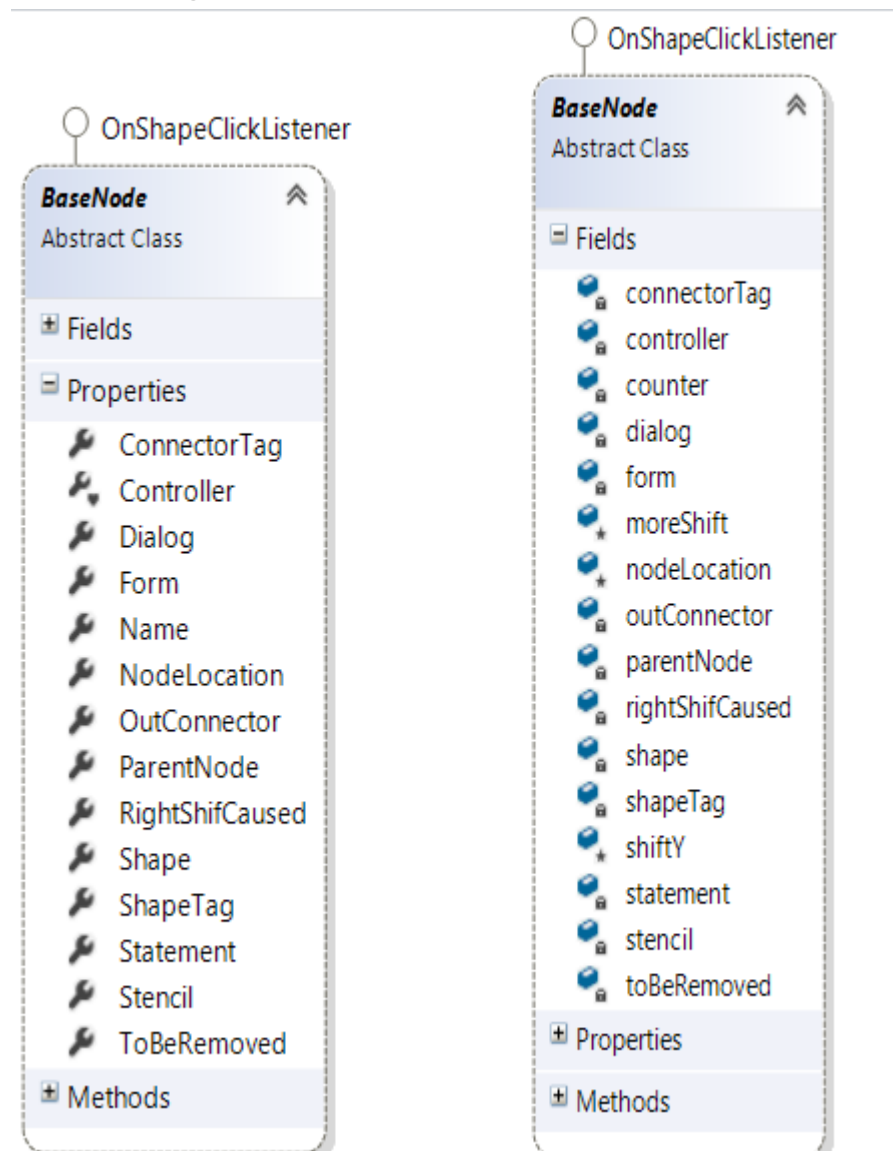
As we discussed before we have different nodes that user can use to Implement the flowchar diagram each node of those has it's own Class to implement it's features and properities

BaseNode Class::

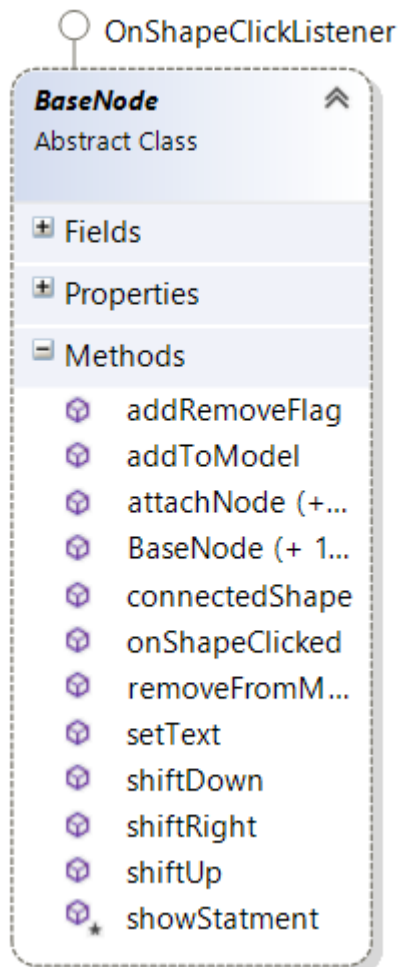
Description::

BaseNode is an abstract class implement the basic features that all nodes should have

Block Diagram



Important Methods



AttachNode Method::

This method takes another `BaseNode` object then it adds it as it's next node and make the node that was previously attached to it to be attached to this new node

```

public void attachNode(BaseNode newNode)
{

    if (this is TerminalNode && newNode is TerminalNode ||
        this is HolderNode && newNode is HolderNode)
    {
        if (newNode.connectedShape() == null)
        {
            //do nothing
        }
        if (this.connectedShape() == null)
        {
            //donothing
        }
        OutConnector.EndNode = newNode;
        newNode.NodeLocation = this.NodeLocation;
        newNode.shiftDown(0);
        return;
    }

    BaseNode oldOutNode = OutConnector.EndNode;
    OutConnector.EndNode = newNode;
    //newNode.OutConnector.EndNode = oldOutNode;
    float x = this.NodeLocation.X;
    float y = oldOutNode.NodeLocation.Y;

    if (this.NodeLocation.X != oldOutNode.NodeLocation.X)
    {
        if (this.ParentNode is IfElseNode
            && this==(this.parentNode as IfElseNode).MiddleNode) {
            x = this.parentNode.NodeLocation.X;
        }
        else if (this is HolderNode)
            x = oldOutNode.NodeLocation.X;
        else if (oldOutNode is HolderNode)
            x = this.NodeLocation.X;
    }
    newNode.OutConnector.EndNode = oldOutNode;
    oldOutNode.shiftDown(0);
    newNode.NodeLocation = new PointF(x, y);
    controller.balanceNodes(newNode);

}

```

ShiftDown Method ::

After attaching any node the nodes below it needs to be shifted down thus the new node invoke shiftDown which shifts it and the nodes after it

```
virtual public void shiftDown(float moreShift = 0)
{
    // this.moreShift = moreShift;

    if (connectedShape() != null)
        NodeLocation = new PointF(connectedShape().Location.X, connectedShape().Location.Y + shiftY);

    if (!(this is HolderNode) && OutConnector.EndNode != null)
        OutConnector.EndNode.shiftDown(moreShift);
}
```

Shift Up ::

After removing any node the nodes below it needs to be shifted down thus the new node invoke shiftDown which shifts it and the nodes after it up

```
NodeLocation = new PointF(NodeLocation.X, NodeLocation.Y - offsetY);
if (OutConnector.EndNode == null || this is DecisionNode)
    return;
if (this is HolderNode) //what about middleNode shift
{
    //to decide shifting middle node or not
    if (this.ParentNode is IfElseNode)
    {
        IfElseNode pNode = this.ParentNode as IfElseNode;
        PointF preLocation = pNode.MiddleNode.NodeLocation;
        pNode.balanceHolderNodes();
        if (pNode.MiddleNode.NodeLocation.Y == preLocation.Y)
            return; //thus don't shift the node after parent node
    }
}
```

ConnectorNode Class::

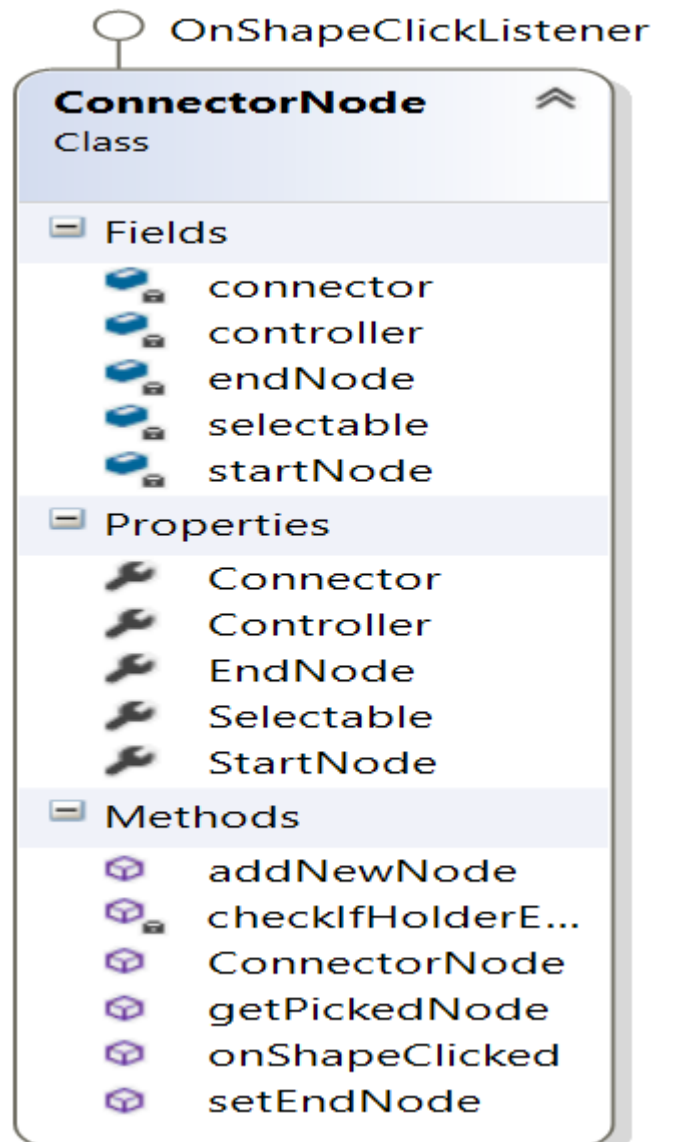
Description::

This class connects between a pair of node one node is considered the start node the other is considered the end node .All the Nodes have a member

variable called OutConnector which represents the connection between the node and the next node

Connector node is used also to add nodes by clicking on the connector shape a picker dialog is opened to choose another node to be added as the EndNode of this connector and the old end node becomes the end node of the new node.

Class Diagram ::



Important Methods ::

AddNewNode Method ::

This methods is invoked after the connector is clicked and new node is choosed from the picker dialog to be added

1 reference | ohefny, 6 days ago | 1 author, 1 change

```
public void addNewNode(BaseNode toAttachNode)
```

```
{
```

```
    if (toAttachNode != null)
```

```
    {
```

```
        if (checkIfHolderExist() != null)
```

```
        {
```

```
            toAttachNode.ParentNode = (checkIfHolderExist()).ParentNode;
```

```
            toAttachNode.ParentNode.attachNode(toAttachNode, this);
```

```
        }
```

```
    else
```

```
    {
```

```
        toAttachNode.ParentNode = startNode.ParentNode;
```

```
        startNode.attachNode(toAttachNode);
```

```
    }
```

```
    toAttachNode.addToModel();
```

```
}
```

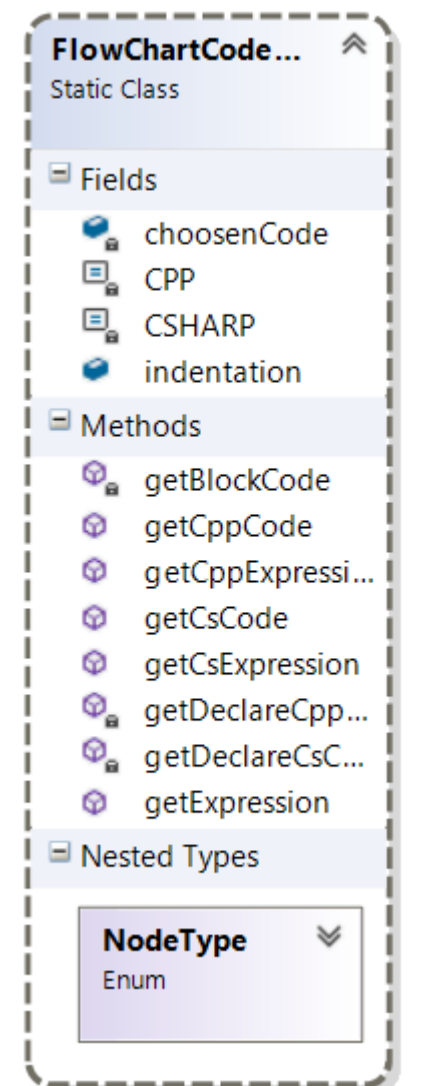
```
}
```

Convert To Code Feature ::

The Best feature in our project is the ability to convert the flowchart to a code in one of two programming language (Csharp , Cplusplus)

FlowChartCodeConverter Class

Block Diagram ::



Description::

This Class is responsible for converting the flowchart diagram to code ..Controller invokes this class with the desired Language

Important Methods ::

GetBlockCode Method::

The Method takes the start and the end node of different blocks in the model and convert it to the coresspondent code and encapsulate it between { }

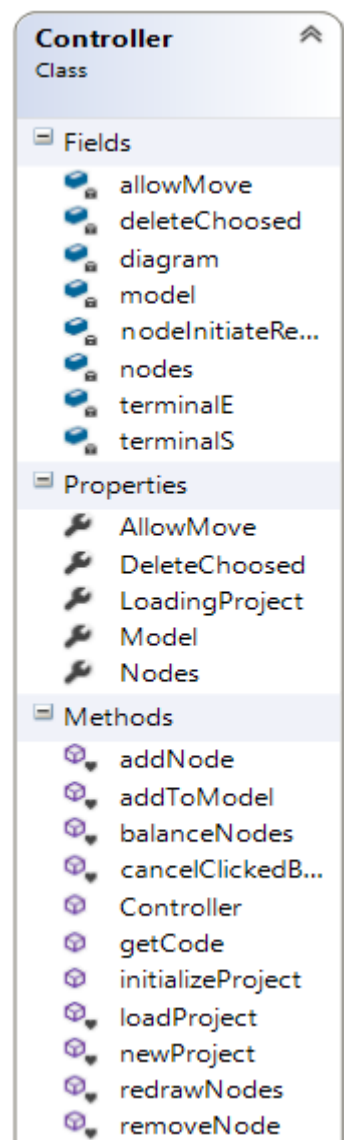
```
BaseNode node = startBlockNode;

while (node != endBlockNode) {
    if(!(node is HolderNode || node is TerminalNode))
        sb.Append(' ', indentation);
    if (node is HolderNode || node is TerminalNode) ;
    else if (node is IfElseNode)
    {
        IfElseNode ifNode = (IfElseNode)node;
        sb.Append(getExpression(NodeType.IFELSE, ifNode));
        sb.Append(getBlockCode(ifNode.TrueNode, ifNode.BackNode));
        sb.Append(' ', indentation);
        sb.Append("else");
        sb.Append(getBlockCode(ifNode.FalseNode, ifNode.BackfalseNode));
    }
    else if (node is IfNode)
    {
        IfNode ifNode = (IfNode)node;
        sb.Append(getExpression(NodeType.IF, ifNode));
        sb.Append(getBlockCode(ifNode.TrueNode, ifNode.BackNode));
    }
    ,
    else if (node is InputNode) {
        sb.Append(getExpression(NodeType.INPUT, node));
    }
    else if (node is OutputNode) {
        sb.Append(getExpression(NodeType.OUTPUT, node));
    }
    else if (node is AssignNode)
    {
        sb.Append(getExpression(NodeType.ASSIGN, node));
    }
    else if (node is DeclareNode)
    {
        sb.Append(getExpression(NodeType.DECLARE, node));
    }
    sb.Append(Environment.NewLine);
    node = node.OutConnector.EndNode;
}

indentation -= 4;
sb.Append(' ', indentation);
sb.Append("}\r\n");

return sb.ToString();
```

Controller Class ::



Description::

This class controls the application it basically the mind of the project it controls the model with it's shapes and lines, creates a new project , removes a node from the mode, add shapes and lines of any node to the model balance nodes and shifts nodes right and left based on the changes that happens on each node during the drawing

Important Methods ::

InitializeProject Method::

This Method clears the model from any nodes to start drawing new diagram

```
public void initializeProject()
{
    Model.Clear();
    Nodes.Clear();
    BaseNode.Controller = this;
    ConnectorNode.Controller = this;
    terminalS = new TerminalNode(TerminalNode.TerminalType.Start);
    terminalE = new TerminalNode(TerminalNode.TerminalType.End);
    terminalS.attachNode(terminalE);
    terminalE.ParentNode = terminalS;
    terminalS.addToModel();
    terminalE.addToModel();
}
```

AddToModel Method ::

This Method takes a Node as a parameter it draws the shapes and lines of this node to the Model

```
internal void addToModel(BaseNode toAddNode)
{
    if (Model == null)
    {
        throw new Exception("Model should be set before calling addToModel");
    }
    if (toAddNode.OutConnector.EndNode != null)
        Model.Lines.Add(toAddNode.ConnectorTag, toAddNode.OutConnector.Connector);
    if (toAddNode.Shape != null)
        Model.Shapes.Add(toAddNode.ShapeTag, toAddNode.Shape);
    addNode(toAddNode);
    if (toAddNode is IfElseNode && toAddNode.NodeLocation.X < 100)
        shiftNodesRight(toAddNode,true); //to be replaced by controller
    if(toAddNode is DecisionNode)
        Model.Lines.Add((toAddNode as DecisionNode).TrueConnector.Connector);
    if (toAddNode is IfElseNode)
    {
        Model.Lines.Add((toAddNode as IfElseNode).FalseConnector.Connector);
    }
}
```

RemoveNode Method ::

This Method takes a Node as a parameter to remove the shapes and lines of this node from the model

```

internal void removeNode(BaseNode toRemoveNode)
{

    if (Model == null)
    {
        throw new Exception("Model should be set before calling addToModel");
    }

    foreach (BaseNode node in Nodes)
    {
        BaseNode nextNode = node.OutConnector.EndNode;
        if (nextNode == toRemoveNode && node.OutConnector.EndNode != node.ParentNode) //problem for back
        {
            node.OutConnector.EndNode = nextNode.OutConnector.EndNode;
            node.OutConnector.EndNode.shiftUp(node.OutConnector.EndNode.NodeLocation.Y - toRemoveNode.No
            break;
        }
    }

    for (int i = 0; i < Nodes.Count; i++) {
        if (Nodes[i].ToBeRemoved) {
            //Nodes[i].OutConnector.EndNode.shiftUp(Nodes[i].OutConnector.EndNode.NodeLocation.Y - toRem
            Nodes.Remove(Nodes[i]);
            i--;
        }
    }
    redrawNodes();
}

```

BalanceNodes Method ::

Responsible for balancing node after adding and removing and changing shapes size

```

internal void balanceNodes(BaseNode newNode)
{
    BaseNode trackNode = null;

    do
    {
        if (trackNode == null)
            trackNode = newNode.ParentNode;
        else
            trackNode = trackNode.ParentNode;
        if (newNode is IfElseNode) {
            //this is the case when adding in the false part of ifelse that is right to main track
            if (trackNode.NodeLocation.X < newNode.NodeLocation.X
                && (newNode as IfElseNode).FalseNode.NodeLocation.X <= trackNode.NodeLocation.X + trackNode.
            {

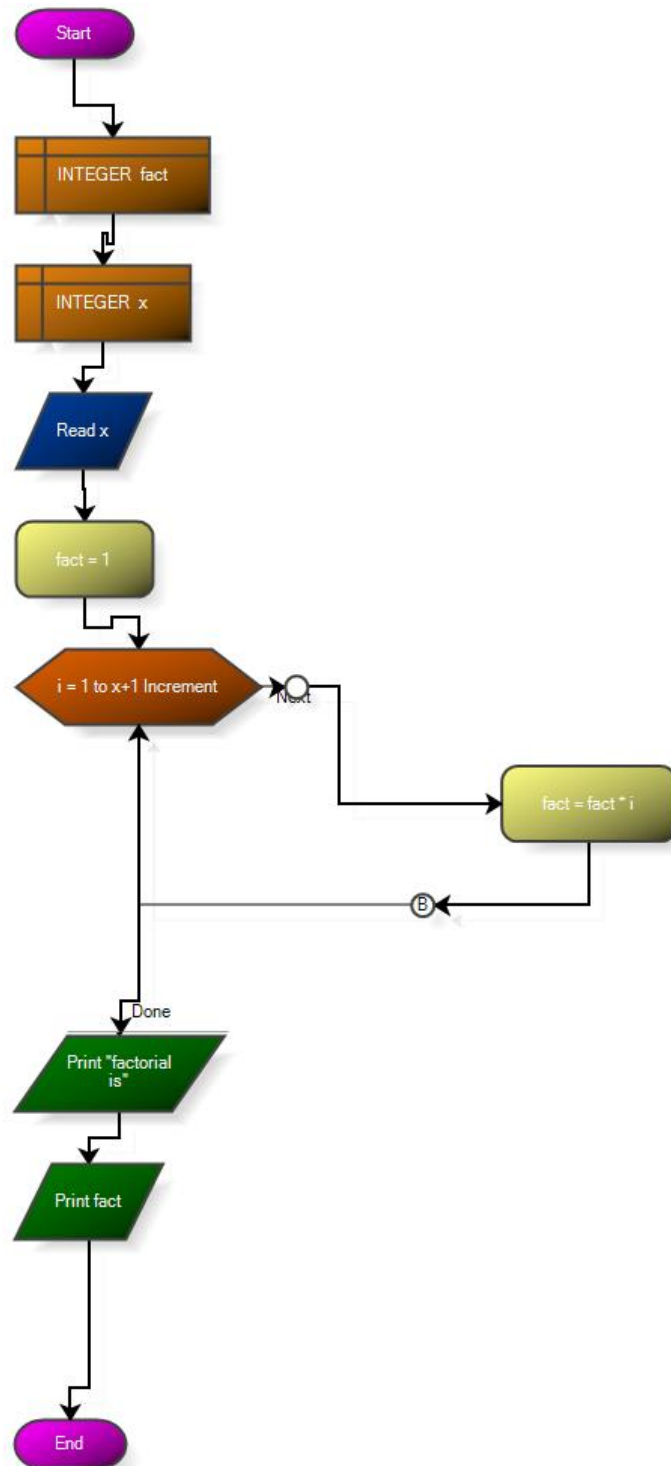
                addNode(newNode);
                shiftNodesRight(newNode, false);

            }
        }
        if (newNode is DecisionNode)
        {
            //this is the case when adding to the true part of Decision that is left to main track
            if (trackNode.NodeLocation.X > newNode.NodeLocation.X
                && (newNode as DecisionNode).TrueNode.NodeLocation.X > trackNode.NodeLocation.X)
            {
                shiftNodesRight(newNode, true);
            }
        }
    }
    else
    {
        //this is the case when the shape is overlapping with node in it's right
        if (trackNode.NodeLocation.X > newNode.NodeLocation.X
            && newNode.Shape.Width + newNode.NodeLocation.X > trackNode.NodeLocation.X)
        {
            shiftNodesRight(newNode, true, 100);
        }
    }
}
while (!(trackNode is TerminalNode)); //loop through parent and grandparents to see any conflict

```

Samples ::

Factorial Program in FlowChart



C-Sharp Code Generated For This Sample ::

```
using System;
using System.Collections.Generic;
using System.Text;
public class MyProgram
{
    public static void Main(String[] args)
    {
        {
            int fact;
            int x;
            input(out x) ;
            fact = 1;
            for ( int i = 1 ; i< x+1 ; i+=1 )
            {
                fact = fact * i;
            }

            Console.WriteLine("factorial is") ;
            Console.WriteLine(fact) ;
        }
    }
}
```

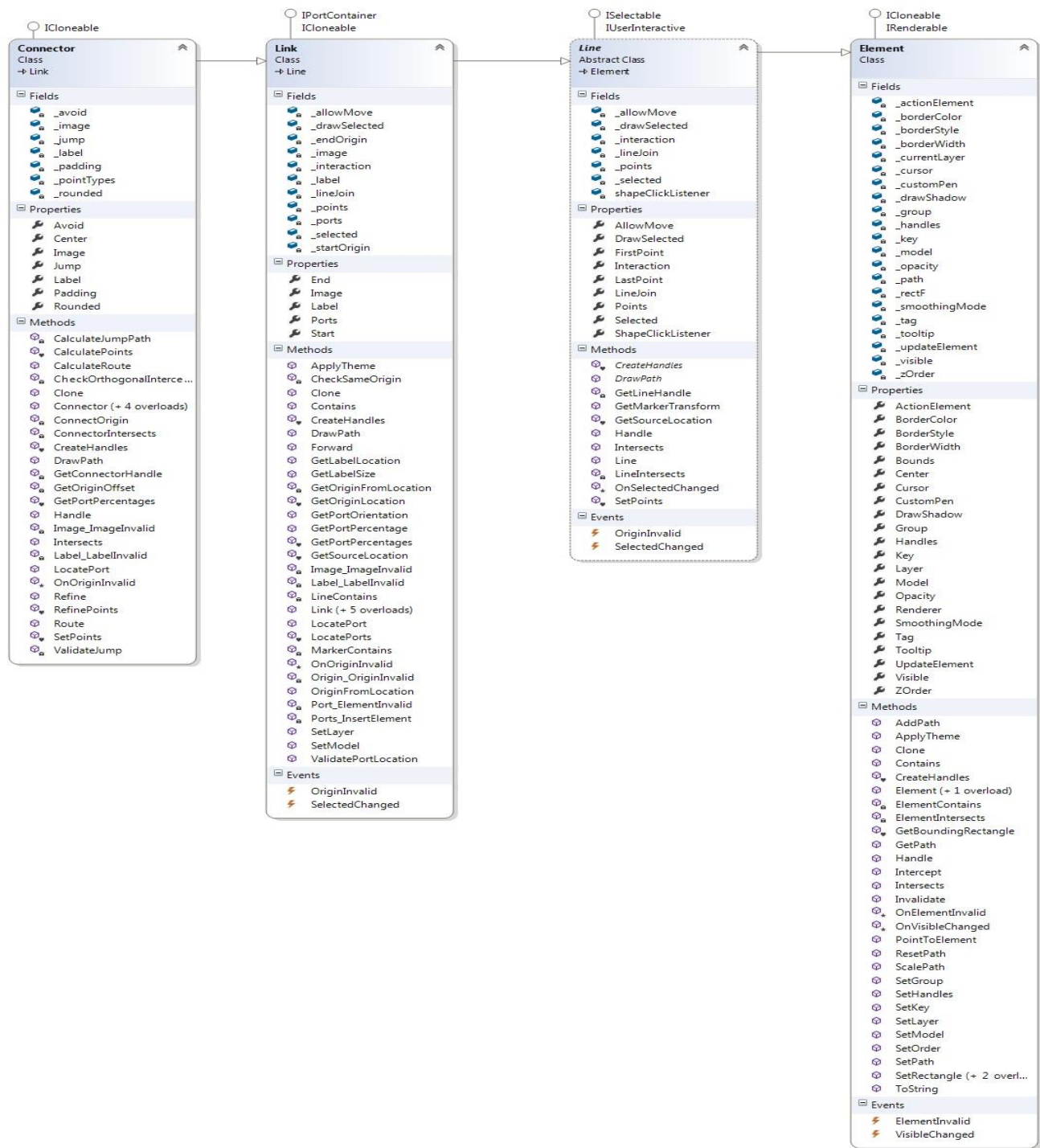
Project Save in Xml Format

```
<?xml version="1.0" encoding="UTF-8"?>
<FlowChart>
    <Start Location="149,10" Statment="Start"> </Start>
    <Declare Location="149,95" Statment="INTEGER fact" Size="0" Single_Variable="True" Variable_Type="INTEGER" Variable_Name="fact"> </Declare>
    <Declare Location="149,180" Statment="INTEGER x" Size="0" Single_Variable="True" Variable_Type="INTEGER" Variable_Name="x"> </Declare>
    <Input Location="149,265" Statment="x"> </Input>
    <Assign Location="149,350" Statment="fact = 1"> </Assign>
    - <For Location="149,435" Statment="int i = 1 ; i< x+1 ; i+=1" StepBy="1" EndVal="x+1" StartVal="1" Direction="Increment" Variable="i" True_End_Location="411.6,597.5">
        - <True>
            <Assign Location="471.6,512.5" Statment="fact = fact * i"> </Assign>
        </True>
    </For>
    <Output Location="149,690" Statment="&quot;factorial is&quot;"> </Output>
    <Output Location="149,775" Statment="fact"> </Output>
    <End Location="149,945" Statment="End"> </End>
</FlowChart>
```

Chapter 4: Open Crainiate Diagramming Framework

Important Classes Used in Our Program ::

Connector


















Controller

Controller

Class
















Fields

-  _checkBounds
-  _clipboard
-  _command
-  _commandFactory
-  _copy
-  _cut
-  _delete
-  _factory
-  _model
-  _paste
-  _redo
-  _redoStack
-  _roundPixels
-  _undo
-  _undoStack
-  _views

Properties

-  AllowCopy
-  AllowCut
-  AllowDelete
-  AllowPaste
-  AllowRedo
-  AllowUndo
-  CheckBounds
-  Clipboard
-  CommandFactory
-  Factory
-  Model
-  RedoStack
-  RoundPixels
-  UndoStack
-  Views

Methods

-  BoundsCheck (+ 2 overloads)
-  CanAdd
-  CanDelete
-  CanDock
-  CloneElement
-  Controller
-  ExecuteCommand
-  Invalidate
-  RedoCommand
-  Refresh
-  Resume
-  SelectElements (+ 1 overload)
-  SnapToLocation
-  Suspend
-  UndoCommand

