◆ Content

◆ Abstract

Programming, a very interesting skill and very important area. Almost every device around us today is related - in a way or another - to programming. For instance, Computers, Smartphones, TVs, Cameras, Calculates, Satellite, Airplanes, Some cars, Cameras … and so on and so forth. So Programming is a very important thing in our life today, but one main drawback of programming is that it is somehow complicated.

It is not very easy to learn the concept of programming at first, or the idea of algorithms, or the flow of the serial or parallel statements. More research in this field has revealed that the weakness of problem-solving skills is because of the complexities associated with the development environment and the language syntax that novices use. All that leads to create an easier way to introduce the novice to Programming.

We as human beings are able to deal with Images and visualizations in better way than written words. For example, it is much easier for us to remember image of objects with specific colors than write down the object's name and the name of its color. So, this is why the idea of visualizing algorithms (or iconic programming) was initialized. Many available software support tools for introducing novices to programming are subsequently based on a flowchart which acts as a visual aid in programming to help increase problem-solving skills.
This paper is showing a review of some programming environments based on an iconic notation is carried out with a focus on the support provided for introducing novices to programming.

◆ Keywords

- Flowchart
- Flowchart-based programming environments
- Novice programmers
- Problem-solving skills.

◆ Fundamentals Technologies used in Iconic Programming Environment Based

Programming environments based on an iconic notation use some kind of graphical/symbolic representation for data types, variables and control structures. In these environments, coding is accomplished with a syntax-directed editor that provides templates and menus with syntactically correct choices for every incomplete part of a program.

Such templates can be based either on textual representations or on some kind of graphical/symbolic representation. The type of templates used distinguishes syntax-directed editors in structure editors and iconic programming languages respectively, with the latter being the focus of this article.

The main goal of syntax editors is to use a programming language to implement an algorithm for solving a problem without having to be concerned about syntactic details. This fact has a positive impact on dealing with the problem of focusing on low-level details, such as learning how to format programs so that they are readable, learning how to use clumsy text editors and learning how to manage the software life cycle. Moreover, it gives the chance to focus attention on issues of structure and design.

Relieved from the problem of focusing on low-level details students can be assigned more demanding assignments that stress structure and design issues. Especially in the case of iconic programming languages that are used for developing programs with the form - usually - of a flowchart, attention is focused on developing problem-solving skills. Besides the technology of syntax-directed editing, programming environments based on an iconic notation utilize the technology of software visualization as well, with the aim of supporting the comprehension of algorithmic and programming concepts.

In the case of programming environments with an iconic notation, software visualization refers to the use of a graphical representation of programs, the visualization of data structures and variables. Moreover, program animation is used in order to provide a detailed, highly visual view of the source code of a program in execution, giving students the opportunity to study programs and program constructs, functions, procedures, recursion, variables, parameter passing mechanisms, data structures and flow of control. Program visualization can be either static (i.e. presentation of a linked list with a diagram) or dynamic.

The simplest example of a dynamic visualization of source code is highlighting the line of the code that is executed by changing the color of its background and showing simultaneously its result. It is obvious that in programming environments based on an iconic notation there is no source code, although their creators refer to the flowcharts developed as programs. No matter how we call it, a flowchart consists of distinct steps that can be highlighted and executed one at a time with a simultaneous update of variables.

Program animation and software visualization in general, draw students' attention and offer many opportunities to novice programmers, such as:

1- studying programs and complex program concepts through the use of pictures.
2- comprehending theoretical concepts through practice, i.e. experimenting with different data sets and investigating various aspects of a problem, concept or algorithm with the desired pace.
3- easier debugging of programs.

◆ Iconic Based Programming Environment

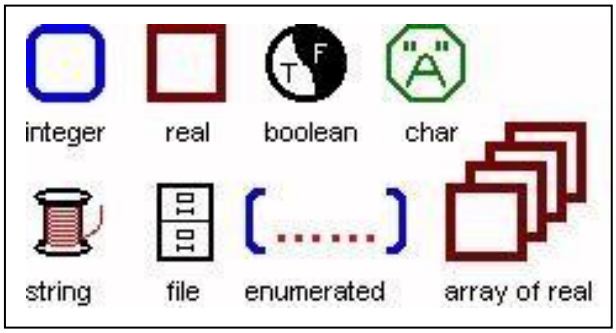
In this section, flowchart-based programming environments that were developed for academic and not commercial purposes are reviewed.

❖ BACCII & BACCII++ (Ben A. Calloni Coding for Iconic Interface):

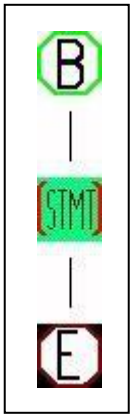
BACCII & BACCII++ were developed at Texas Tech University with the aim of learning procedural and object-oriented programming concepts. Research has been done by the co-authors contributed to developing window based iconic design environment, which is BACCII. BACCII gives the user the ability to design algorithms of main programming concepts such as loops and conditional statements by using graphical iconic representation where the environment is syntax-directed. After finishing designing an algorithm using graphical iconic representation the user can easily convert that into a code of any text-based languages. Later-on BACCII included object-oriented properties, and this is where BACCII++ has been created.

- **Icons:**
  the following Figure showing Icons developed for BACCII++ environment. these icons are used for typing programming statements and designing algorithms. The icons have representative symbols explaining clearly the idea and the meaning of a statement. These icons symbols are not the common symbols used in flowcharts.

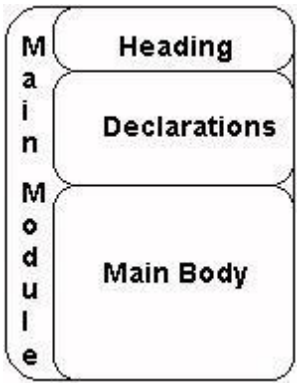
- **Data Types:**
  As mention BACCII & BACCII++ Icons' design clarifying the ideas behind the icons and its functionality of each icon clearly which help the users to quickly understand the icons and how/where to use it.
  for example, the integer icon contains rounded (smooth) corners which indicate the idea that the integer number is a round or a whole number. where the real number icon has sharp corners to indicates the accuracy.



A very important concept which is a block of code concept (starting a block of code in a specific position by using opening bracket, and ending it in another position by using closing brackets) is shown in Figure 2. in BACCII++ this concept is applied by using two nodes (it means opening and closing brackets) begin-node and end-node. Statements between these begin-node and end-node are organized sequentially.

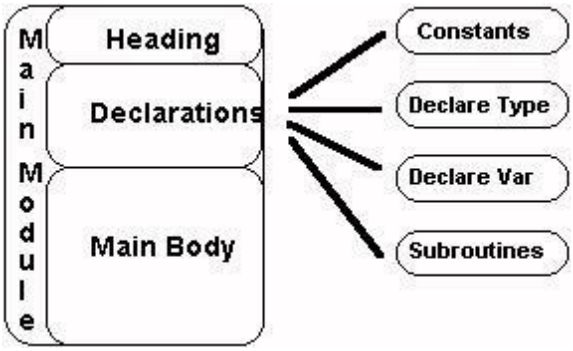


- **Main Screen Display:**
  Activating the BACCII++ for the first time will present to the user a standard Windows layout. The grayed menus give the user the ability to select the FILE option. when a FILE has selected a drop-down menu showing up giving the user the choice to either create a new file or open an existing one. After finishing this process of selection, the user later-on will be informed with more information.



  Now there are 3 choices available to the user:
  - Heading
  - Declarations
  - Main Body.

  The name of the program MUST be declared in BACCII++ before declaring anything else. Also, before creating the Main Body at least one declaration has to be done. at the same the when the user enters the info to "Heading" section, the "Declarations" section shows a graphic of the available choices.



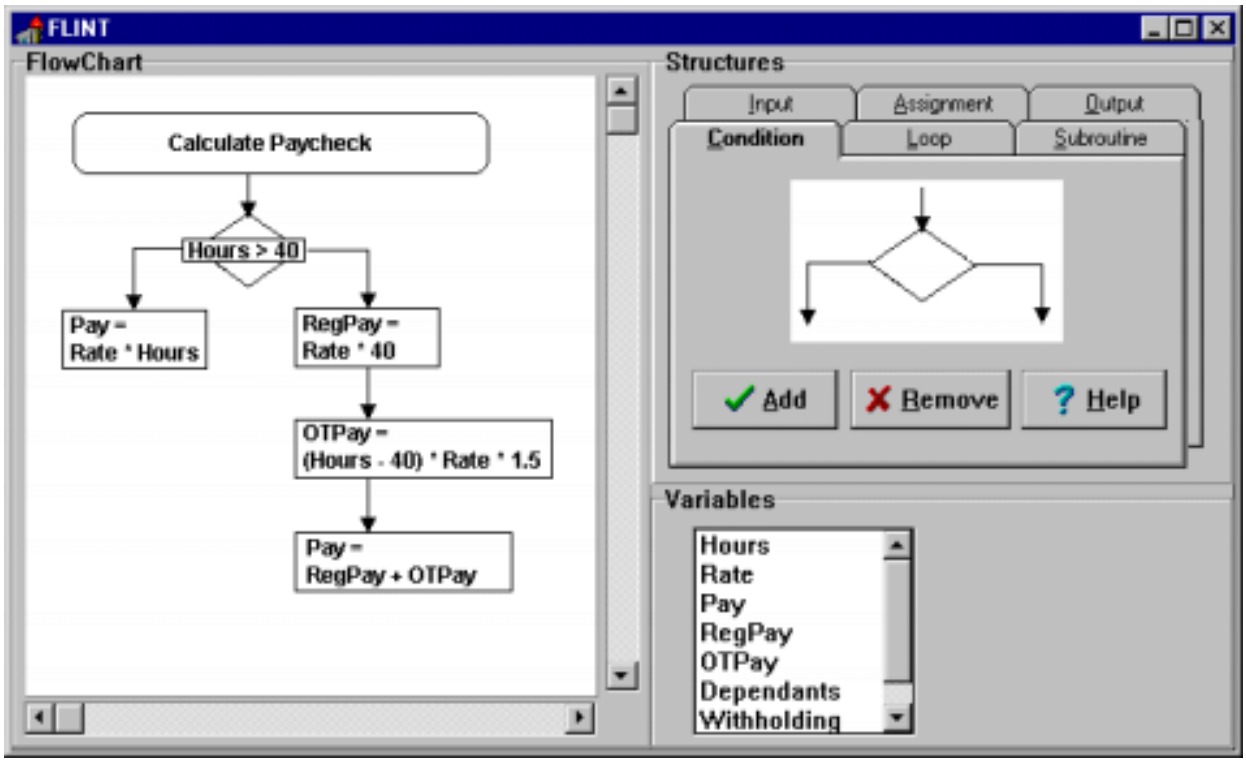
- **Automatic Source Code Generation:**
  After a program is completed the system will automatically generate corresponding text-based correct source code in Pascal, C, C++, Fortran, and Basic.

❖ FLINT (Flowchart Interpreter System):

FLINT was developed at Western Kentucky University. As we mention before learning the concepts of algorithms designing and gaining the skills of problem-solving is not a quite easy task for those begin just beginning studying programming. this is what FLINT (and every other iconic-based programming environment) aiming to do. discovering and solving the problems and difficulties of understanding the programming concepts and make it much easier to understanding programming syntax.

FLINT is an iconic-based programming environment facilitates constructing and designing algorithms to solve problems by drawing charts that logically flows from the top to the bottom implementing the steps of solving a problem.

(Figure) The FLINT interface:



- **Programming in FLINT is presented as activities, such as:**
  - Design (using structure charts).
  - Implementation (using flowcharts).
  - Testing and debugging (using the interpreter).

  Providing iconic interface allows the students to improve their programming and problems-solving skills without focusing on syntactic details of a programming language, in addition to provide a big picture of the full software life cycle.

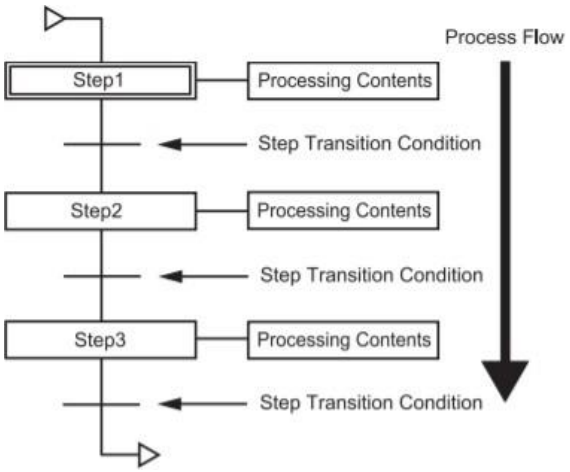
- **Program development and execution:**
  In FLINT the users should first build a structured diagram to determine the key functionality issues.in that structured flowchart (SFC) the user defines the node before defining the process. This makes the users to exposed more to functional decomposition before exposed to sequence structure fundamentals likewise the iterations and selection structures. Also, the in FLINT while developing and structuring the flowcharts, different colors are applied to its symbols which makes it much easier for the user to differentiate between flowchart's symbols and its functionalities.

- **FLINT system features:**
  input, output, if-else conditions, variables, while-loop, and methods.

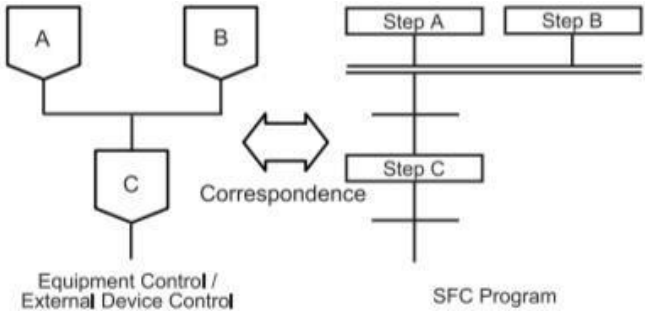
- **Program execution:**
  It supports programs visual execution by highlighting the variables During the program execution along with each diagrammatic component.

❖ SFC (Sequential Function Chart):

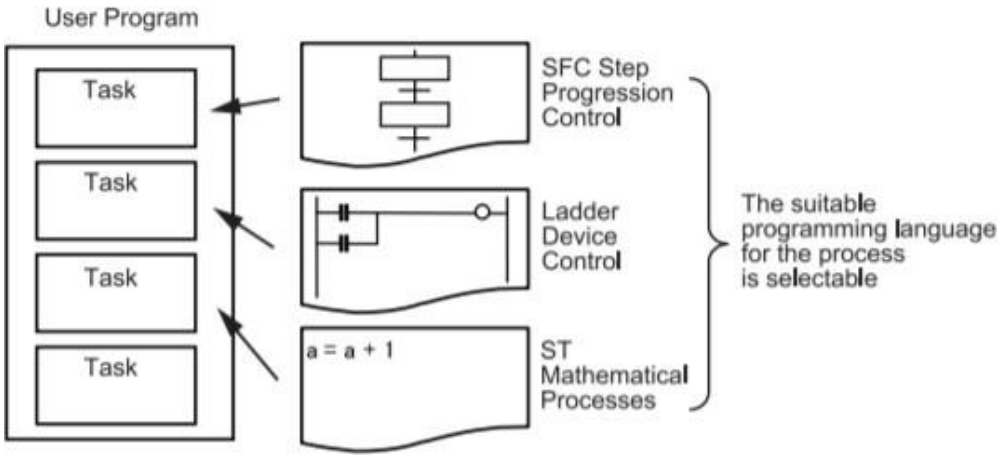
- **Describing Programs with SFC:**
  SFC is iconic-based (graphical) programming language that use the shapes and icons to construct diagrams shows the flow of a program's processes. That allows the user to understand and control the processes sequence by clarifying the condition of transaction for each step, that how SFC helps the user to understand the order of processes and transition status.



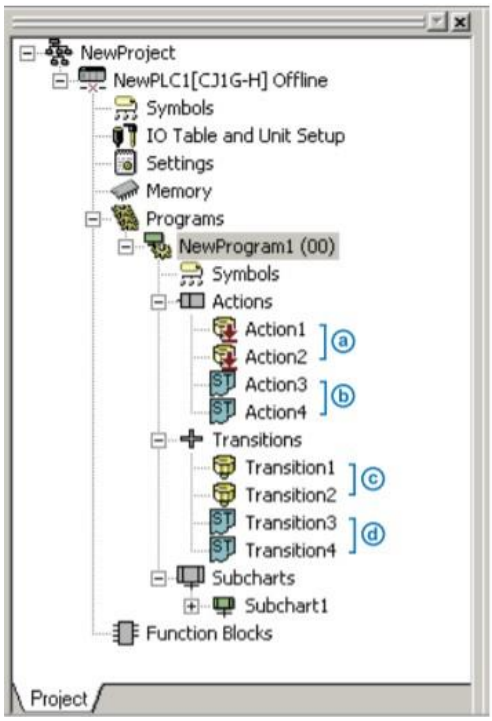
- **Correspondence of Steps and Programs:**
  The actual step flow is corresponded to the program control components. this makes it easier to when it came to maintaining and debugging a program.



- **Choosing a Programming Language for Your Application:**
  in SFC a program can be coded in combination of SFC, ladder and Structured Text or (ST). The user can select the appropriate programming language for each process. for instance, using SFC to code step progression, ladder for device control, and structured-text with arithmetic processes.

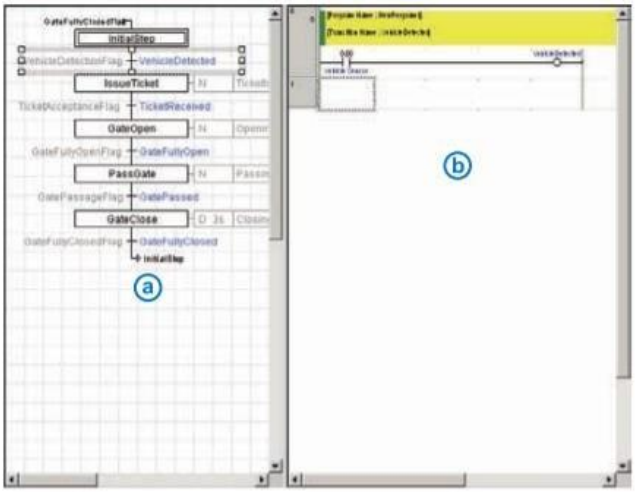


SFC can be used for code flow and order of controls, where ladder can be used to code actions within steps, also transitions between steps. and Structured-Text can be used for Boolean variables.

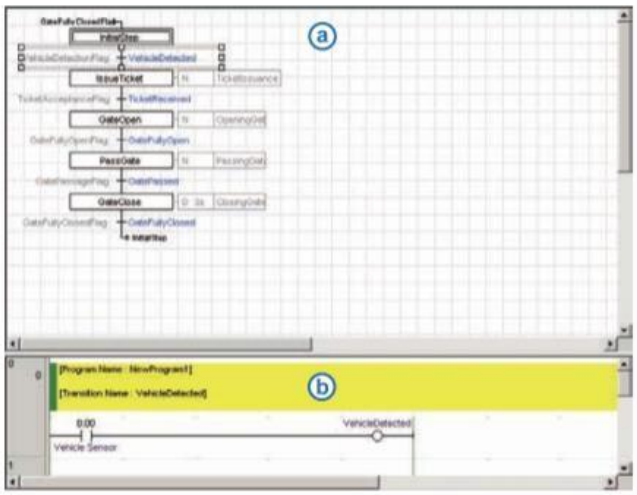


ⓐ : Action Program (Ladder)
ⓑ : Action Program (ST)
ⓒ : Transition Program (Ladder)
ⓓ : Transition Program (ST)

SFC editor gives the user the ability to display and edit both SFC chart & programs transition/actions at the same time. This gives the advantage to the user to see and edit the process and transition conditions during keeping the perspective on the program as a whole. SFC editor can show the Program view & the SFC view in both alignment vertical/horizontal.

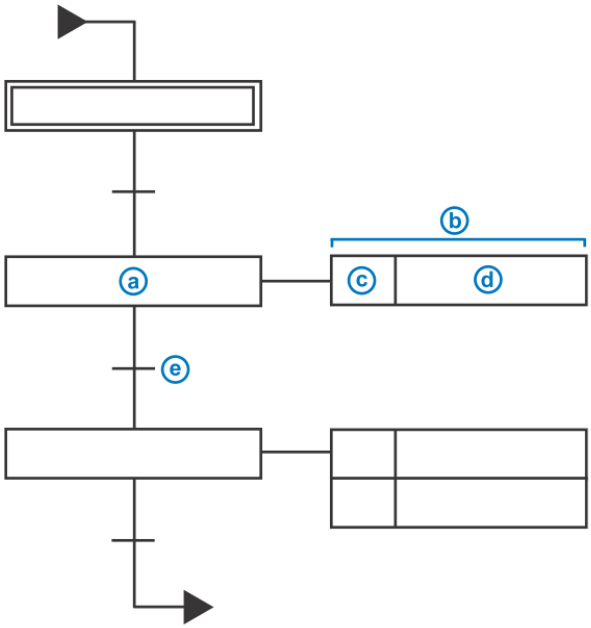


ⓐ : SFC View
ⓑ : Program View



- **Elements of SFC:**

ⓐ: Step
A step is an element of SFC programs, and represents a single process within the overall process flow. When a step becomes active, action blocks assigned to that step are executed.
The first step of a program is referred to as the "initial step".



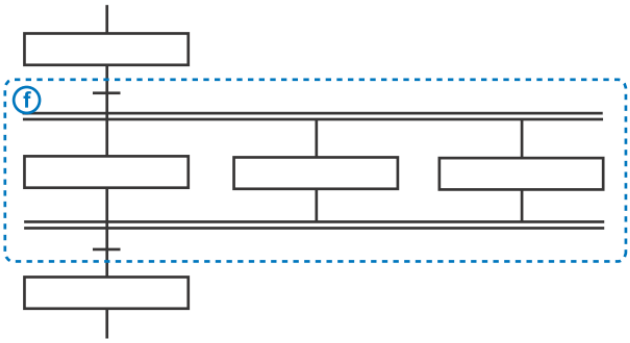
ⓑ: Action Block
An action block contains the step processes (actions) for a single step.

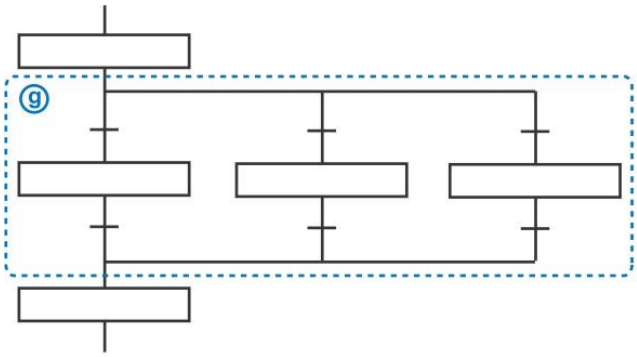
ⓒ: Action Qualifier
An action qualifier defines the execution timing and status retention preference for each action.

ⓓ: Action Name
For each action, specify a Boolean variable (contact) or an action program name.

ⓔ: Transition
A transition represents the condition that transfers the active status from the step before the transition to the step after the transition.

ⓕ: Simultaneous Sequence Divergence/ Convergence
A simultaneous sequence divergence is a structure in which a single transition is followed by multiple steps. When the transition condition is satisfied, all of the connected steps become active simultaneously.
A simultaneous sequence convergence is a structure in which multiple steps are followed by a single transition. Active status is transferred when the transition condition is satisfied and after all the steps have been activated.



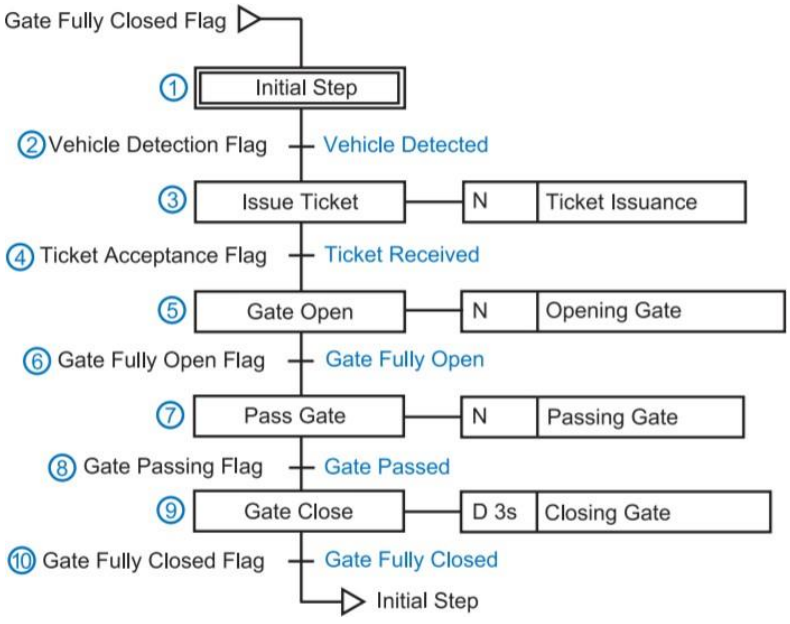


ⓖ: Divergence/Convergence
A divergence is a structure in which a single step is followed by multiple transitions. The active status is transferred to the transition for which the condition is satisfied.
A convergence is a structure in which multiple transitions are each followed by a step. This structure merges a flow that has been branched.

- **Programming Example:**
  A model program for the depicted stopping entryway framework is appeared as follows. The numbering in the chart compares to the numbering in the 2-2-2 Action Flow Diagram.

  Note: the projects utilized in this manual are accommodated instructive purposes just, to help in seeing how CX-Programmer functions. When structuring a program for real use, make sure to make contemplations for security as far as equipment gadgets and control techniques

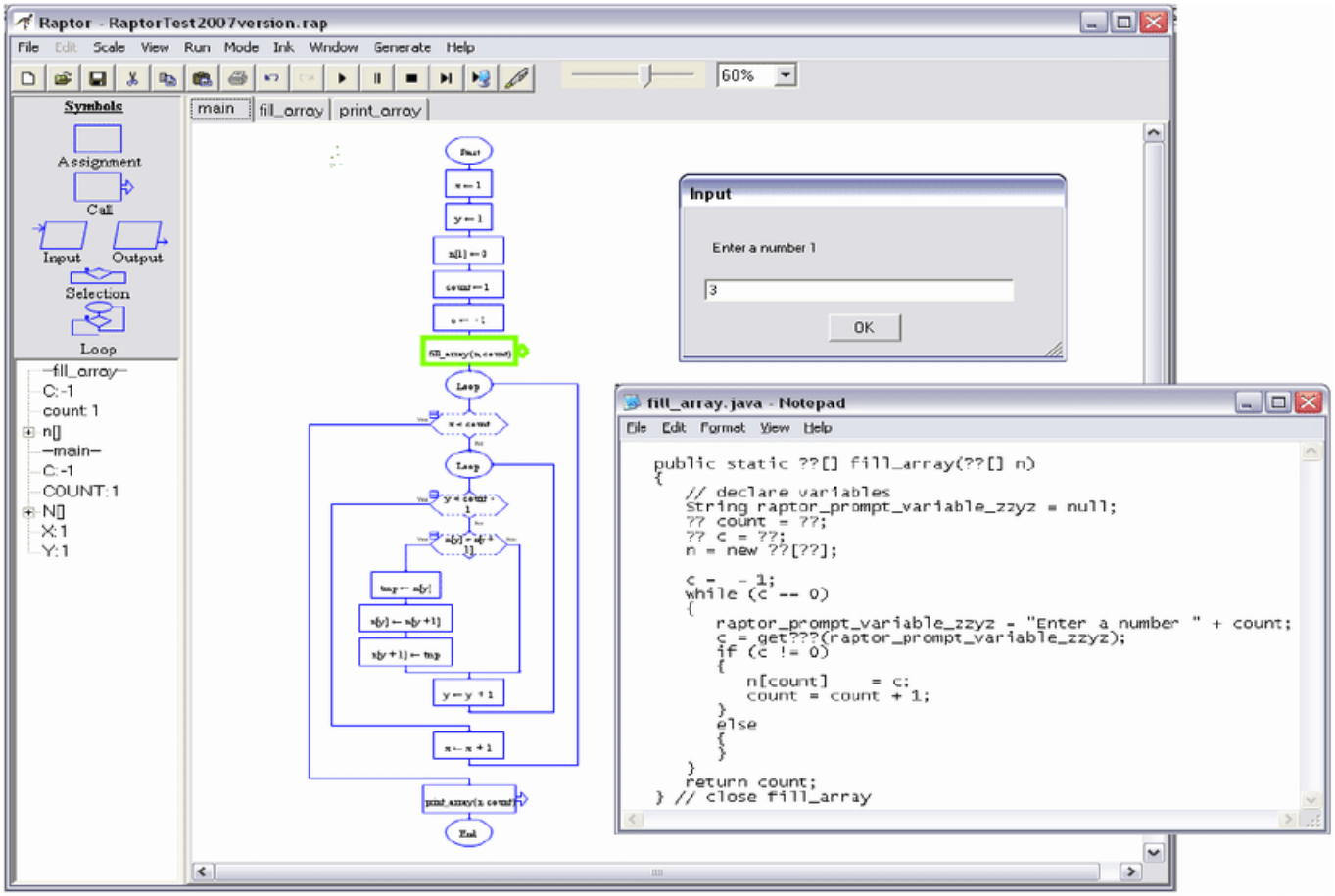
  C Chart: Make an SFC graph dependent on the framework activity. Utilize a Step to code steps, an Action Program for procedures inside a
  stage, an Action Qualifier for execution timing, and a Transition for progress conditions.



❖ RAPTOR:

RAPTOR is an iconic-based (graphical) programming environment that was built to improve the Academy of the USA Air Force's computing courses. RAPTOR includes imperative and OOP properties. RAPTOR aims to learning process by visualize the algorithms that student would build. RAPTOR focuses mainly on improving problem-solving skills along with getting rid of syntactic programming problems.

RAPTOR Generating a Code (Figure):



- **Program Development:**
  in RAPTOR instead of constructing the program using individual icons and arcs, the programs are built (auto-structured) through drag and drop complete structures into the flowchart. one of the main advantages of RAPTOR is that it redrawing itself without user intervention, which helps the users to improve themselves and be only attention to programming and how to solve the problems in front of them instead of focusing how to build a flowchart.

- **RAPTOR System Features:**
  input, output, comments, assignment, conditions/selection, loops, methods/functions, pre-test, mid-west, post-test, one-dimensional & two-dimensional array, files, strings, and a graphics library for enhanced user interaction. in addition to that, RAPTOR environment has several pre-defined functions to perform some mathematic operations such as generating random numbers and draw some graphs, as well as work with trigonometric computation, and some more. RAPTOR environment includes the feature of auto-completion that facilitates the users to call functions. Another interesting feature that is supported in RAPTOR environment is that the errors are kept reported all the time during the process of developing and the system provide some feedback about these errors helping the user to get rid of them by suggesting some solution to fixed those errors.

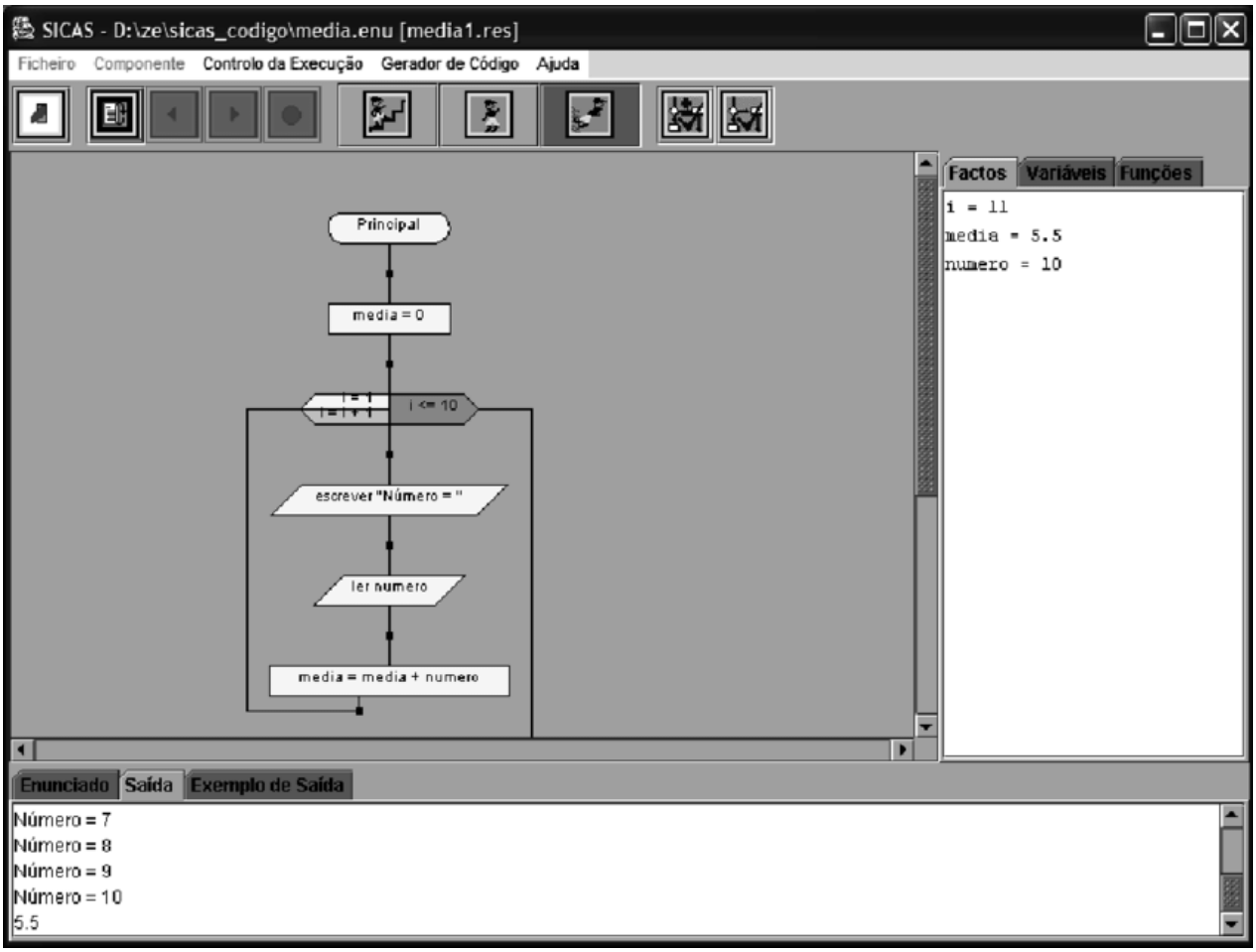
- **Program Execution:**
  Speed can be determined by the user in the process of tracing through a flowchart. as well as step by step manner is supported in RAPTOR environment as well. a statement that is currently being executed is highlighted, and any change in variables' values is mentioned during the execution.

- **Automatic Source Code Generation:**
  the last versions of RAPTOR supports generating (from a flowcharts) the source code in different programming languages such as Java, C++, and Ada, and it is also create standalone executable programs.

❖ SICAS:

SICAS stands for Interactive System for Algorithm Development and Simulation in Portuguese. SICAS is a tool that aims to improve the enhance the problem-solving skills based on constructivist theories. This tool was designed to strengthen skills such as structuring algorithms and logical thinking and programming skills in general

SICAS interface (Figure)



- **Program Development:**
  SICAS user can construct a flowcharts diagram by choosing icons and symbols those are included in the toolbar and then place them in the required position in the design space. then information can be entered through the dialog boxes. the connection between flowchart's components is done automatically by bunch of lines.

- **SICAS System Features:**
  input, output, conditions/selection, arrays, numeric variables, alphanumeric variables, assignment, repetition, methods/functions, including recursive ones.

- **Program Execution:**
  SICAS supports step by step execution, it also highlighting the statement that currently being executed, and keep updating the variables values. a great feature that SICAS support during the process of execution is that it allows the user to step back.

- **Automatic Source Code Generation:**
  SICAS support generation source code in some programming languages such as Java and C, it also able to convert the developed algorithm to pseudo code.

❖ SICAS-COL

SICAL-COL later-on was undertaking based on SICAS. SICAL-COL used to support people who are learning remotely. in this tool there are three workspaces: singular work, bunch exchange, sharing outcomes.

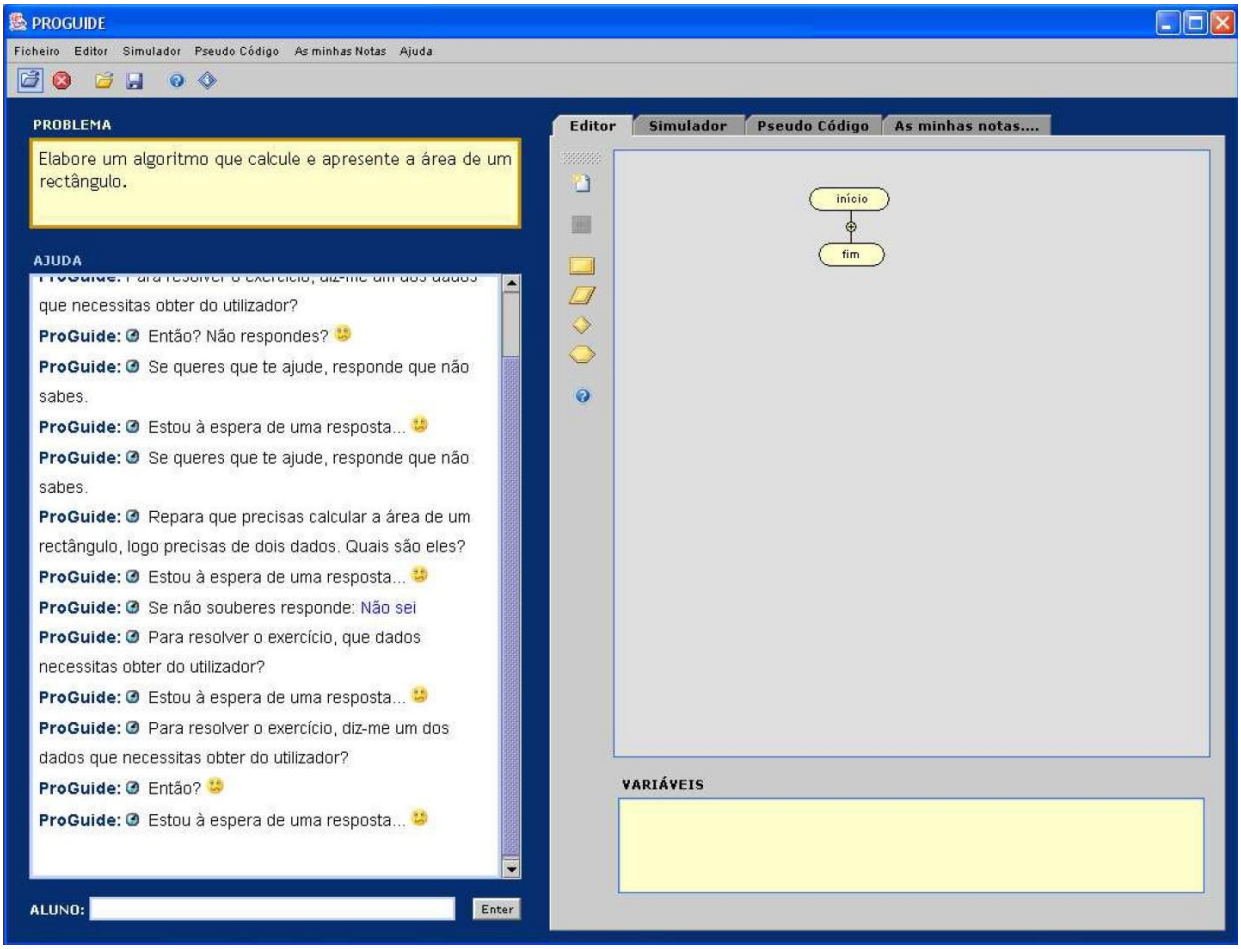
❖ H-SICAS

H-S ICAS is a handheld version of the SICAS environment to be used in cell phones. using SICAS on cell phone and being available on that kind of environment is a big motivation for learners. Cell phones these days are essential and supporting SICAS this feature is an important step because, it would become will reach to the user much easier. Of course H-SICAS provide similar advantages with its work area, but that requires a big modification on the system interface to make possible to be used on cell phone.

❖ ProGuide:

ProGuide system was developed at the University of Coimbra in Portugal. It produced by the same team who is worked on SICAS family environment. ProGuide aimed to enhance the problem-solving skills for novices' students in programming, as well as guiding the student during the process of solving a programming problem, and helping the student to reach to an initial design even if it was a flawed design, because that can be improved while interaction with the environment.

ProGuide Interface (Figure):



- **Program development.**
  Once the user opens the ProGuide system he/she will find an interface that consist of three main parts:

  1- Problem Box (on the top left corner):
  This is where the problem (that the user working on) description is written.

  2- Editor or Simulator (on the right side):
  This is where an algorithm to solve a problem is designed or simulated using a flowchart diagram.

  3- Chat Section (on the bottom left corner):
  This is where an instructor can communicate and guide the student.

- **ProGuide System Features:**
  An algorithm in ProGuide can be presented just like how it was in SICAS, but only some essential features are supported by the ProGuide, which are:

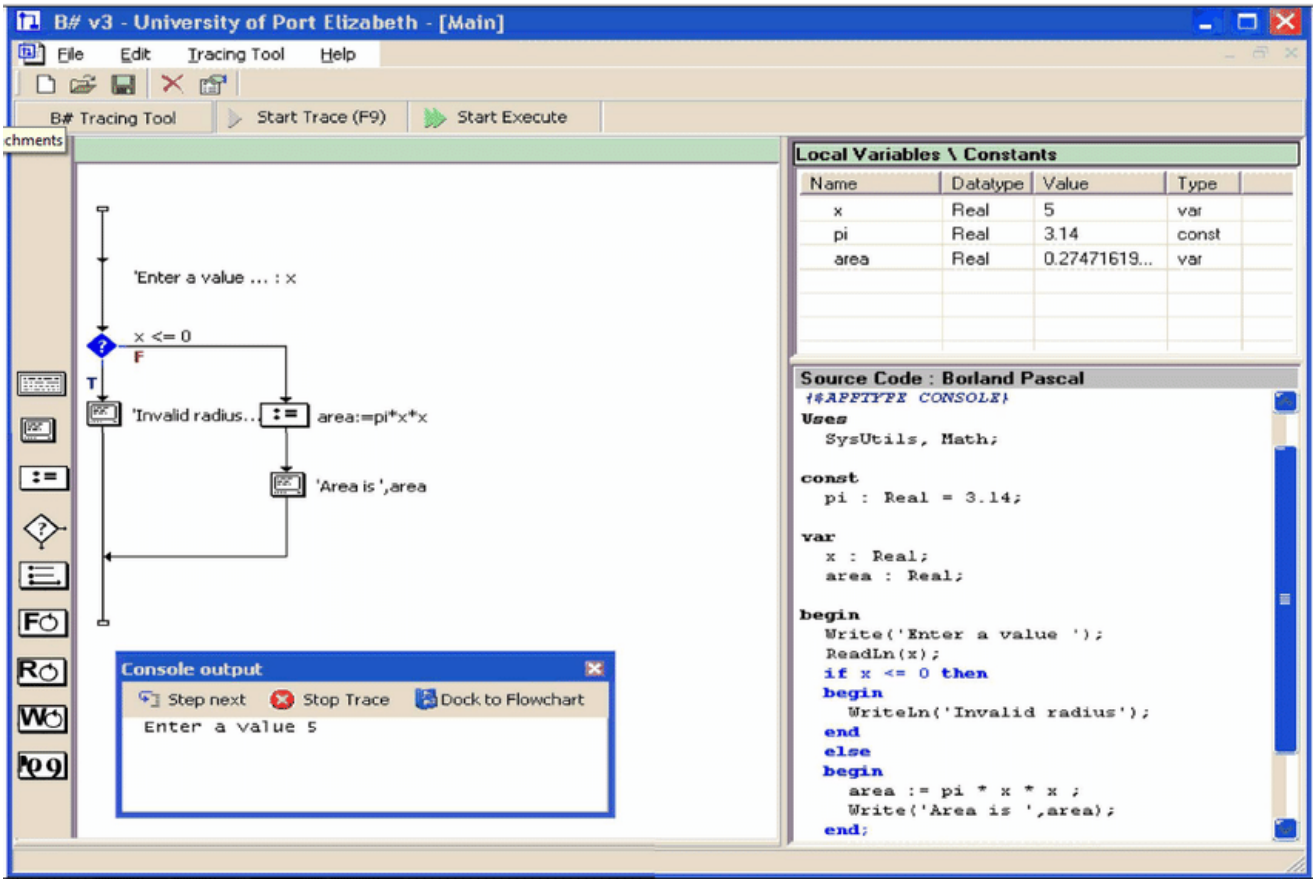
  input, output, conditions/selections, assignment, repetition structures otherwise ProGuide is similar to the rest of the iconic-based systems that we have discussed yet.

  ProGuide supporting animation on flowchart diagram that used to build a program. It also supporting tutoring system, which is communicating with a student through the Chat Section we mentioned before to inform the student with the selected exercise. this communication is not compulsory the student can turn it off if he/she want so. if the student chooses to continue with this communication, the system will keep asking the student some questions, and presenting them with hints or warnings (sometimes).

❖ B#:

B# is an iconic-based programming language along with an integrated programming environment, was developed at Nelson Mandela Metropolitan University. B# aiming to strengthen and enhance student's start with CS1 course.

B# environment interface:



- **Program Development:**
  in B# environment a flowchart diagram can is designed by using some icons similar to those ones used on BACCII & BACCII++ environment (icons has a special design that clarifying the meaning behind the icon, and make it easier to the user to understand the icon's functionality), which are different common flowchart's symbols those are usually used to design a flowchart.

  in order to construct a flowchart, the user can basically select and icon and then drag and drop it into designing space to add it to the flowchart diagram. the user should determine the related parameters, and the environment later-on will check that in the syntactic mode during the process of adding new items to the diagram, so if any error is exist the user will be alerted in the future.

- **B# System Features:**
  input, output, variables, conditions/selection, sequence, and iteration. B# also supporting auto-structuring feature to construct a flowchart diagram.

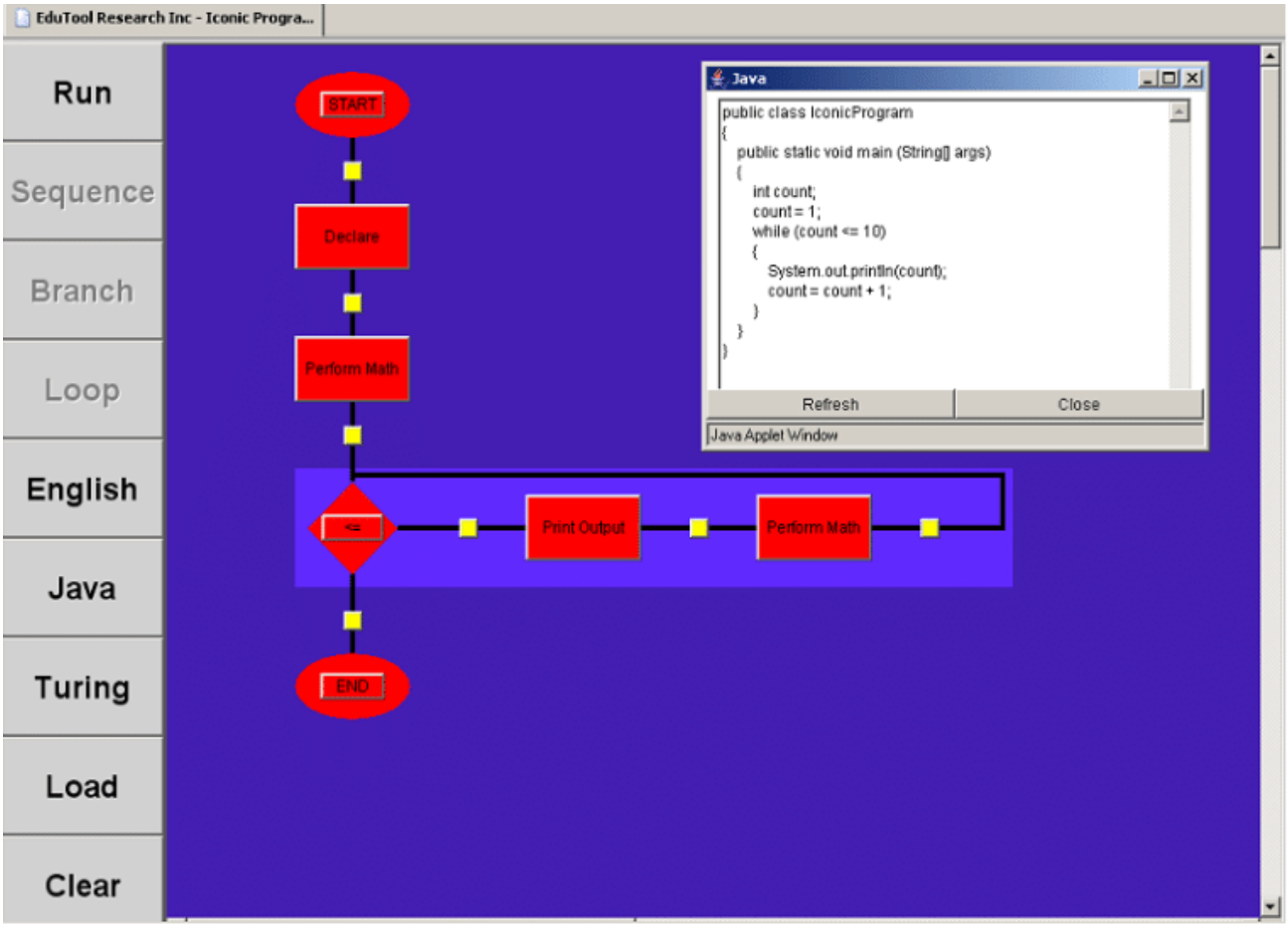
- **Program execution:**
  After the flowchart is constructed be adding it's component by drag and drop them into the designing area, the environment will automatically will generate a source code written in Pascal programming language, after that the flowchart diagram will be structed again and it will get redrawn.

  B# viewing to the user the generated code and the flowchart side by side so that it's easier to understand the code and its algorithm. and this is one of main strength point in B# that is synchronizing the flowchart with a source code that generated by that flowchart. in B# when the user selects a particular component in the flowchart, the corresponding line of code will be highlighted but not vice versa.

❖ IP (Iconic Programmer):

The Iconic Programmer or briefly IP. IP is Microsoft Windows based. it aims to give the opportunity to the users to construct and build and generate executable codes & flowcharts.

IP's interface (Figure)



- **Program Development:**
  The initial interface that the user starting with is a simple flowchart that consist of two icons (START & END) connected to each other by a line. and there is a small square position at the middle of that line. the user can click on this small square (where window going to pop-up asking for selection) to determine statement's type that the user want to insert at that point in the flowchart. Symbols available in IP are icons (not flowchart's common symbols) containing a label and statement's type.

- **IP System Features:**
  IP supports some component such as: input, output, conditions/selections, declaration, assignment, sequence, repetitions.

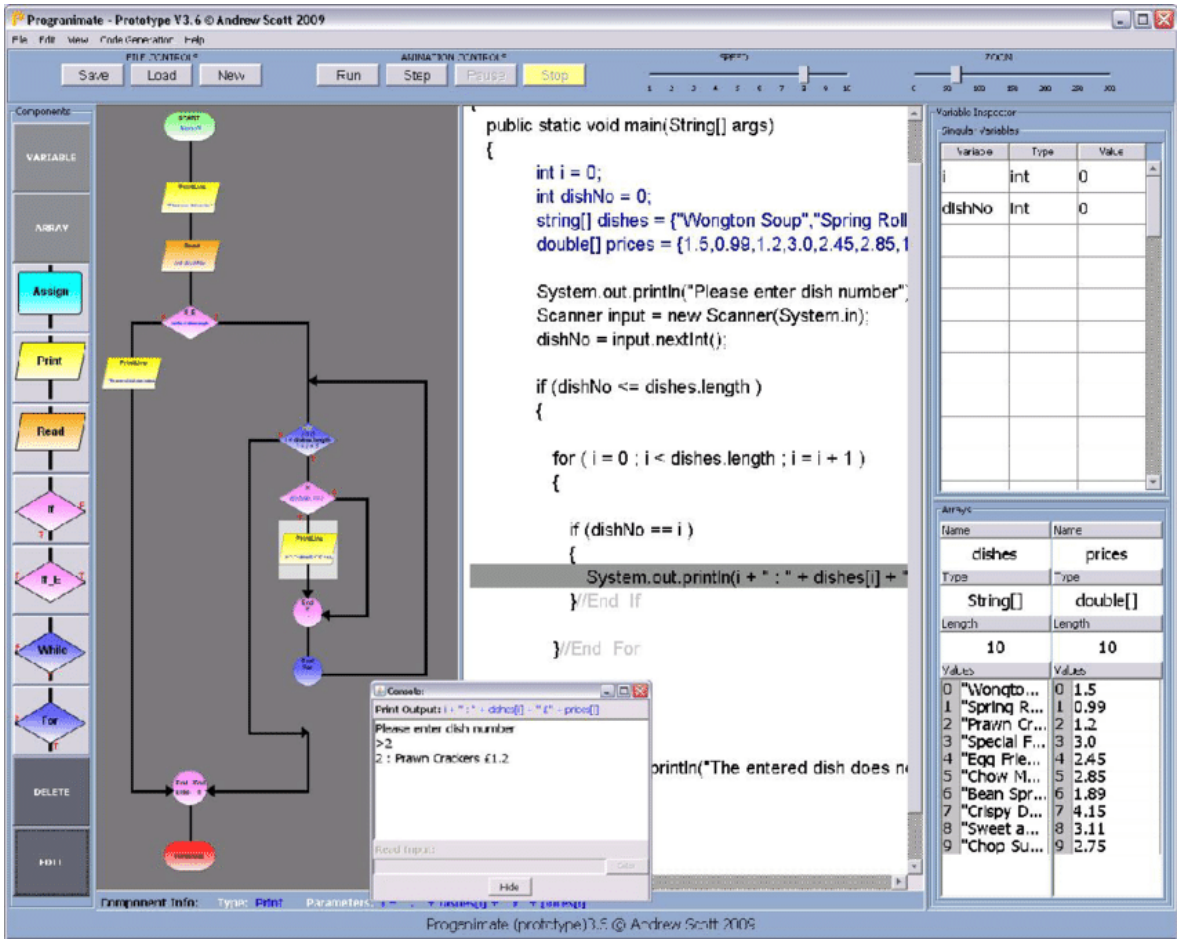
- **Program Execution:**
  IP also supporting the step by step feature during the process of executing the programs, that gives the user the abilities to trace accurately their programs. as usual when a user is executing his/her program the flowchart's component currently being executed is highlighted, and the variable are kept updated if its values is changed, also and explanation written in natural language is viewed to the user in the run panel, to explain to the user the statement currently being executed.

- **Automatic Generation Code:**
  IP system supports generating a source code automatically for a constructed flowchart in many different programming languages such as Java, C, C++, and Turing. IP also can generate pseudocode for the constructed flowchart. in case of programming languages source code, the user must copy the code and use some other system to be able to run that code has been generated.

❖ Programinate:

Programinate is a web-based environment. Programinate allows the users to design and construct flowcharts those can be executed later-on.

The Programinate programming environment interface (Figure)



Programinate environment interface is divided mainly into three sections:

1- **Flowchart Diagram Section:** the work-space area where the users can design and construct their flowchart's diagrams.

2- **Source Code Generation Section:** where the source code has been generated automatically for a constructed flowchart is presented.

3- **Variable Inspector:** where all information and details about the variable in the program is mentioned.

- **Program Development:**
  In Programinate environment the users can develop their flowcharts diagrams either by dealing with the flowchart itself or interacting with the selected code view. more precisely the user either choose the structure that he/she wants (from the panel placed on the right side of the main window) and then placed it on the position that he/she want to be placed to in the flowchart, or the user can do that by the source code. So, information can be entered by the user when it is necessary (such as if statement condition) through a dialog boxes which will show up when needed.

  In Programinate the symbols used to construct a flowchart is the common flowchart symbols, also in colors are used just to avoid any kind of confusion. for instance, the conditional statements such as "if" and "while" are presented in different colors.

- **Programinate Features:**
  input, output, variables, assignment, conditions/selection, repetition, arrays.

- **Program Execution:**
  Programinate is also supporting the step by step feature during the process of executing the programs with simultaneous data visualization, the user can also trace his/her programs step by step. Programinate special feature is that the flowchart and the source code are synchronized execution.

- **Automatic Generation Code:**
  Programinate supporting generating source code for constructed flowchart in programming languages such as Java, VisualBasic.NET, Visual-Basic 6.0, Pascal or JavaScript. or it can convert the constructed flowchart to a pseudo code.

◆ Comparison:

Comparative analysis of flowchart-based programming environment

Environment	Avail-ability (free)	Programming paradigm/technique		Notation: flowchart with		Execution:	Automatic source code generator	Evaluation	Special feature
		Imperative-procedural	*Object-Oriented*	*common symbols*	*same shaped icons*	*Program animation*			
BACCII/ BACII++		√	√		√		Pascal, C, Fortran, Basic, C++	√	
FLINT		√		√		√		√	Imposes the waterfall programming model
SFC Editor	√	√		√			Pseudo code in C++ or Pascal	√	
RAPTOR	√	√	√	√		√	Ada, C#, C++, Java	√	Notation: UML class diagram Source code generation: allows development of generators for other languages
SICAS		√		√		√	Pseudo code, C, Java	√	Execution: Stepping backward
SICAS-COL		√		√		√	≠		Supports collaborative activities
H-SICAS		√		√		√	≠		Adaptation of SICAS for usage on mobile devices
ProGuide		√		√		√			Uses a tutoring system model
B#		√			√	√	Borland Pascal	√	Both the flowchart and the generated program can be executed
Iconic Programmer	√	√			using text	√	Pseudo code, Java, Turing, C/C++		Execution: Explanatory visualization
Progranimate	√	√		√		√	Java-like pseudo code, Java, VisualBasic.NET, VisualBasic 6.0, Pascal, JavaScript	√	Deployment: easily integrated into a web page Execution: Synchronized execution of flowchart & source code

Comparative analysis of flowchart-based programming environment (continued)

Features	*BACCII*	*FLINT*	*EC*	*FCI*	*Raptor*	*SFC*	*SICAS*	*VL*	*IP*	*DF*	*A&Y*	*B#*	*PG*	*G&G*	*CVF*	*Web-flowchart*	*Programinate*
Year published	*1992*	*1999*	*2002*	*2003*	*2004*	*2004*	*2004*	*2004*	*2005*	*2006*	*2006*	*2006*	*2007*	*2007*	*2009*	*2010*	*2010*
Flowchart	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flowchart-based programming	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓			✓	✓
Flowchart generates code	✓				✓	✓	✓	✓	✓	✓		✓					✓
Structural rules enforced	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓		✓		✓
Colour used to fully differentiate components and structures	✓	✓	✓							✓							
Code	✓				✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓
Code-based programming															✓		
Code generates flowchart															✓		
Code and flowchart displayed concurrently						✓					✓	✓		✓	✓		✓
Synchronised highlighting of flowcharts and code											1/2	1/2		✓	✓		✓

Comparative analysis of flowchart-based programming environment (continued)

Features	*BACCII*	*FLINT*	*EC*	*FCI*	*Raptor*	*SFC*	*SICAS*	*VL*	*IP*	*DF*	*A&Y*	*B#*	*PG*	*G&G*	*CVF*	*Web-flowchart*	*Programinate*
Year published	*1992*	*1999*	*2002*	*2003*	*2004*	*2004*	*2004*	*2004*	*2005*	*2006*	*2006*	*2006*	*2007*	*2007*	*2009*	*2010*	*2010*
Synchronised visual execution of flowcharts and code											✓			✓			✓
Non visual execution				✓				✓									
Visual execution		✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓			✓
Variable inspector	✓	✓	✓	✓	✓		✓		✓		✓	✓					✓
Error feedback	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓			✓
Java support					✓		✓	✓	✓						✓		✓
Variables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sequence	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Selection	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Iteration	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Arrays	✓				✓	✓	✓	✓							✓	✓	✓
Procedures	✓	✓			✓		✓	✓							✓	✓	
Empirically evaluated	✓	✓			✓			✓				✓					✓
Associated pedagogy		✓											✓	✓			✓
OS dependency	Win	Win	Win	Win	Win	Win	Win	Win	Win	Win	Any	Win	Win	Win/ mac	Win	Win	Independent

❖ Availability:

Different set of iconic-based programming environments have been created in the early 1990 such as BACCII. This sort of environments got a lot of attention, some of them even keep developing new version for their environments yet. For instance, Emanuel de Jesus in 2011 present his interest in developing a new web-based application for students to write programs in a structured language and then the system will automatically generate for them the flowchart and vice versa. This environment will take advantage of the advances offered by HTML5 and JavaScript, so as to run on any modern browser independently of the operating system and hardware and be easily integrated into a Learning Management System.

❖ Notation:

In all the environments programs are developed with the form of a flowchart. However, in some cases, the symbols used are not the common symbols used in structured flowcharts. Specifically, symbols that have the same shape and a characteristic icon are used. In this case, the creators of the corresponding environments (i.e. BACCII/BACCII++, B#) refer to the notation used with the term iconic programming language. Iconic programming languages, usually, have an extended instruction set, supporting the declaration of variables, manipulation of text files and generally constructs that are not commonly depicted in flowcharts. Some environments use a hybrid approach. For example, in Iconic Programmer same shaped icons are used for input, output and expressions that have just a label with the type of the statement and not the specific statement (i.e. "Declare" without the name of the variable being declared). This visualization of the flowchart, however, is not considered as helpful as the one used in Progranimate where common flowchart symbols are used having as a label both the type of the statement and the statement itself. Moreover, in Progranimate, color is used for distinguishing between same shaped symbols for different structures, such as the ones used for "if" and "while". The flowcharts developed in Iconic Programmer make more sense during execution that utilizes explanatory visualization for presenting users with the information not visible in the flowchart itself. In one of the environments, namely RAPTOR, UML class diagrams are also supported when the user selects the object-oriented mode.

❖ Programming Paradigm/technique:

Flowcharts were traditionally used as a medium of expressing structured algorithms representing imperative-procedural programs. Since flowcharts are used for introducing novices to algorithms and programming, the vast majority of flowchart-based programming environments that were developed support the imperative-procedural programming paradigm. As Scott Watkins and McPhee state the learning curve of an objects first approach adds difficulties in the imperative approach and consequently it is much steeper. However, two of the environments analyzed support both the procedural and the object-oriented programming paradigm. These environments are BACCII++ and RAPTOR. In BACCI++ it is not clear how object-orientation is supported, since the environment is not freely available. In RAPTOR users can design a UML class diagram using direct manipulation techniques and interaction. Inheritance and polymorphism, as well as class nesting, association, composition, aggregation and dependency can be depicted in the user's UML class diagram. In this object-oriented mode of RAPTOR, structured flowcharts are used for implementing the body of each method declared in a class, helping novices realize more easily and clearly how the imperative programming paradigm is used in the context of the object-oriented programming paradigm.

❖ Execution:

All the tools, with the exception of BACII/BACII++ and SFC Editor, offer the ability of executing flowcharts, or to be more precise the ability of program animation with automatic update of variables. Especially in the case of SICAS even backward step by step execution is possible, according to its developers. Moreover, in the case of B# users can execute the Borland Pascal source code automatically generated from the flowchart, while in ProgrAnimate a synchronized step by step execution of both the flowchart and the code automatically generated in the selected language is offered. In Iconic Programmer explanatory visualization is used for presenting explanatory messages in natural language. This feature is partly supported in Progranimate as well by presenting information about the component of the flowchart being executed, such as the type of a control structure and the condition, at the bottom of the main window. All the aforementioned features are extremely important for novice programmers, since they provide great support in comprehending the semantics of programming structures and flow of control, as well as locating and correcting logic errors.

◆ Conclusion

we are considering that the flowcharts a visual helper for novice students, therefore the main approach here is the flowchart. all result made by some experiments are giving us the result like that these kinds of environment are improving the student's abilities in understanding and dealing with algorithms and improving the problem-solving skills, some of these results are listed below:

- the process of generating the source code for construct flowchart improve novice programmers to problem-solving skills because it helping theme to focus on how to solve the problem not how to write a code.

- sometime the syntactic programming language contain some ambiguous which make it difficult for novice student to understand the algorithms behind the code, but flowcharts are universal representation that presenting an algorithm in such clear way which improving the student's skills in this area.

- Novice programmers who use a flowchart-based learning tool in an introductory programming course may show significant improvements in logic and code writing skills.

- Evaluating some of the reviewed systems shows that initiating the process of devising a solution to a given problem contributes to the improvement of problem-solving skills of novice programmers.

References

- **Teaching Programming Concepts Using an Icon-Based Software Design Tool**
  By Donald J. Bagert, Senior Member, IEEE and Ben A. Calloni.

- **The Flowchart Interpreter for Introductory Programming Courses**
  Department of Computer Science Western Kentucky University Bowling Green, KY 42101
  By Thad Crews, Uta Ziegler

- **Using Flowchart-based Programming Environments for Simplifying Programming and Software Engineering Processes**
  Department of Technology Management University of MacedoniaNaoussa, Greece
  By Stelios Xinogalos,

- **Flowchart-based programming environments for improving comprehension and problem-solving skill of novice programmers.**
  Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya
  By Danial Hooshyar, Rodina Binti Ahmad, Mohd Hairul Nizam Md Nasir and Shahaboddin Shamshirband

- **SFC Introduction Guide**
  <
  https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjT6P3P9dLhAhU2BWMBHbw9C9kQFjAAegQIARAC&url=https%3A%2F%2Fwww.fa.omron.com.cn%2Fdata_pdf%2Fmnu%2Fr149-e103_sfc.pdf&usg=AOvVaw2freZbTJ335o65WH5JJ5mB
  >