



THE PEER-LED TEAM LEARNING INTERNATIONAL SOCIETY
PROCEEDINGS OF THE INAUGURAL CONFERENCE
MAY 17-19, 2012
NEW YORK CITY COLLEGE OF TECHNOLOGY OF
THE CITY UNIVERSITY OF NEW YORK
BROOKLYN, NY 11201-2983

Learning Programming Logic Using Executable Flowchart
Ongard Sirisaengtaksin

Abstract

The main objective of the project is to develop a programmable and executable flowchart application. The application will allow students to create a program with very minimal syntax and independent from any programming language. Students will be able to create a program using block symbols like symbols that are used in a flowchart. Students can create a program just by adding a block symbol and add a programming code/statement into the block symbol one by one until the program is complete. Then, students have an option to either run or step through the program. This allows students to visualize the flow of execution of the program and focus on the logic of the program rather than the syntax of the language. This application can also be used to demonstrate concepts such as functions, if, and loop constructs.

Introduction

In general, students find a programming course difficult, especially non-computer science majors. This is true no matter what programming language is being taught and used in the course, whether it is C++, Visual Basic, or Java. The main reason is that students must have both good problem solving skills or logic, and a command of the programming language syntax to be able to write a complete program from start to end. This implies that in order to create a program, one must be able to come up with an algorithm for the solution to the problem and then convert the algorithm into code according to the programming language used. Most students lack problem solving skills or logic. Some students have difficulty understanding programming constructs and logic. Some even have a hard time comprehending the syntax of the programming language. One possible solution to alleviate these learning problems is to engage students in a programming environment that requires only logic skills not syntax. Furthermore, students can also visualize the structure and logic of the program as well as the flow of execution of the program.

So, our main objective is to develop a programmable and executable flowchart application that allows students to create a program with very minimal syntax and independent from any programming language. Students should be able to create a program using block symbols like symbols that are used in a flowchart. Block symbols that are used in the application should be limited to a small set of symbols such as input, output, and condition. The reason is to avoid a confusion which block symbol to be used. So, students can concentrate on the logic of the program. Then, students are able to create a program just by adding a block of symbol and add programming code/statement into the block of symbol one by one until the program is complete. Students also have an option to either to run or step through to the program. This allows students to visualize the flow of execution of the program and focus on the logic of the program rather than the syntax of the language. This application can also be used to demonstrate concepts of programming constructs.

We previously used a commercial application called Visual Logic,[Cengage Learning] in our C++ course. The application provides functionalities that we need to create flowcharts and be able to visualize step by step execution of the program. Even though Visual Logic application requires minimal syntax, some students found it confusing to use since the application uses Visual Basic syntax. That was one of the reasons why we decided and started developing our own application. During the course of development, we learnt of another application that shared the same goals and objectives as our proposed application. The application is called RAPTOR. RAPTOR is a freeware developed by Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, and Steven M. Hadfield, Department of Computer Science, United States Air Force Academy [Carlisle, et al, 2003]. Carlisle also shared the same objectives of implementing this application environment that is to alleviate syntactical knowledge that students have to learn and concentrate on developing algorithms.

RAPTOR - Executable Flowchart

RAPTOR is written in a combination of Ada, C# and C++, and runs in the .NET Framework. RAPTOR begins by opening a blank workspace with a start and end symbol as shown in Figure 1. The user can then add flowchart symbols corresponding to loops, selections, procedure calls, assignments, inputs and outputs by selecting from the palette in the upper left corner and then dragging the symbol and inserting at an appropriate point in the flowchart as shown in Figure 2.

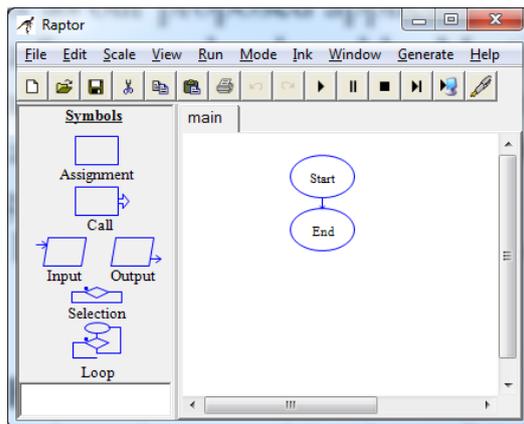


Figure 1: RAPTOR Blank Workspace

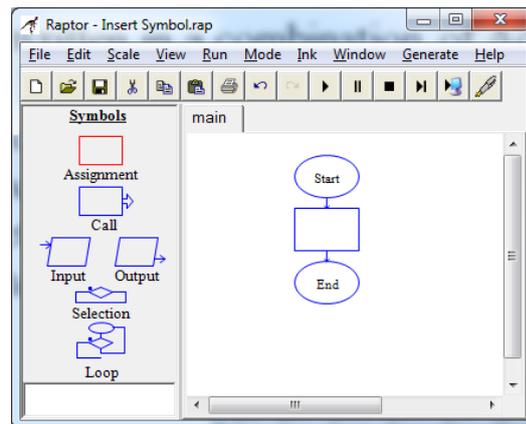


Figure 2: Flowchart with Assignment Symbol Added

The flowcharts are forced to be structured. This will illustrate to students that each step must be executed in sequence. Selections and loops must be properly nested, and each loop has a single exit point. Loops, however, allow the exit condition to be tested at any point inside the loop body. The student may select to use a pre-test, mid-test, or post-test loop simply by adding flowchart symbols before and/or after the loop test.

RAPTOR syntax used within a flowchart symbol is designed to be flexible. Elements have been borrowed from both C and Pascal-style languages. For example, in an expression, either “**” or “^” may be used as an exponentiation operation, and “&&” or “and” may be used as a Boolean “and” operator. RAPTOR enforces syntax checking on each flowchart symbol as it is edited. Therefore, it is impossible to create a syntactically invalid flowchart. If the user enters “x+” as the right hand side of an assignment, they will get an error message and be required to fix the arithmetic expression before leaving the assignment box.

Commenting is done by right-clicking on a flowchart symbol and selecting “comment”. The comment appears as a “talking bubble” next to the flowchart symbol. The comments can be clicked and dragged to improve the aesthetic of the flowchart.

RAPTOR has over 40 built-in functions and procedures which allow the student to generate random numbers, perform trigonometric computations, draw graphics (including circles, boxes, lines, etc.), and interface with pointing devices. As seen in Figure 3, RAPTOR will automatically suggest completions to procedure names.



Figure 3: Entering a procedure call

In addition, RAPTOR will automatically search the current folder for an instructor provided dynamically linked library named “plugins.dll”. If such a file is present, the student will be allowed to call those procedures from within the flowchart, and those procedures will appear in the suggestion list. This allows the instructor to create more interesting assignments by raising the level of abstraction. In Figure 3, “Draw_Board” is from the Tic-Tac-Toe sample plug-in.

During execution, the student can select to single step through the flowchart, or run continuously. The speed of execution is adjustable by moving the slider shown at the top right of tool bar. At each step, the currently executing flowchart symbol is shown in green. Additionally, the state of all of the variables is shown in a watch window at the bottom left corner of the screen.

Example – Guessing Game Exercise

The following is an example of a group exercise for students to go through in our C++ class:

Exercise 6: Guessing Game

Objectives:

- Develop an algorithm for a Guessing Game – guessing a number
- Use chart to illustrate the algorithm

Introduction

In this exercise, you and your partner will take turn to play a Guessing Game. This is how the game plays:

1. You pick any number and write it down on an index card. Of course, your partner will not be able to see the number that you wrote down.
2. Ask your partner to make a guess.
3. If your partner's guess is not correct, your response will be either the guess is too low or too high.
4. Ask your partner if he/she wants to continue.
5. If the respond from your partner in Step 4 is yes, repeat Step 2-4. Otherwise, quit the game.

Step 1: Develop an algorithm by writing down the actual steps. Then create an algorithm chart from those steps. What type of construct do you use in your algorithm, *if-then-else* or *loop* or both?

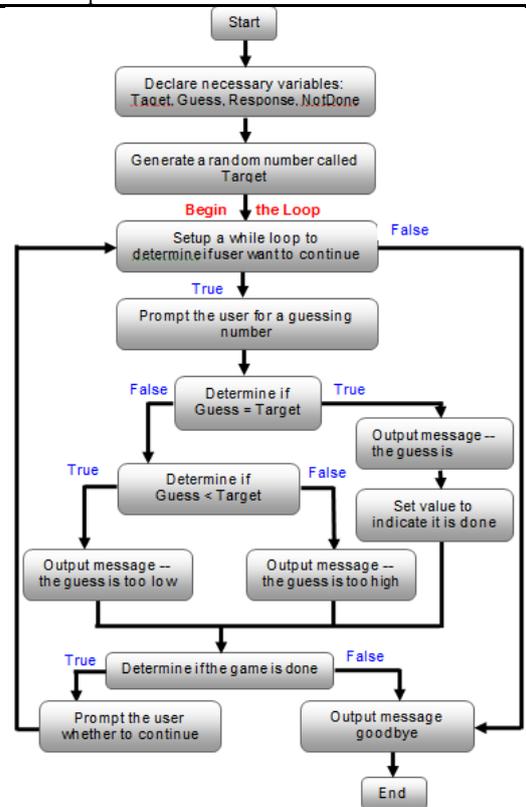
Step 2: Convert the algorithm to C++ code to verify your algorithm is correct.

Step 3: Suppose that you want to make the game more interesting by limiting your partner a certain number of guesses, say 5 guesses. Modify your algorithm, steps and chart so that it will work with this situation.

Step 4: Modify the C++ code in Step3 according to the modification that you made in Step 3 to verify your algorithm is correct.

Students then will come up an algorithm and a simple chart as follows:

1. Generate a random number.
2. Prompt the user for a guess number.
3. Determine if the guessing number is
 - a. Equal to the random number: If it is, output a message that says the guessing is correct and quit the game.
 - b. Less than the random number: If it is,
 - i. Output a message "Too low,"
 - ii. Ask if the user wants to continue the game. If yes, ask for another guess number; otherwise quit the game.
 - c. Greater than the random number: If it is,
 - i. Output a message "Too high,"
 - ii. Ask if the user wants to continue the game. If yes, ask for another guess number; otherwise quit the game.
4. Output a goodbye message.



Then students will be able to create an executable flowchart using RAPTOR, as shown in Figure 4, according to the algorithm and the chart that they come up with. Furthermore, they can test their algorithm for correctness without writing a single code.

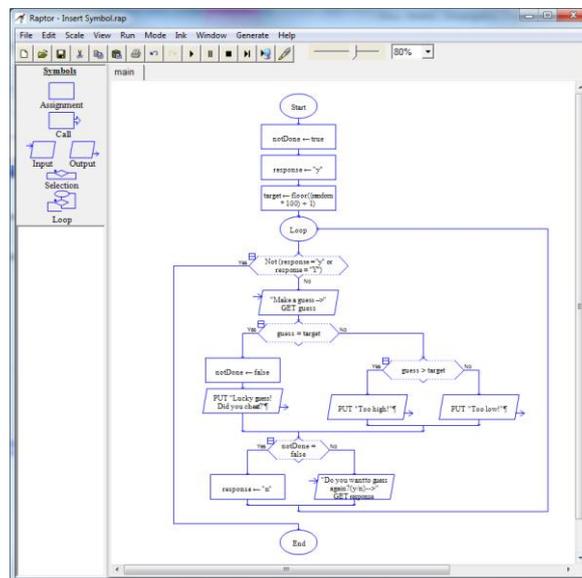


Figure 4: Guessing Game Flowchart in RAPTOR

Conclusion

Executable flowchart such as RAPTOR provides a tool for students to be able to develop an algorithm for a given problem and test it without having to know a programming language. This will provide more time for students to develop their problem solving skills rather. RAPTOR is a flowchart-based programming environment, designed specifically to help students visualize their algorithms and avoid syntactic knowledge. Furthermore, it allows programs to be created visually and executed visually by tracing the execution through the flowchart.

References

- Cengage Learning, Visual Logic. Retrieved from <http://www.course.com/visuallogic>
- Carlisle, M.C., Wilson, T.A., Humphries, J.W., Hadfield, S.M.. *RAPTOR: Introducing Programming to Non-Majors with Flowcharts*. Retrieved from http://raptor.martincarlisle.com/raptor_paper.doc

Cite this paper as: Sirisaengtaksin, O. (2013). Learning programming logic using executable flowchart. *Conference Proceedings of the Peer-Led Team Learning International Society*, May 17-19, 2012, New York City College of Technology of the City University of New York, www.pltlis.org; ISSN 2329-2113.