



Improving and Understandability of Programming using RAPTOR

S.Phani Kumar¹ | K.Narendra Kumar² | P.V.V.S.Eswar Rao³

^{1,3}Assistant Professor, Department of Information Technology, Sasi Institute of Technology and Engineering, Tadepallegudem, Andhra Pradesh, India.

²Associate Professor, Department of CSE, Chalapathi Institute of Engineering and Technology, Guntur, Andhra Pradesh, India.

To Cite this Article

S.Phani Kumar, K.Narendra Kumar and P.V.V.S.Eswar Rao, "Improving and Understandability of Programming using RAPTOR", *International Journal for Modern Trends in Science and Technology*, Vol. 03, Special Issue 01, 2017, pp. 108-113.

ABSTRACT

When students are learning to develop algorithms, they very often spend more time dealing with issues of syntax than solving the problem. Additionally, the textual nature of most programming environments works against the learning style of the majority of students. RAPTOR is a flowchart-based programming environment, designed specifically to help students visualize their algorithms and avoid syntactic baggage. RAPTOR programs are created visually and executed visually by tracing the execution through the flowchart. Required syntax is kept to a minimum. Students preferred using flowcharts to express their algorithms, and were more successful creating algorithms using RAPTOR than using a traditional language or writing flowcharts without RAPTOR.

KEYWORDS: J48, Naïve Bayes, Decision Tree, IBK, Data mining

Copyright © 2017 International Journal for Modern Trends in Science and Technology
All rights reserved.

I. INTRODUCTION

We previously observed that the use of a particular programming language in an introduction to computing course tends to "annoy and distract attention from the core issue of algorithmic problem solving." In our experience, it also distracts attention from the teaching of algorithmic problem solving. Instructors spend class time where they expect students to have the most difficulty. Consequently, they often focus on syntactic difficulties that they expect students will encounter (e.g. the inappropriate use of " \leq " instead of " $<=$ " in C-based languages, or the improper placement of a semicolon).

Furthermore, we notice that most students are visual learners and that instructors tend to present information verbally. Studies [5,8] estimate that between 75% and 83% of our students are visual

learners. Because of their highly textual rather than visual nature, the use of either traditional programming languages or pseudo-code provides a counter-intuitive framework for expressing algorithms to the majority of our students.

RAPTOR, the Rapid Algorithmic Prototyping Tool for Ordered Reasoning, was designed specifically to address the shortcomings of syntactic difficulties and non-visual environments. RAPTOR allows students to create algorithms by combining basic flowchart symbols. Students can then run their algorithms in the environment, either step-by-step or in continuous play mode. The environment visually displays the location of the currently executing flowchart symbol, as well as the contents of all variables. Also, RAPTOR provides a simple graphics library, based on AdaGraph. Not only can

the students create algorithms visually, but also the problems they solve can be visual.

We teach an “Introduction to Computing” course that is required for all students. Previously, the algorithms block of this course was taught in Ada 95 or MATLAB. This summer, we taught the same course using RAPTOR. On the final exam, we tracked three questions that required the students to develop algorithms. The students were allowed to use any method to express their algorithm (Ada, MATLAB, flowcharts, etc.) Given this choice, students preferred to use flowcharts, and those taught using RAPTOR performed better.

RAPTOR is a visual programming development environment based on flowcharts. A flowchart is a collection of connected graphic symbols, where each symbol represents a specific type of instruction to be executed. The connections between symbols determine the order in which instructions are executed. These ideas will become clearer as you use RAPTOR to solve problems.

II. RELATED WORK

We use RAPTOR in CS110 for several reasons.

- The RAPTOR development environment minimizes the amount of syntax you must learn to write correct program instructions.
- The RAPTOR development environment is visual. RAPTOR programs are diagrams (directed graphs) that can be executed one symbol at a time. This will help you follow the flow of instruction execution in RAPTOR programs.
- RAPTOR is designed for ease of use. (You might have to take our word for this, but other programming development environments are extremely complex.)
- RAPTOR error messages are designed to be more readily understandable by beginning programmers.
- Our goal is to teach you how to design and execute algorithms. These objectives do not require a heavy-weight commercial programming language such as C++ or Java.

RAPTOR Program Structure

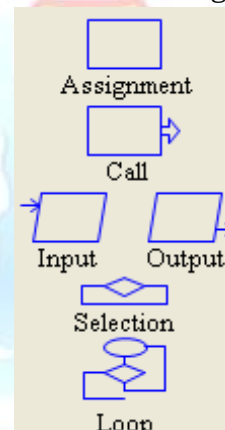
A RAPTOR program is a set of connected symbols that represent actions to be performed. The arrows that connect the symbols determine the order in which the actions are performed. When executing a RAPTOR program, you begin at the Start symbol and follow the arrows to execute the program. A RAPTOR program stops executing when the End

symbol is reached. The smallest RAPTOR program (which does nothing) is depicted at the right. By placing additional RAPTOR statements between the Start and End symbols you can create meaningful RAPTOR programs.



Introduction to RAPTOR Statements/Symbols

RAPTOR has six (6) basic symbols, where each symbol represents a unique type of instruction. The basic symbols are shown at the right. The top four statement types, Assignment, Call, Input, and Output, are explained in this reading. The bottom two types, Selection and Loops, will be explained in a future reading.

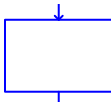
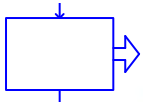
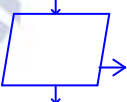


The typical computer program has three basic components:

- INPUT – get the data values that are needed to accomplish the task.
- PROCESSING – manipulate the data values to accomplish the task.
- OUTPUT – display (or save) the values which provide a solution to the task.

These three components have a direct correlation to RAPTOR instructions as shown in the following table.

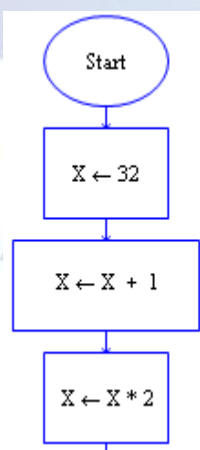
Purpose	Symbol	Name	Description
INPUT		input statement	Allow the user to enter data. Each data value is stored in a variable.

PROCESSING		assignment statement	Change the value of a variable using some type of mathematical calculation.
PROCESSING		procedure call	Execute a group of instructions defined in the named procedure. In some cases some of the procedure arguments (i.e., variables) will be changed by the procedure's instructions.
OUTPUT		output statement	Display (or save to a file) the value of a variable.

The common thread among these four instructions is that they all do something to variables! To understand how to develop algorithms into working computer programs, you must understand the concept of a variable. Please study the next section carefully!

RAPTOR Variables

Variables are computer memory locations that hold a data value. At any given time a variable can only hold a single value. However, the value of a variable can vary (change) as a program executes. That's why we call them "variables"! As an example, study the following table that traces the value of a variable called X.

Description	Value of X	Program
When the program begins, no variables exist. In RAPTOR, variables are automatically created when they are first used in a statement.	Undefined	
The first assignment statement, $X \leftarrow 32$, assigns the data value 32 to the variable X.	32	
The next assignment statement, $X \leftarrow X + 1$, retrieves the current value of X, 32, adds 1 to it, and puts the result, 33, in the variable X.	33	
The next assignment statement, $X \leftarrow X * 2$, retrieves the current value of X, 33, multiplies it by 2, and puts the result, 66, in the variable X.	66	

Expressions

The expression (or computation) of an assignment statement can be any simple or complex equation that computes a single value. An expression is a combination of values (either constants or variables) and operators. Please carefully study the following rules for constructing valid expressions.

A computer can only perform one operation at a time. When an expression is computed, the operations of the equation are not executed from left to right in the order that you typed them in. Rather, the operations are performed based on a predefined "order of precedence." The order that operations are performed can make a radical difference in the value that is computed. For example, consider the following two examples:

$$x \leftarrow (3+9)/3 \quad x \leftarrow 3+(9/3)$$

In the first case, the variable x is assigned a value of 4, whereas in the second case, the variable x is assigned the value of 6. As you can see from these examples, you can always explicitly control the order in which operations are performed by grouping values and operators in parenthesis. The exact "order of precedence" is

1. compute all functions, then
2. compute anything in parentheses, then
3. compute exponentiation (^,**) i.e., raise one number to a power, then
4. compute multiplications and divisions, left to right, and finally
5. compute additions and subtractions, left to right.

An operator or function directs the computer to perform some computation on data. Operators are placed between the data being operated on (e.g. $X/3$) whereas functions use parentheses to indicate the data they are operating on (e.g. $\text{sqrt}(4.7)$). When executed, operators and functions perform their computation and return their result. The following lists summarize the built-in operators and functions of RAPTOR.

basic math: +, -, *, /, ^, **, rem, mod, sqrt, log, abs, ceiling, floor

trigonometry: sin, cos, tan, cot, arcsin, arcos, arctan, arccot

miscellaneous: random, Length_of

The result of evaluating of an expression in an assignment statement must be either a single number or a single string of text. Most of your expressions will compute numbers, but you can also perform simple text manipulation by using a plus sign (+) to join two or more strings of text into

a single string. You can also join numerical values with strings to create a single string. The following example assignment statements demonstrate string manipulation.

```
Full_name ← "Joe " + "Alexander " + "Smith"
Answer ← "The average is " + (Total /
    Number)
```

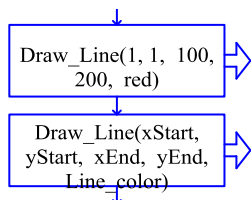
RAPTOR defines several symbols that represent commonly used constants. You should use these constant symbols when you need their corresponding values in computations.

pi is defined to be 3.14159274101257.
e is defined to be 2.71828174591064

Procedure Call Statement/Symbol

A procedure is a named collection of programming statements that accomplish a task. Calling a procedure suspends execution of your program, executes the instructions in the called procedure, and then resumes executing your program at the next statement. You need to know two things to correctly use a procedure: 1) the procedure's name and 2) the data values that the procedure needs to do its work, which are called arguments.

RAPTOR attempts to minimize the number of procedure names you need to memorize by displaying any procedure name that partially matches what you type into the "Enter Call" window. For example, after entering the single letter "d," the lower portion of the window will list all built-in procedures that start with the letter "d". The list also reminds you of each procedure's required arguments. In the example to the right, the lower box is telling you that the "Draw_Line" procedure needs 5 data values: the x and y coordinates of the starting location of the line, (x1, y1), the x and y coordinates of the ending location of the line, (x2, y2), and the line's color. The order of the argument values must match the arguments defined by the procedure. For example, Draw_Line(Blue, 3, 5, 100, 200) would generate an error because the color of the line must be the last argument value in the argument list.

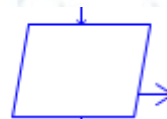


When a procedure call is displayed in your RAPTOR program you can see the procedure's name and the argument values that will be sent to

the procedure when it is called. For example, when the first procedure call on the right is executed it will draw a red line from the point (1,1) to the point (100,200). The second procedure call will also draw a line, but since the arguments are variables, the exact location of the line will not be known until the program executes and all the argument variables have a value.

RAPTOR defines too many built-in procedures to describe them all here. You can find documentation on all built-in procedures in RAPTOR's help screens. In addition, your instructor will introduce relevant procedures as we tackle various problem solving tasks in the coming lessons.

Output Statement/Symbol



In RAPTOR, an output statement displays a value to the MasterConsole window when it is executed. When you define an output statement, the "Enter Output" dialog box asks you to specify three things:

- Are you displaying text, or the results of an expression (computation)?
- What is the text or expression to display?
- Should the output be terminated by a new line character?

The example output statement on the right will display the text, "The sales tax is" on the output window and terminate the text with a new line. Since the "End current line" is checked, any future output will start on a new line below the displayed text.

When you select the "Output Text" option, the characters that you type into the edit box will be displayed exactly as you typed them, including any leading or trailing spaces. If you include quote marks (") in the text, the quote marks will be displayed exactly as you typed them.

When you select the "Output Expression" option, the text you type into the edit box is treated as an expression to be evaluated. When the output statement is executed at run-time, the expression is evaluated and the resulting single value that was computed is displayed. An example output statement that displays the results of an expression is shown on the right.

You can display multiple values with a single output statement by using the "Output Expression" option and building a string of text using the string plus (+) operator. When you build a single string from two or more values, you must distinguish the text from the values to be calculated by enclosing any text in quote marks (""). In such cases, the quote marks are not displayed in the output window. For example, the expression,

"Active Point = (" + x + "," + y + ")"

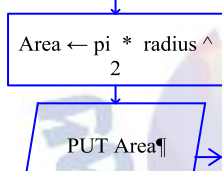
will display the following if x is 200 and y is 5:

Active Point = (200,5)

Notice that the quote marks are not displayed on the output device. The quote marks are used to surround any text that is not part of an expression to be evaluated.

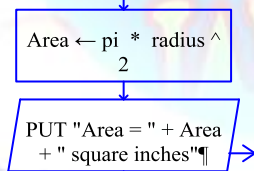
Your instructor (or a homework assignment) will often say "Display the results in a user-friendly manner". This means you should display some explanatory text explaining any numbers that are output to the MasterConsole window. An example of "non-user-friendly output" and "user-friendly output" is shown below.

Non-user-friendly output



Example output: 2.5678

User-friendly output



Example output: Area = 2.5678 square inches

Comments in RAPTOR

The RAPTOR development environment, like many other programming languages, allows comments to be added to your program. Comments are used to explain some aspect of a program to a human reader, especially in places where the program code is complex and hard to understand. Comments mean nothing to the computer and are not executed. However, if comments are done well, they can make a program much easier to understand for a human reader.

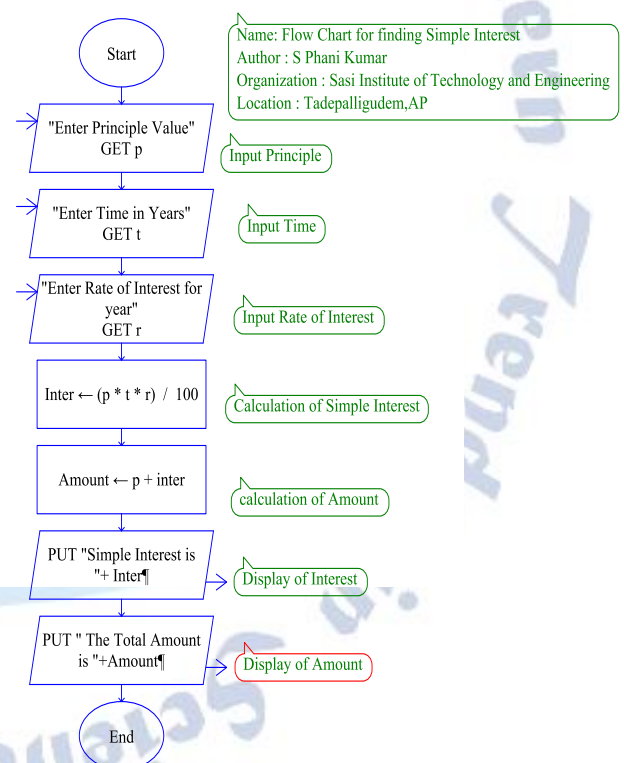
To add a comment to a statement, right-click your mouse over the statement symbol and select the "Comment" line before releasing the mouse button. Then enter the comment text into the "Enter Comment" dialog box, an example of which

is shown to the right. The resulting comment can be moved in the RAPTOR window by dragging it, but you typically do not need to move the default location of a comment.

There are three general types of comments:

- Programmer header – documents who wrote the program, when it was written, and a general description of what the program does. (Add to the "Start" symbol)
- Section description – mark major sections of your program to make it easier for a programmer to understand the overall program structure.
- Logic description – explain non-standard logic. Typically you should not comment every statement in a program. An example program that includes comments is shown below.

III. SIMPLE INTEREST FLOWCHART



IV. CONCLUSION

RAPTOR provides a simple environment for students to experiment with developing algorithms. Instructors can customize the environment and facilitate more interesting exercises by adding to the built-in procedures. Students, when given a choice, overwhelmingly prefer to express their algorithms visually using flowcharts. Even when

primarily taught a third generation programming language, 95% of students chose instead to use a flowchart on the final exam. The visual nature of the flowcharts makes it easier for students to follow the control flow in their programs, and to solve problems more easily.

In a small experimental section, we found that students using RAPTOR who entered the course with a much lower incoming GPA outperformed students with a higher incoming GPA using Ada or MATLAB.

REFERENCES

- [1] Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, Steven M. Hadfield RAPTOR: Introducing Programming to Non-Majors with Flowcharts
- [2] Dr. Wayne Brown Introduction to Programming with RAPTOR
- [3] Cardellini, L. An Interview with Richard M. Felder. Journal of Science Education 3(2), (2002), 62-65.