RAPTOR - A Vehicle to Enhance Logical Thinking

Dr. Nikunja Swain P.E., South Carolina State University

Dr. Swain is currently a Professor at the South Carolina State University. Dr. Swain has 25+ years of experience as an engineer and educator. He has more than 50 publications in journals and conference proceedings, has procured research and development grants from the NSF, NASA, DOT, DOD, and DOE and reviewed number of books on computer related areas. He is also a reviewer for ACM Computing Reviews, IJAMT, CIT, ASEE, and other conferences and journals. He is a registered Professional Engineer in South Carolina and ETAC of ABET reviewer for Electrical Engineering Technology and Computer Engineering Technology.

Ms. Wanda Moses, South Carolina State University
Dr. James Allen Anderson P.E., South Carolina State University
Ms. Cynthia T Davis, SC State University

EDUCATION 1979 B. S. Magna Cum Laude, Mathematics Education Benedict College, Columbia, South Carolina

1982 M. S. Computer Science Atlanta University, Atlanta Georgia

1992 - 1996 Additional Study, Mathematics University of South Carolina, Columbia, South Carolina South Carolina State University, Orangeburg, South Carolina

PROFESSIONAL EXPERIENCE 1990 – Present Instructor of Computer Science South Carolina State University Orangeburg, South Carolina

1987 - 1990 Assistant Professor of Computer Information Science Morris College Sumter, South Carolina

1986 - 1987 Instructor of Computer Science Benedict College Columbia, South Carolina

1982 - 1986 Instructor of Computer Science Atlanta Junior College Atlanta, Georgia

PUBLICATIONS King, V. J., Davis, C., 2007. A Case Study of Urban Heat Islands in the Carolinas. Journal of Environmental Hazards, volume 7 (4), 353-359.

PROFESSIONAL ASSOCIATIONS Recording Secretary, Kappa Mu Epsilon Mathematics Honor Society

Member, Association of Computing Machinery (ACM)

RAPTOR – A Vehicle to Enhance Logical Thinking

Abstract

Research shows that logical and critical thinking are very important for writing efficient programs but unfortunately many of our students lack these skills. This in turn affects our ability to produce graduates to meet the needs of increasingly computer dependent industries and to maintain our nation's position as the global leader of high technology arena. The educators and administrators are challenged to find ways to engage and promote success and retention of students while maintaining standards in introductory computing courses. We believe that this problem can be effectively addressed by enhancing students' logical and critical thinking through the use of visual programming tools such as RAPTOR in introductory computing courses. RAPTOR is a visual programming development environment based on flowcharts. Students can build simple procedural programs without learning the details of a language. These features of RAPTOR has helped us in providing an Interdisciplinary Integrated Teaching and Learning experiences that integrates team-oriented, hands-on learning experiences throughout the engineering technology and sciences curriculum and engages students in the design and analysis process beginning with their first year. The objective of this paper is to discuss our experiences with the use of RAPTOR in various science and technology courses.

Introduction

Automation is becoming part and parcel of every industry, and industries need a trained workforce to manage this new development. Engineering and technology graduates must have a comprehensive background covering a wider range of technical subjects. The graduates must be proficient in the use of computers, engineering and scientific equipment, conducting experiments, collecting data, and effectively presenting the results [1, 2, 3, 4]. In addition to having a good training in their respective disciplines, all graduates must be well-trained in courses and laboratories dealing with computer programming; computer aided design; computer organization and architecture; and others. Unfortunately most of our graduates do not perform well in required introductory computing courses due to lack of logical and critical thinking skills, leading to poor performance in subsequent courses. This affects their ability to be competitive in the global economy. We are challenged to find ways to address this problem.

One cost-effective way of achieving this is through the use of visual programming tools and a number of such tools are available for this purpose. These visual programming tools play an important role in education and are used to enhance logical and critical thinking skills. In his doctoral dissertation, titled "Using Flowcharts, Code and Animation for Improved Comprehension and Ability in Novice Programming"[5], Dr. Andrew Scott discussed in detail the nature of the difficulties and skill deficiencies novices have when learning programming and the potential of visualization and how the use of dynamic structured flowcharts may be effective in aiding the conceptual understanding and problem solving skills of novice programmers. According to Dr. Scott, "In comparison to other subjects (studied currently and previously) the novice programmer will encounter an unusually large number of errors and impasses. The large number of errors encountered can lead to de-motivation and anxiety which will inevitably have an impact on a beginner's self- confidence and efficacy in programming. This is another major

factor contributing to the proportionally high dropout rates. This is another major factor contributing to the proportionally high dropout rates." Out of the three types of errors (syntax error, run time error and logical error), syntax error is the most frequently error encountered by a programmer and this is true for the novice programmer.

The fundamental building blocks of any program are variables, sequence (input, output, assignment), selection (case, condition/boolean, if-then-else), iteration (for, while, do while), arrays, file I/O, and functional decomposition (functions, procedures) and a novice programmer must have some understanding of few of these concepts. The visual programming tools can be effective and helpful tools to aid the novice programmers to learn these concepts.

Majority of the visualization tools are based on flowcharts because flowcharts use a graphical representation to describe the detailed logic of a process or set of rules. Flowcharts can be easily understood with little or no prior training. This small learning curve is due in part to the simplicity of their notation; any program can be expressed using only five basic flowcharting symbols as shown in Figure 1 below:

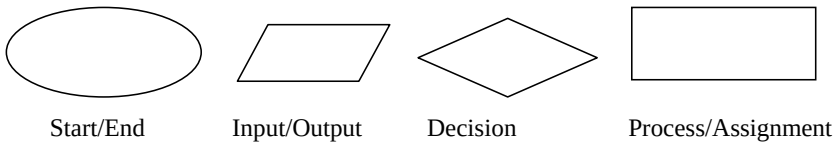

Figure 1 – Basic Flowchart Symbols

Flowcharts are ideal for modeling and visualization of small programs usually encountered by novice programmers. It allows the novice to envisage the flow of execution, the semantics of the conditional structures (i.e. if, if else, while, for etc) and how the individual pieces of a program interact to form higher level concepts, whilst limiting the syntactic overheads of a programming language. Dynamic flowcharts are used to animate the dynamic nature of a computer program, its flow of execution, the changing state of program variables and the interactivity between program components. A dynamic flowchart also has better cognitive efficiency, as the viewer does not need to manually keep track of the state of program data, thus minimizing the amount of information the learner has to hold in working memory or write down. By obeying the key rule of structured programming, that every structure has only one entry and exit point, flowcharts can easily communicate a well-defined and structured program. Subsequently, the transition from flowchart to structured code will be more apparent, requiring less reinterpretation. This should make the novice's transition from problem specification through to resultant structured code much simpler and more cognitively efficient.

There are number of visualization tools that use flowcharts. Dr. Scott in his dissertation compared fourteen of these visualization tools starting from 1992. He also summarized his findings in a Table as shown in Figure 2.

Feature	BACCII	FLINT	EC	FCI	Raptor	SFC	VL	IP	DF	A&Y	B#	PG	G&G	CVF
Year Published	1992	1999	2002	2003	2004	2004	2004	2005	2006	2006	2006	2007	2007	2009
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														*
Generated code compiles without modification	*							*	*		*			
Code and Flowchart Displayed Concurrently.						*				*	*		*	*
Synchronised highlighting of flowcharts and code										1/2	1/2		*	*
Synchronised Visual Execution of Flowcharts and code										*			*	
Non visual execution				*			*							
Visual execution		*	*	*	*		*	*		*	*	*	*	
Variable inspector	*	*	*	*	*			*		*	*			
Strongly typed	*							*	*	*	*			*
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	Any	Win	Win	Win/ Mac	Win

Figure 2 – Comparison of few of the existing visualization tools

We decided to use RAPTOR because of the following:

- it is available over internet without any cost
- Size of RAPTOR is small and takes little time to download
- It is easy to install
- Good documentation and user guide is available in RAPTOR website
- Textbook dealing with RAPTOR is available to be used as supplementary text
- The website is frequently updated

- The RAPTOR program is easy to use
- Raptor provided capability of generating JAVA, C++, C and Ada

RAPTOR is designed and required to be successful in programming courses[6].We will be discussing some of the features RAPTOR visual programming tool and the instructional modules developed using RAPTOR below:

Features of RAPTOR Visual Programming Tool

Features of RAPTOR [6,7,8]

RAPTOR is a free visual programming tool developed by computer science faculty at the United States Air Force Academy. It is developed using C and Ada and freely available. Some of the features of RAPTOR are the following:

Some of the features of RAPTOR are [6]:

- The RAPTOR development environment minimizes the amount of syntax the student 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 students to follow the flow of instruction execution in RAPTOR programs.
- RAPTOR error messages are designed to be more readily understandable by beginning programmers.

When students are learning to develop algorithms, they very often spend more time dealing with issues of syntax rather 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 visual programming environment, designed specifically to help students envision their algorithms and avoid syntactic baggage. RAPTOR programs are created visually and can be executed visually by tracing the execution through the program. Required syntax is kept to a minimum. Students preferred expressing their algorithms visually, and were more successful creating algorithms using RAPTOR than using a traditional language or writing flowcharts. RAPTOR has 6 basic statements: Input, Output, Assignment, Call, Selection, and Loop. Each of these statements is indicated by a different symbol in RAPTOR as shown below in Figure 3.

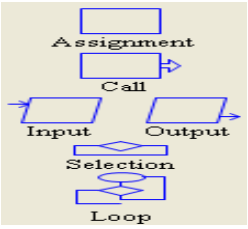

Figure 3 – Symbols in RAPTOR

Examples of RAPTOR Visual Programming Exercises [7, 8]

Introductory Computer Science Course CS 151

We use number of different exercises to introduce our Freshman STEM students to programming concepts in our introductory computer science course. Concepts such as variables, inputs, outputs, processes, loops, and arrays are introduced without relying on any specific programming language. The students used RAPTOR programming to achieve these objectives. Examples of few of these exercises are presented below:

Example 1 - Write a program to determine the real roots of a Quadratic Equation (we will discuss complex roots in another program): $ax^2 + bx + c = 0$. User Inputs: a, b, and c; Program Outputs: Root1 and Root2; Equation to be used: $Root1,2 = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. Program addresses division by zero and tested with a=1, b = 3, c=1. For a = 1, b = 3, c = 1, the roots are Root 1 = - 0.3812 and Root 2 = -2.618. Figure 4 below shows the C programming language solution and Figure 4 shows solution using RAPTOR.

Source Code

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  /* Declare variables.  */
  double a, b, c, root1, root2;
  double temp1, temp2, temp3;
  int y = 0;

 /* Get time value from the keyboard.  */
  printf("Enter a non-zero value for a: \n");
  scanf("%lf",&a);
  printf("Enter a value for b: \n");
  scanf("%lf",&b);
 printf("Enter a value for c: \n");
  scanf("%lf",&c);

/* Check if a = 0 & perform calculations */

 if (a !=0)
   {
    temp1 = -b/2*a;
    temp2 = (b*b - 4*a*c);
   }
    else
   printf("a can not be zero, Enter a non zero value for a and run program again\n");

/* Check for real and complex roots */
     if (a!=0 && temp2 >=0)
     {

    temp2 = (sqrt(temp2)/2*a);
    printf("Real Roots\n");
        root1 = temp1+temp2;
        root2 = temp1-temp2;
        printf("Root1 = %lf \n", root1);
        printf("Root2 = %lf \n", root2);
    }
    else
       printf("Complex Roots\n");

  /*  Exit program.  */
  return 0;
}
/*------------------------------------------------------------------*/
```

Program Output

```
Enter a non-zero value for a:
1
Enter a value for b:
3
Enter a value for c:
1
Real Roots
Root1 = -0.381966
Root2 = -2.618034

Process returned 0 (0x0)   execution
time : 11.364 s
Press any key to continue.
```

Figure 4 – Solution with C Programming Language (Program outputs are from CODEBLOCK C Compiler)

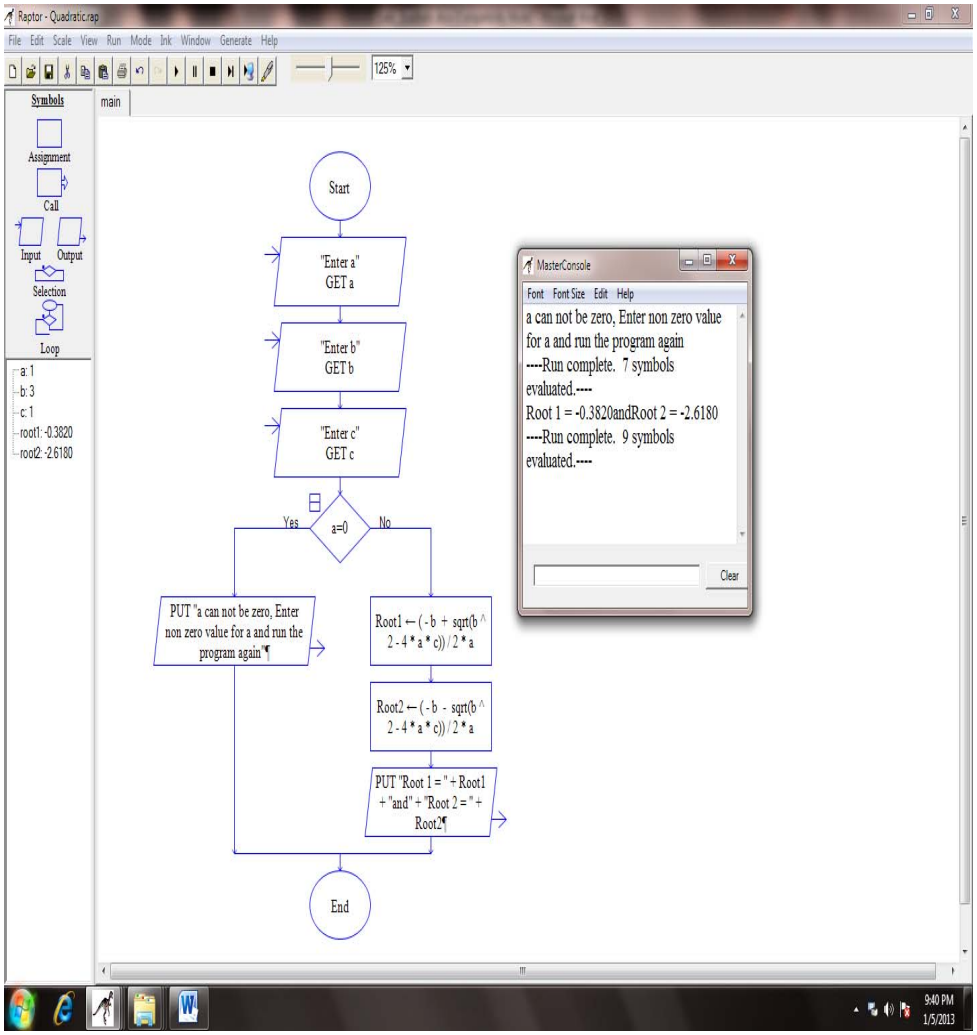

Figure 5 – Solution using RAPTOR Graphical Programming Language

As seen from figures 4, the C programming language solution requires understanding of the syntax and requires a good understanding of variables, loops, input/output and processes. This may be time consuming for many students and may be intimidating. On the other hand, the RAPTOR solution is independent of syntax and does not require detailed understanding of loops, input/output and processes. It takes less time to understand the concept since only 6 symbols are used to solve the problems. RAPTOR lacks the flexibility and veracity of programming languages like C, C++, and JAVA and suitable for teaching introductory programming.

Example 2 - Review the Flow Chart shown below in Figure 6 and show count and sum for each pass through the loop in the table provided.

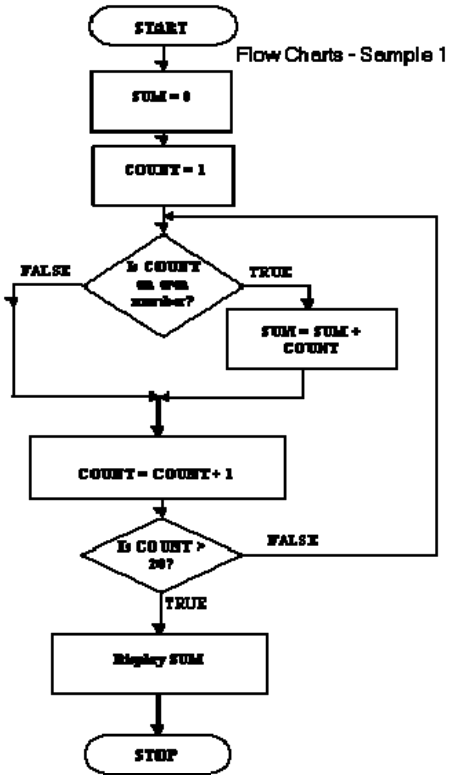

Figure 6 – Flow chart for Example 2

SOLUTION

COUNT	SUM	COUNT	SUM
1	0	12	42
2	2	13	42
3	2	14	56
4	6	15	56
5	6	16	72
6	12	17	72
7	12	18	90
8	20	19	90
9	20	20	110
10	30		
11	30		

Example 3 - For the algorithm given below, determine the sum if the values are: 0, 5, 9, 7, 25, 30 using RAPTOR.

1. Start; 2. Sum = 0; 3. Get a value; 4. sum = sum + value; 5. Go to step 3 to get next Value
6. Output the sum; 7. Stop

RAPTOR PROGRAM USING CENTENNIAL and OUTPUT – The program asks the user to enter the numbers. Program stops when user enters the centennial value of -999 and displays the sum. Solution using RAPTOR is shown in Figure 7.

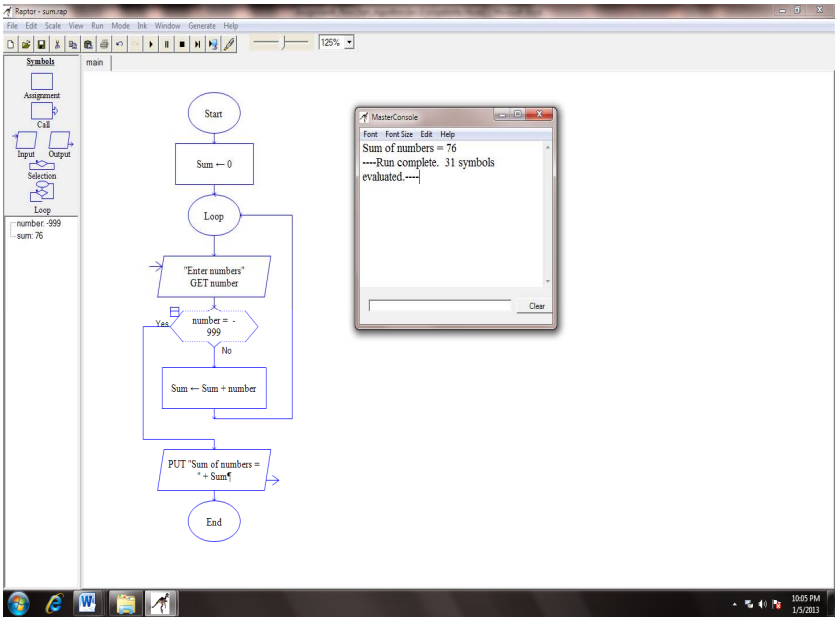

Figure 7 – RAPTOR Solution for Example 3

RAPTOR PROGRAM USING COUNTER and LOOP – Example 3 is repeated with a counter. In this case the number of inputs is known in advance. A counter is set with this value. As user enters the number, the counter is decremented by one. Sum is updated. This process is repeated until counter is zero. The program stops when the counter is zero and the Sum is displayed. The sum of 0, 5, 9, 7, 25, and 30 is equal to 76 and RAPTOR solution calculated 76 as shown in Figure 7. A C programming solution for this example is shown in Figure 8. Figure 9 shows the corresponding RAPTOR solution. As mentioned earlier, the RAPTOR program is found to be free of syntax and easy to implement. Students completed the assignment easily RAPTOR. Many students encountered problems in comprehending the FOR Loop in the C programming solution.

C Program

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  /* Declare variables.  */
  double sum = 0, number;
  int count = 6;
  int i;

  /* Get time value from the keyboard.  */
  for (i=0; i<count; i++)
  {
     printf("Enter numbers: \n");
     scanf("%lf",&number);
     sum = sum + number;
  }

   printf("The sum = %.2f\n", sum);

  /*  Exit program.  */

  return 0;
}
/*--------------------------------------------------------------*/
```

Program Output

```
Enter numbers:
0
Enter numbers:
5
Enter numbers:
9
Enter numbers:
7
Enter numbers:
25
Enter numbers:
30
The sum = 76.00
Process returned 0 (0x0)   execution time
: 13.997 s
Press any key to continue
```

Figure 8 – C Programming solution

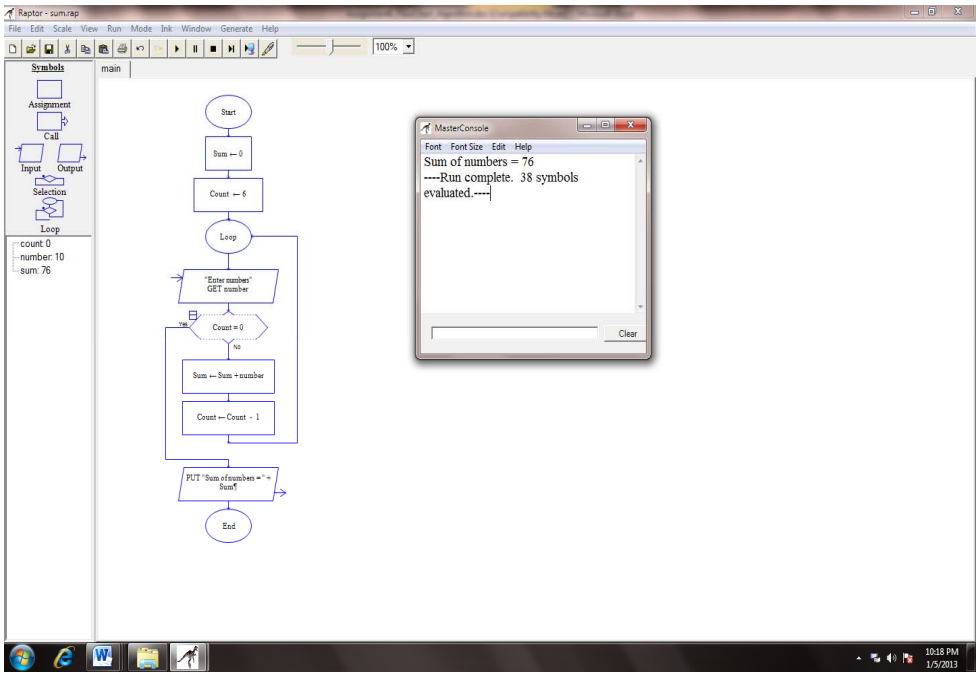

Figure 9 – RAPTOR Solution

Third Year Computer Science Course CS 300 – Digital Logic

We also used RAPTOR in our CS 300 course. The course prerequisite for this course is JAVA programming, the first programming course in our curriculum. The students are required to complete a programming project dealing with number system conversion and they use JAVA programming to achieve this objective. Some of the students have difficulty in completing the JAVA programs and last semester we asked all students to use, both RAPTOR and JAVA for this objective. Our objective was to understand student perceptions about RAPTOR. The problem statement and corresponding RAPTOR program is shown below in Example 4 and Figure 10.

Example 4 – Design a program to convert Binary number to equivalent Decimal number.
Solution – For binary value of 1011, Decimal value is equal to 11.

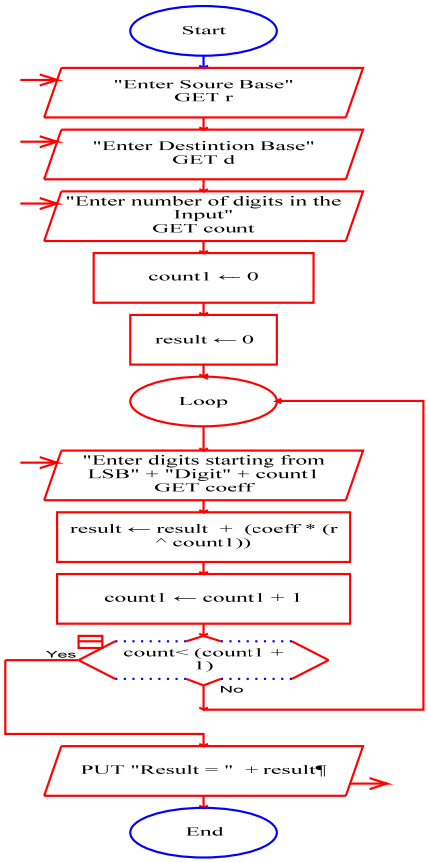

Figure 10 – Raptor Solution to Example 4

Summary and Conclusions

The sample modules presented above are user friendly and performed satisfactorily under various input conditions. These and other modules helped the students to understand the concepts in more detail. The students were able to compare their theoretical calculation with the RAPTOR output. RAPTOR program helped the students to enhance their logical and critical thinking skills. RAPTOR can be used in conjunction with other teaching aids to enhance student learning in various courses and will provide a truly modern environment in which students and faculty members can study engineering, technology, and sciences at a level of detail.

In the CS 151 course, 63 students used RAPTOR during 2012 (Spring, Summer, and Fall). 80% of the students were satisfied with the easy to use nature of the program. 70% of the students completed all assignments without any help and they answered questions related to variables, processes and assignments, selections and iterations clearly. We plan to monitor these students in subsequent programming courses for their performance and present the findings in another paper. In the CS 300 course, we used RAPTOR in 2012 Fall semester. There were 12 students and most of them (90%) liked RAPTOR for small problems (small flowcharts). Since the flowchart for their number system converter was large and required a deeper understanding some of the features of RAPTOR, only 25% of the students thought RAPTOR is helpful for solving complex problems. Since the students in this group have prior JAVA and/or C coding experience, majority of the students in this group thought that programs like JAVA and/or C are much easier for complex problems. We plan to extend the use of RAPTOR in other courses such as Numerical Analysis (M 410) and Computer Applications in Mechanical Engineering Technology (MET 425). These students have very limited experience in programming using high level programming languages such as C, C++, FORTRAN, and JAVA. Since RAPTOR is free from syntax and easy to use, the RAPTOR programming environment may be suitable for these students.

References

1. Swain, N. K., Korrapati, R., Anderson, J. A. (1999) “Revitalizing Undergraduate Engineering, Technology, and Science Education Through Virtual Instrumentation”, NI Week Conference, Austin, TX..
2. Elaine L., Mack, Lynn G. (2001), “Developing and Implementing an Integrated Problem-based Engineering Technology Curriculum in an American Technical College System” Community College Journal of Research and Practice, Vol. 25, No. 5-6, pp. 425-439.
3. Buniyamin, N, Mohamad, Z., 2000 “Engineering Curriculum Development: Balancing Employer Needs and National Interest--A Case Study” – Retrieved from ERIC database.
4. Kellie, Andrew C., And Others. (1984), “Experience with Computer-Assisted Instruction in Engineering Technology”, Engineering Education, Vol. 74, No. 8, pp712-715.
5. Scott, A, 2010. “Using Flowcharts, Code and Animation for Improved Comprehension and Ability in Novice Programming” - University of Glamorgan, UK.
6. RAPTOR – Retrieved from http://raptor.martincarlisle.com/
7. W. Brown, “Introduction to Programming with RAPTOR,” Retrieved from http://www.scribd.com/doc/13826372/Introduction-to-Programming-With-RAPTOR
8. Venit, S and Drake, E. (2011) – Prelude to Programming – Concepts and Design, 5th Edition, Addison-Wesley.