

A New Technique for Teaching Introductory Programming Course using Constructivist Approach

Zulfiqar A. Khan

Sir Syed University of Engineering & Technology (SSUET), Karachi, Pakistan

Abstract

Programming is difficult to learn but easy to forget. Time constraints also exaggerate the problem. Teachers normally teach in an A-Z style, which hinders them to give judiciously enough time to address difficult topics lying at the bottom half of the course. Surveyed literature shows that active learning techniques, wrapped up in series of examples adhering to a constructivist approach can greatly help to improve programming skills. This is because these specially tuned programming examples constantly refresh students' knowledge by relating it to the previously accomplished tasks. This paper tries to tackle the issue of teaching difficult topics first, in the context of constructivist approach to first semester Computer Engineering students. Results indicate that it turned out to be a healthy exercise in improving the performance and developing programming maturity among the students.

1. Introduction

The idea of constructivist approach is not new. It is a common practice to build larger programs using already created smaller programs. Surveyed literature shows that this idea has served as a launching point for several research studies in IT education. Carbone et al. (2001) used it in the context of highlighting poor learning tendencies. Hadjerrouiti (1999) and Whittington (2005) applied the constructivist approach for teaching OOP. Djordjevic (2007) showed how graphic based progressive assignments can be designed by building new knowledge upon the prior knowledge. However, both Whittington and Djordjevic have described an initial problem scenario and added more complexity into it in a progressive style. The distinguishing feature of this paper is to focus on weighted class exercises concerning different problem domains. The relationship factor between the new knowledge and the already grasped concepts is expressed through functions from the very beginning. This is done in two ways: either by upgrading the previous version of functions to suit current programming requirements or by encapsulating it in a brand new function as a "function call" statement.

There were several benefits of this process. All programming examples were presented to the students in a descriptive manner as a unified single problem. But soon the students were motivated to decompose the programming examples into functional abstractions by illustrating the "Top Level" of the code. Some of the functional

abstractions visualized by the students had cooked solutions (previously done). This is where the constructivist approach comes into play while the others were unknown to them, thus forcing the students to exhibit their mental ingenuity. In short, the combination of the above two types of functional abstractions on one hand helped the teachers to progress towards the culmination of the course and on the other hand helped the students to retain their knowledge. It is worth mentioning that the topic of Functions is considered as a difficult topic and is normally taught in the middle of an introductory programming course of C-language. By teaching this difficult topic first, students got more time to understand it. This has been stressed also in Whittington (2003). One unexpected outcome of the research comes up as a result of implementing a technique for program representation. This technique unites the concepts of pseudo code and flowcharts along with some new ideas. The main theme of these ideas is to highlight the relationship between the previous and the newly created functions which is the core of this research. This is a descriptive research carried out at the SSUET Computer Engineering department. The population includes the students of first semester, taught by the author during the years 2006-2007. Findings are based upon student comments and mid-term exams (M.T.E.s) and in-class activities. Normal class size is 60 students and there were no attendance requirements for appearing in M.T.E. This paper describes the sample questions for in-class activities in a week-wise fashion, the top-level of the program, built-in functional abstractions (F.A), the pedagogical goals, student comments, and the results of this study.

2. Background

The examples discussed in this paper were used in an introductory "C" language programming course. The mode of teaching was Active Learning. The major topics of the course were: C Building Blocks, Loops, Decisions, Functions, Arrays, and Strings.

Earlier in 2006, the same course was taught by the author but the M.T.E results showed that students faced problems normally in the topic of function. These problems became evident during the assessment of midterm exams. M.T.E consists of four questions: error detection, output evaluation, coding a program, and definition. All these were related to functions.

Both Whittington (2005) and Djordjevic (2007) have used a constructivist approach for developing assignments and have reported promising results. However, in case of assignments, students may resort to unfair means (Djordjevic, 2007). This models our course around programming exercises, based upon a Z-A style, which were delivered during the break-in period of Active Learning sessions up to the midterm exam (first eight weeks). Though this pattern in examples was followed even after midterm exams, this research only reports the results of first eight weeks. Stepwise refinement method was applied for solving these programming exercises. This is used in many software development organizations. First, the top level is visualized using functional abstractions. Then these functional abstractions are defined either from scratch or by using previously worked out code. Finally, the program is executed and tested for required outputs.

Use of functions from the beginning of the course turned out to be a difficult proposition for students. For using any function in the program, students have to supply three pieces of information: its declaration, its definition, and its abstraction in the top level, which is programmatically referred as the function call statement. This caused some students to complain that the material was difficult and hard to grasp. These students were handled patiently by giving more attention during the lab sessions and consultation hours. Towards the end of first half (before the midterm), the majority of students was able to demonstrate command on all the three sections of functions. There were a few dropouts and like the previous time a few students did not appear in the midterm exam due to personal reasons. Copying was strictly monitored during in-class activities. Students were distributed separate sheets for coding the in-class activities. Afterwards, sheets of some of the students were collected in a random fashion and returned back in the labs to execute these programs. Successful students were rewarded credits for their work. All students went through this exercise twice during the eight weeks' duration. In this process, help was also sought from teaching assistants, which are normally not denied in a programming course. This practice also was followed in 2006 and meant for improving class attendance and as a post-test for weekly activities. The only difference in 2007 is the use of constructivist oriented programming exercises in conjunction with Z-A teaching style.

2. Program Representation

The problem was to design programming examples which exhibit constructivist approach in the context of functional abstractions. Due to this peculiar nature of problem, it was felt necessary to illustrate the top level of the program using some program representation technique. This would help to bring forth the functional abstractions involved and to establish their relationship with the previous work. Moreover, the designed program representation technique

must be powerful enough to demonstrate the chain effect hidden in the process of function calls. When a function is invoked, then it may call other functions. Commonly used program representation techniques like flowcharts and UML lack the mechanisms for showing function call chain effect behavior. The program representation technique mentioned in this paper is capable of demonstrating this deficiency and is termed as "Function based program representation technique" (FPRT). By using FPRT in a very small space, all the functional abstractions involved in program development become evident. Along with function call, some other statements were also included in this program representation diagram in order to clarify the diversity of topics covered.

3. Programming Examples using Constructivist Approach

Programming examples provided during the break-in period of active learning techniques were actually presented to the students in the form of programming exercises. They not only provided respite to the students with low absorption capacity but also served as a catalyst to strengthen their concepts. However, in the context of this research, this drill served two purposes: invoking mental ingenuity of students and relevance to the previous knowledge that is also termed as the constructivist approach. Thus, the basic theme of each programming example was to draw a clear line between what the students had done in the past and what they were going to accomplish presently. It was also thought that it was necessary that programming examples be subjected to several contortions to reflect the mathematical, textual, and real life problems.

The connectivity factor between the past and current concepts within the programming examples was achieved through functional abstractions. But the question was how to switch to the functional abstractions by relating it to the previous knowledge of students at the launching phase. Built in TurboC mathematical functions like sin (sine), cos (cosine), sqrt (square root), and pow (power) served as good starters in this context and were used to project the concepts of how to invoke functions and how to pass and return values from them. Once these elementary concepts were unraveled to the students, the class moved towards creating their own user defined functions.

The programming exercises that were considered in this research also exhibited the characteristics of two commonly used software development models, namely Spiral and Water Fall models. These two models were used to order the presentation of concepts related to data, operations, and programming constructs which are inherent to any program. The flow of concepts in the data and operation cycles demonstrate the Water Fall model as shown in Figure 1 and Figure 2. However, advancements in the programming

construct cycle projects the Spiral model (see Figure 3), which was in close resemblance to what has been illustrated in Djordjevic (2007). The difference lies in the programming language domain. The description of these cycles is provided below, which is followed by weekly representation of sample questions.

3.1 Data Cycle

Progressive exercises were designed in such a manner that they introduced several techniques for supplying data to the program within the course boundaries. Initially bombarded examples aimed at making the students comfortable with the given data and constant values. Next, students were taught how to input data programmatically. Subsequently, students learned how to restrict the input data, for example, how to restrict the numerical input data to only positive values. This process was further extended to input values bounded by a minima and a maxima. Finally, students were guided how to randomly generate the program data.

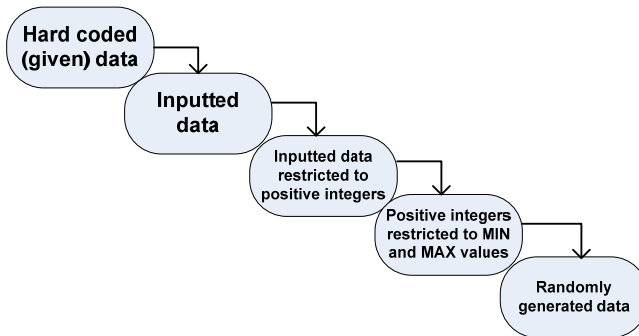


Figure 1. Data cycle

3.2 Operation Cycle

In the operation cycle, the application of constructivist approach is exploited from the very beginning. This was done through the blessings of the college mathematic courses in which students are given thorough knowledge of trigonometric functions and algebraic formulas. This led to start the operation cycle by primarily focusing on the sine, cosine, square root, and other built-in functions having mathematical applications. After doing several examples with these mathematical functions, students become quite enthusiastic to develop their own functions. Initially the user-defined functions were coded to return the values generated by the above mentioned built-in functions but later on they were used to evaluate complex mathematical and graphical formulas. Culmination of this process resulted in discussing series generation problems in the context of iterative constructs. This scenario was extended to develop shapes using graphical, numerical, and the asterisk characters. Finally, numerical data searching and ordering techniques were exhorted to the students.

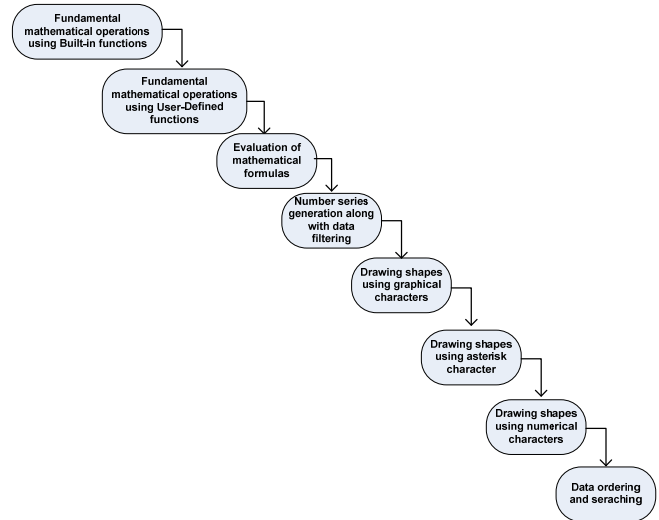


Figure 2. Operation cycle

3.3 Programming Construct Cycle

Due to the peculiar teaching style of focusing on functions from the very beginning in this research, it was not possible to discuss the programming constructs in a normal flow as done in many text books. After introducing the “main” function, which is the top level of the program, statements to

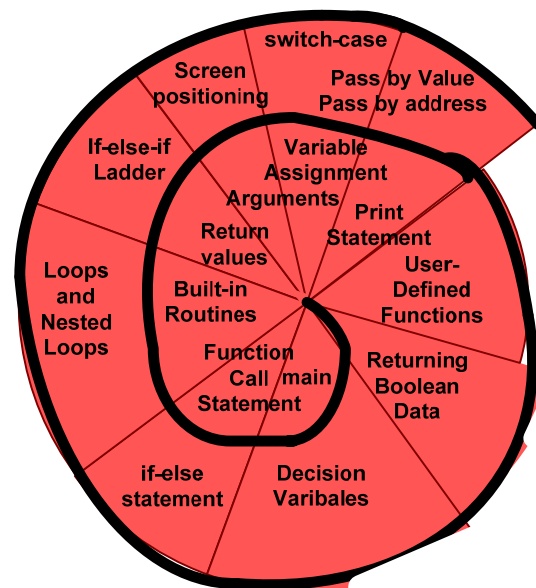


Figure 3. Programming construct cycle
invoke the built-in functions were presented to the students. During this process, the concept of return values, arguments, and variable assignments was also brought to light. This led

the class to write their first program that also wrapped the “printf” statement along with all its cumbersome formatting details. In the next phase, user-defined functions were coded and shown how they could return Boolean data and in this context, decision variables and statements comprising of if-else constructs were highlighted. Afterwards, programs were empowered with loops and the problem domain was shifted to developing shape generation programs using screen positioning statements. Finally, the switch-case statement and its relationship with break statement were explained for writing menu-driven programs. These examples also dealt with the so far untouched of pass-by-value and pass-by-address parameter passing.

4. Week-wise Discussion of Sample Questions

4.1 Week1

4.1.1 New Topics

Built-in Mathematical functions of Turbo C Library, variables and data types, printf statement, format specifiers, function call statement, and arguments.

4.1.2 Top Level Explanation

Top Level along with other details is shown in Figure 4. Top Level indicates that built-in mathematical functions are invoked with appropriate arguments and results are printed. For instance, sine function is invoked using double variable dou. Note that the program data consist of given values. Return values are avoided for clarity.

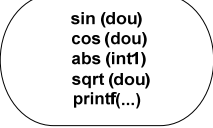
Top Level	Pedagogical Goals	F.A built-in	Questions
	From math background to built-in F.A. Declare variables, assign and display values. User-Defined functions?	sin (...), cos (...), abs (...), sqrt (...), printf (...)	Q1. Calculate sine, cosine, square root, and absolute value of given data.

Figure 4. Description of sample Question of Week1

4.1.3 Application of Constructivist Approach

Previous college level mathematical background is exploited to start the programming process.

4.2 Week2

4.2.1 New Topics

Function declaration, definition and return values, and typecasting.

4.2.2 Top Level Explanation of Q1

Top Level for the sample questions of this week along with other details is shown in Figure 5. Top Level indicates that the user defined function CalSin(...) is invoked using integer argument, int1 that is typecast into a double variable dou. Sine of dou is evaluated and the result is returned as dou1.

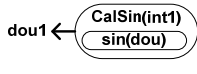
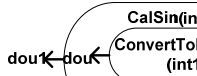
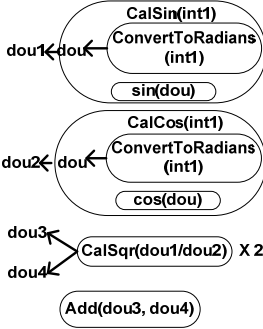
Top Level	Pedagogical Goals	F.A built-in	Questions
	From built-in sin(...) to user-defined CalSin(int)	sin(...), printf(...)	Q1. Calculate sine of a given value using a function
	Modified CalSin(int) to convert the argument into radians	sin(...), printf(...)	Q2. Calculate sine of a given value by converting the value into radians using a function
	Developed Mirror function CalCos(int), add and square user-defined functions. Input values?	sin(...), cos(...), printf(...)	Q3. Verify the results: $\sin^2\theta + \cos^2\theta = 1$, using given values.

Figure 5. Description of sample Questions of Week2

4.2.3 Application of Constructivist Approach

User defined function CalSin(...) is introduced, which finds the sine of its argument using the built-in sin(...) function which was covered in previous week.

4.2.4 Top Level Explanation of Q2

In this program, the CalSin(...) function is modified. It calls the function ConvertToRadians(...) which transforms its integer argument, int1, into radians. The radian value is returned as a double variable dou. CalSin(...) then calculates the sine of dou and returns the result as a double value dou1.

4.2.5 Application of Constructivist Approach

CalSin(...) is modified by providing a statement to call ConvertToRadians(...). Again, the logic of ConvertToRadians is borrowed from college trigonometry.

4.2.6 Top Level Explanation of Q3

Here the Top Level first calls the CalSin (...) and then the CalCos(...) functions, one after the other. Both these functions call ConvertToRadians(...) function. The radian value returned by these functions is then used to find the sine and cosine values which are stored in variables dou1 and dou2 respectively. Sine and cosine values are then squared by calling the CalSqr(...) function twice. This is indicated by “x 2” term in Figure 5 after the CalSqr(...) ellipse. CalSqr(...) returns dou3 and dou4 values as a result of two separate invocations. Finally the squared values are added by invoking the Add(...) function.

4.2.7 Application of Constructivist Approach

CalCos(...) function is introduced, which is a counterpart of previously introduced CalSin(...). More functionality is added to square and add the values.

4.3 Week3

4.3.1 New Topics

Input statements, Expressing true and false values, if and else statement, Conditional operator, and Arithmetic operators.

4.3.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 6. Top Level calls the InputInt() function twice. InputInt() implicitly calls the scanf(...) function and returns the radius of the circle (int2) and the angle at the required point on the circle (int1) as a result of two separate invocations. CalX(...) and CalY(...) functions are invoked with these two integer values and after processing, these functions return the respective double values, dou1 and dou2, for the coordinates of the point on the circle. Processing of CalX(...) and CalY(...) involves invoking CalCos(...) and CalSin(...) functions respectively, which were introduced in the last week.

Top Level	Pedagogical Goals	F.A built-in	Questions
	Input values and then use them for calculations instead of given values	sin(...), cos(...), scanf(...), printf(...)	Q1. Input the radius and the angle for a point on a circle and then calculate the polar coordinates of the point using $x = \cos\theta$ $y = \sin\theta$
	Implementing Mathematical formulas and equations, show the limitations of square root and division operations. Repetition of same task?	scanf(...), printf(...), sqrt(...), pow(...), ceil(...), floor(...)	Q2. Apply quadratic formula to determine the roots of quadratic equation. Input the required parameters.

Figure 6. Description of sample Questions of Week3

4.3.3 Application of Constructivist Approach

CalX(...) and CalY(...) functions are created which call the previously introduced CalCos(...) and CalSin(...) functions respectively. Instead of given values, data is inputted using InputInt() function.

4.3.4 Top Level Explanation of Q2

Top Level calls the InputInt() function thrice and obtains three integers (viz: a, b, c according to the quadratic

formula). These values are passed to CalDeterminant(...) function. CalDeterminant(...) evaluates the term:

$$b^2 - 4 * a * c$$

known as Determinant. The result thus obtained is passed to IsSqrtPossible(...) function. This function returns 0 (false value) if Determinant is negative and 1 (true value) if the Determinant is positive. Based upon these values, CalRealRoots(...) and CalComplexRoots(...) functions are invoked.

4.3.5 Application of Constructivist Approach

The concept of return values that was introduced in week1 and later on strengthened in week2 is extended to return Boolean data in this week. This Boolean data is stored in a decision variable, which is tested to invoke either CalComplexRoots(...) or CalRealRoots(...).

4.4 Week4

4.4.1 New Topics

For, while, and do-while loops.

4.4.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 7. Top Level calls only one function InputPosInt(), which returns only positive values. InputPosInt() is called ten times in a for loop. Each time, it calls InputInt() in an infinite while loop (not shown). The integer value returned by InputInt() is tested for being positive/negative. Positive values are returned while if the value is negative InputInt() is called again. The other statements of the Top Level are not the functional abstractions and are thus not shown.

Top Level	Pedagogical Goals	F.A built-in	Questions
	Input positive values using Read-Ahead technique (while loop), Differentiate between while and for loops. Drawing shapes with fixed coordinates?	scanf (...), printf (...)	Q1. Input ten positive integers and then display the sum of even and odd integers separately

Figure 7. Description of sample Questions of Week4

4.4.3 Application of Constructivist Approach:

Previously introduced InputInt() function is modified to return only positive values.

4.5.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 8. Top Level inputs a character using built-in function. This character is used to select a particular menu option. Based upon this character, we can choose either to draw a line, a bar, a square, or a

Top Level	Pedagogical Goals	F.A built-in	Questions
<p>Loop: while (infinite times) char ch=getchar ()</p> <p>int1 ← int ← InputPosInt() InputInt()</p> <p>If statement</p> <p>DrawLineORBar(char,int1) If statement DrawRows(int1) else DrawRows(int1)</p> <p>else if statement</p> <p>DrawSqrORRect(char,int1) int2 ← int ← InputPosInt() InputInt()</p> <p>If statement DrawRows(int1) DrawCols(int1) else DrawRows(int1) DrawCols(int2)</p> <p>else statement exit (0) end while</p>	<p>Drawing simple geometrical structures in Text mode using graphical characters. Drawing asterisk shapes using variable coordinates?</p>	<p>getchar(...), scanf(...), gotoxy (...), getch (...), getche (...), clrscr (...), printf(...)</p>	<p>Q1. Input a character. If the character is: 'b': draw a bar 'l': draw a line 'r': draw a rectangle 's': draw a square 'EOF': terminate the program</p>

Figure 8. Description of sample Questions of Week5

rectangle. Next, the size or the length of geometrical object is inputted. These two values are passed to the functions, DrawLineORBar(...) and DrawSqrORRect(...). Using these two functions the above mentioned four objects are drawn. For rectangle and square, both rows and columns are drawn. This is done by calling DrawRows(...) and DrawCols(...) function. In case of a rectangle, row and column size is not same. For this reason, another integer value is inputted in DrawSqrORRect(...) function.

4.5.3 Application of Constructivist Approach

More examples of previously introduced iterative constructs are given in the context of drawing geometrical objects using graphical characters in a if-else-if construct. DrawRows(...) and DrawCols(...) are introduced for implementation of geometrical objects.

4.6 Week6

4.6.1 New Topics

Nested loops, Series, and Sequences.

4.6.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 9. Top Level calls InputandValidateInt(...) twice. This function returns x(int1) and y(int2) values of the coordinates of the centre of flower within the screen boundaries. Once the valid coordinates are obtained, LeftPetal(...), RightPetal(...), TopPetal(...) and BottPetal(...) functions are called which draw the four petals of the flower. The processing which is not shown uses the coordinates of centre of flower to evaluate the coordinates of the tip of each petal. These coordinates are passed as arguments to the above mentioned functions. Top and bottom petals call DrawRows(...) while left and right petals call DrawCols(...) function.

Top Level	Pedagogical Goals	F.A built-in	Questions
<p>int1/ int2 ← InputandValidateInt(int) X 2 int ← InputInt()</p> <p>LeftPetal(int1, int2) DrawCols(int x, int y)</p> <p>RightPetal(int1, int2) DrawCols(int x, int y)</p> <p>TopPetal(int1, int2) DrawRows(int x, int y)</p> <p>BotPetal(int1, int2) DrawRows(int x, int y)</p>	<p>Developing shapes using asterisk series. Drawing Numerical shapes using variable center coordinates?</p>	<p>scanf(...), gotoxy (...), getmaxx(...), getmaxy(...), printf(...)</p>	<p>Q1. Input coordinates of center of flower shown in Figure 12 and draw the entire flower anywhere on the screen.</p>

Figure 9. Description of sample Questions of Week6

4.6.3 Application of Constructivist Approach

Instead of using graphical characters, shapes are drawn using asterisk characters and can lie anywhere on the screen. This involves modifying the input process by encapsulating the previously defined InputInt() function in a new function called InputandValidateInt(...).

4.7 Week7

4.7.1 New Topics

More on Nested Loops, Series, and Sequences.

4.7.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 10. Top Level calls InputandValidateInt(...) twice. This function returns x(int1) and y(int2) values of the coordinates of the centre of star within the screen boundaries. Once the valid coordinates are obtained, top and bottom rows of the star are drawn with specific characters as shown in Figure 10.

Top Level	Pedagogical goals	F.A built-in	Questions
<p>int1/ int2 ← InputandValidateInt() X 2 int ← InputInt()</p> <p>Loop: For(7 times) DrawTwoRows(int1, int2, int3, int4) DrawRows(int1, int2, int3/int4) end For</p>	<p>Generating shapes using Numerical series. Menu-Driven programs?</p>	<p>scanf(...), gotoxy (...), getmaxx(...), getmaxy(...), printf(...), clrscr (...)</p>	<p>Q1. Input coordinates of center of star shown in Figure 13 and draw the entire star anywhere on the screen.</p>

Figure 10. Description of sample Questions of Week7

4.7.3 Application of Constructivist Approach

Instead of using graphical characters, shapes are drawn using numerical characters. DrawRows(...) is modified by passing four arguments, the last value represents the integer value of the character to be printed and int3 represents the row size.

4.8 Week8

4.8.1 New Topics

Multiple selection using switch, pass-by-value, and pass-by address parameter passing.

4.8.2 Top Level Explanation of Q1

Top Level along with other details for the sample questions of this week is shown in Figure 11. Top Level first calls the DisplayMenu() function to display the list of options. Then the user is prompted to choose an option by calling InputandValidateCh(). If the option is “1,” a random number is generated by calling GenerateRandomNum() function. Next the Guess(...) function is called in which the process of guessing the random number is carried out in seven tries. If the option is “2,” GenerateRandomNum() function is called three times, which generates three random numbers which are stored in integer variables, int1, int2, and int3. These numbers are sorted using if statements in the sort(...) function. The resultant arguments are then displayed using the DisplayNums(...) method. If the option is “3,” the program terminates.

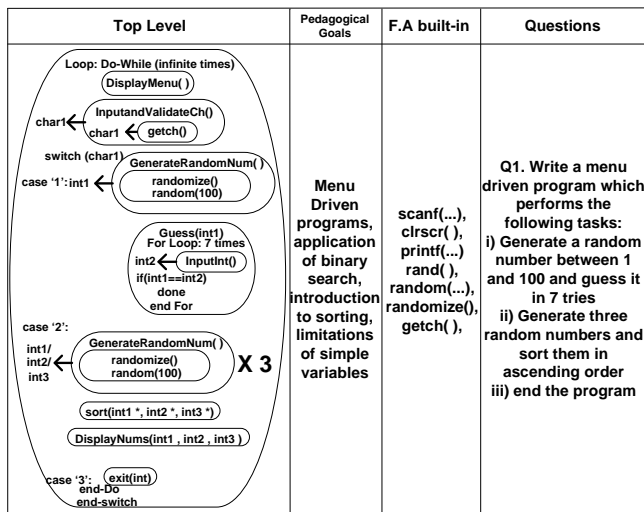


Figure 11. Description of sample Questions of Week8

4.8.3 Application of Constructivist Approach

Data generation process is modified. Instead of using InputInt() function, which is introduced in Week3, random number generation functions are used.

5. Results and Observations

The contents of the program are segmented into Data, Operation, and Programming construct cycles. During the concluding remarks of each exercise, the constraints of these segments are illustrated and extensions are proposed which adhere to the constructivist style. The enhancements are made using functions which are started from the very beginning of the session. This proved to be extremely effective in two ways. Firstly, the relevance to previous knowledge increased students’ interest. Secondly, introduction of function topic at an earlier stage increased the maturity level of students.

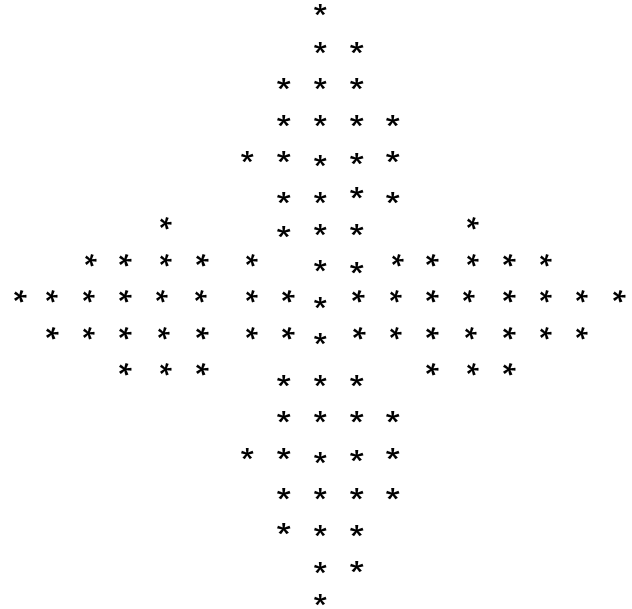


Figure 12. Flower shape for Week6 problem

The following considerations are used to evaluate the results of this study as done by Berglund and others (1996):

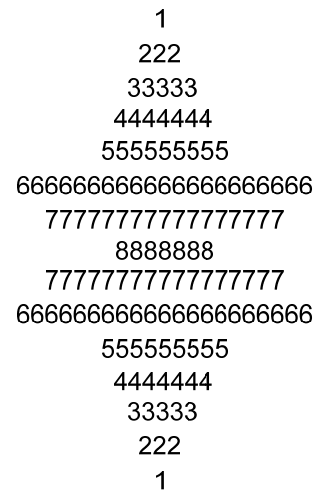


Figure 13. Star shape for Week7 problem

5.1 Teachers’ Time

As far as teachers’ workload matters, teachers do not have to shoulder any extra assessment burden. Additionally, the creation of four or five programming exercise questions weekly is not a hectic job.

5.2 Students’ Time

The good aspect of this course is that it makes the students come prepared in the class due to the weighted active

learning sessions (Carver, 1996). This increased workload for students should be viewed positively.

5.3 M.T.E Results

M.T.E. followed the same 2006 pattern. However, this time results showed considerable increase in both the sections taught as compared to last year. The M.T.E results (maximum marks =40) for the two sections are summarized in Table1.

Both the passing percentage and average class marks have shown significant increase. The passing percentage more than doubled, while the average class marks came close to doubling.

	Fall 2006 Section 'A'	Fall 2006 Section 'B'	Fall 2007 Section 'A'	Fall 2007 Section 'B'
Passing Percentage	29	27	64	61
Average Class Marks	19	18	33	33

Table 1: Midterm exam results

5.4 Pre-Test

No formal pre-test was conducted. During introductory session, it was learned that three students of 2006 had a background in "C" programming language. Out of these only one was able to perform well, while the other two took the course lightly and their performance deteriorated. In 2007, two students claimed to have studied "C" language and their performance was quite satisfactory. During consultation, it was found that they enjoyed the course because they did not have to go through the simple details in the beginning.

5.5 Post-Test

As discussed earlier, in-class exercises were used as Post-Test. It shows an interesting pattern. In 2006, performance was good in the beginning but became pathetic towards the end. On the other side, in 2007, students were slow in the beginning but improved substantially at the end as shown in Figure 14.

5.6 Students' Innovations and Self-Study Habits

Due to the teaching of advanced topics first, students became more self-confident. They started using advance data structures like "files" in their programs. This became evident during labs and student consultations. Some students even created a library of functions and used it in their final class project.

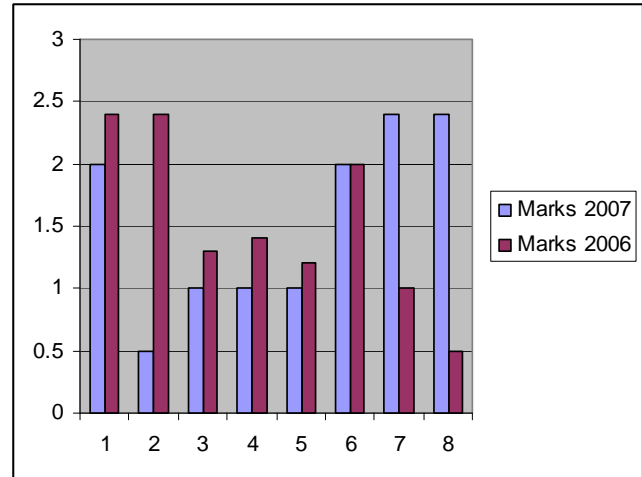


Figure 14. Average class marks for in-class programming activities for the two sections taught in 2006 and 2007

5.7 Student Comments

Some of the students' comments that were collected in fall 2006 are given below:

"There should be more focus on advance topics."

"Initially it seems fun but at the end things become very cumbersome."

A few of the comments collected in fall 2007 are given below:

"Programming exercises related to Mathematical problems were very interesting, carry on."

"It can't be much better."

The latter quotations suggest that students appreciated the techniques applied in this research and it is safe to carry on this technique in other programming courses.

6. Conclusion

In this paper, the focus has been on simpler programming examples, which are normally used in teaching introductory programming courses but demonstrated how complex programming constructs like functions can be used at the startup of their implementation (Z-A teaching style). The study shows the work of first eight weeks. However, the remaining eight weeks can be taught in the same spirit by concentrating on advance topics like sorting, searching, and two-dimensional arrays from the beginning. By using separate sheets and other techniques, an environment of honesty has inculcated among the students, which can be treated as a professional value as quoted in Fuller (2008). The programming language domain is not object oriented programming but these functions can easily be encapsulated

in a class construct. Thus, these examples can be transformed to reflect the object oriented programming style, although it is still a debatable topic to teach object oriented programming from the first semester. Comparatively, results are better than the results of the course, which was taught the previous time. One interesting outcome of the research is the development of a technique for program representation based upon functions termed as FPRT. Future work could extend this technique for advanced level programming courses.

References

- Berglund, A., Daniels, M., Lundqvist, K., & Westlund, E. (1996). Encouraging active participation in programming classes, 7th National Conference on College Teaching and Learning (pp. 1-5), Jacksonville, USA.
- Carbone, A., Hurst, J. Mitchell, I., & Gunstone, D. (2001). Characteristics of programming exercises that lead to poor learning tendencies: Part II. *Proceedings of the Conference on Innovation and Technology in Computer Science Education* (pp. 93-96). Canterbury, UK: ACM Press.
- Carver, C.A., Howard, R. A., & Lane, D. W. (1996). A Methodology for Active, Student-Controlled Learning: Motivating our Weakest Students. *ACM SIGCSE Bulletin*, 28(1). Philadelphia, PA: ACM Press. 195-199.
- Djordjevic, M. (2007). Teaching introductory programming course with progressive graphics examples. *Proceedings of the 2007 Computer Science and IT Education Conference* (pp. 177-185). Mauritius, Publisher@InformingScience.org. Retrieved October 14, 2008, from <http://csited.org/2007/63DjorCSITEd.pdf>
- Fuller, U., & Keim, B. (2008). Assessing students' practice of professional values, *Proceedings of the 13th annual Conference on Innovation and Technology in Computer Science Education* (pp. 88-92). Madrid, Spain: ACM Press.
- Hadjerrouit, S. A. (1999). Constructivist approach to object-oriented design and programming. *Proceedings of the Conference on Innovation and Technology in Computer Science Education* (pp. 171-174). Cracow, Poland: ACM Press.
- Whittington, K. J. (2003). Implementation of alternative spacing in an introductory programming sequence. *Proceedings of the 4th Conference on Information Technology Curriculum* (pp. 47-53). West Lafayette, Indiana: ACM Press.
- Whittington, K. J. (2005). Progressive programming assignments. *The Journal of Issues in Informing Science and Information Technology*, 2, 451-459.

Author Information

Zulfiqar Ali Khan, MS
 Assistant Professor
 Sir Syed University
 Computer Engineering Department
 Karachi, Sindh, Pakistan
 111-994-994
zulfi6000@yahoo.com

Zulfi enjoys discussing teaching and learning in higher education courses, especially concerning the application of software technologies. He teaches most of the undergraduate courses related to programming.