TESTING PROGRAMMING APTITUDE IN INTRODUCTORY PROGRAMMING COURSES *

James Harris
Department of Computer Science
Georgia Southern University
jkharris@georgiasouthern.edu

ABSTRACT

This paper is a work in progress on establishing a programming aptitude test for CS1 courses. It involves using a simple language, Programming Aptitude Language (PAL), along with a specialized tutorial/test to establish a measure for programming aptitude.

INTRODUCTION

According to the University of Kent [17], "Programming aptitude tests for computing jobs broadly fall into three groups:

1. A standard battery of tests assessing competencies such as numerical reasoning, logical reasoning and non-verbal reasoning which are required in technical computing jobs.

2. A hybrid test comprising of elements involving logical reasoning, numerical problem solving, pattern recognition, ability to follow complex procedures and attention to detail.

3. A programming simulation involving pseudocode"

The first two groups assume no knowledge of programming. Early tests, such as IBM's Programmer Aptitude Test [10] and Wolfe's series of programming aptitude tests [19,20,21] fell into the first two groups. Since then, there have been numerous tests developed using psuedocodes [1], games [12], and various other methodologies [18,7,9,2,5,8,11,1]. More recently, Dehnali and Bornat[3] developed a programming aptitude test that they claimed " picks out, with 100% accuracy, those people who have a chance of learning to program and rejects, with 100% accuracy, those who have no chance" [4]. Validation of programming aptitude tests usually involves either

programmer productivity, as in industry, or succeeding in a course or program, as in academia. This paper is concerned with the latter, more specifically, success in a CS1 course.

JUSTIFICATION

There are many predictors of success in a CS1 course [14,14,6]. Some of the more commonly used predictors are SAT scores, grades in math courses, and overall grades. Very often it is difficult to gather and process this information early in the semester to determine strategy when teaching an introductory programming course. It is also difficult to test programming aptitude early in the semester in an introductory course because it cannot be assumed that students have any prerequisite programming knowledge. That was the motivation for creating a very limited and simple programming language with a simple syntax that could be quickly and easily learned through a short tutorial, giving coding examples to show how the language can be used, and then asking questions to test the ability to learn and use the language. This is exactly what is expected of students in an introductory programming course, albeit in this case, it is on a much smaller scale. In addition, this language is syntactically and structurally different enough from popular programming languages, such as Java, and simple enough that students with previous programming backgrounds do not have much of an advantage over those who do not.

DESIGN

Programming Aptitude Language (PAL) was designed so that students do not need any programming background to write programs. It is based on a very simple, assembly-like language with as simple a syntax as possible. The Programming Aptitude Test and Tutorial (PATT) were designed to measure aptitude in a procedural paradigm and were structured so that questions become more difficult as students proceed. It was designed so that reading and interpreting problems gives clues on how to solve future problems. This is very similar to how students learn a programming language, such as Java, in which learning occurs through understanding code samples. The test and tutorial are shown below.

PROGRAMMING APTITUDE LANGUAGE (PAL) AND PROGRAMMING APTITUDE TEST (PATT)

Please read the following tutorial about the Programming Aptitude Language (PAL) and try to answer as many questions as possible
In PAL, there are 100 memory cells labeled [0], [1]…[99], each of which can store a single number.
There are only six PAL instructions:

store instruction:
store <memory cell or number>,<memory cell> - stores the contents of the first memory cell (or number) into the second memory cell. The previous contents of the second memory cell are destroyed. Examples:
store 27,[2] - stores the number 27 into memory cell [2]. Whatever was in memory cell [2] before this instruction is destroyed.

store [3],[6] - stores the contents of memory cell [3] into memory cell [6]. Whatever was in memory cell [6] before this instruction was executed is destroyed.

add instruction:
add <memory cell>,<memory cell>,<memory cell> - adds the contents of the first two memory cells and stores the result in the third memory cell. The previous contents of the third memory cell are destroyed.
Example:
add [2],[45],[31] - adds the contents of memory cell [2] to the contents of memory cell [45] and stores the result in memory cell [31]
read instruction:
read <memory cell> - reads a value from the keyboard into the memory cell (the user types in a value). The previous contents of the memory cell are destroyed.
Example:
read [5] - reads in a value from the keyboard and stores it into memory cell [5]
print instruction:
print <memory cell> - prints the contents of the memory cell on its own line
Example:
print [56] - prints the contents of memory cell [56]

end instruction:
end - stops execution of the program. This instruction can appear anywhere in the program, but almost always appears as the last instruction.
The sixth instruction is explained later. Each line in a PAL program starts with a line number, followed by a colon, followed by an instruction. The first instruction is line number one and the following instructions are numbered consecutively. The statements are executed in order, in other words, in this order by line number 1, 2, 3, etc.
Here is a sample PAL program:
1:store 8,[0]
2:store 9,[1]
3:print [1]
4:print[0]
5:end
The first line stores the value 8 into memory cell [0]. The second line stores the value 9 into memory cell [1]. The third line prints out the value stored in memory cell [1] (which is 9). The fourth line prints out the value in memory cell [0] (which is 8) and the fifth line ends the program.

Problem 1: What does this PAL program print out?
1:store 5,[0]
2:store 6,[1]
3:add [0],[1],[2]
4:print [2]
5:end
Answer:

Problem 2: In the space below, write a PAL program that adds together the three numbers 15, 35 and -5 and prints out the result.

Problem 3: What does this PAL program print out?

```
1:store 5,[0]
2:store 6,[1]
3:store [0],[2]
4:store [1],[0]
5:store [2],[1]
6:print [0]
7:print [1]
8:end
```

Answer: _____

There is also the following instruction:

jumplarger instruction:

jumplarger <memory cell>,<memory cell>,<line number> - if the contents of the first memory cell is larger than the contents of the second memory cell then the next instruction executed will be on line number <line number>, otherwise the instruction on the line number after the jumplarger instruction is executed next.

Example:

8:jumplarger [2],[5],2 - if the content of memory cell [2] is larger than the contents of memory cell [5], then jump to line number 2 and execute that instruction next, otherwise execute the instruction on line number 9 next.

Please notice that this instruction can change the order the instructions are executed. For example:

```
1:store 5,[1]
2:store 6,[2]
3:jumplarger [2],[1],5
4:print [0]
5:print [1]
6:end
```

Since the contents of memory cell [2] is 6, which is larger than the contents of memory cell [1], which is 5, execution will jump to line number 5 and line number 4 will be skipped. In other words, only the contents of memory cell [1] will print.

Problem 4: What does this PAL program print out?

```
1:store 4,[0]
2:store 1,[1]
3:store 6,[2]
4:add [1],[0],[0]
5:jumplarger [2],[0],4
6:print [0]
7:end
```

Answer: _____

Problem 5: In the space below, write a PAL program that reads in two numbers and prints out the larger of the two numbers (assume the numbers are different)

Problem 6: In the space below, write a PAL program reverses the contents of the first two memory cells (cells [0] and [1]). For example, if cell [0] contains the value 123 and cell [1] contains the value 225 then after your program runs cell [0] should contain the value 225 and cell [1] should contain the value 123.

Problem 7: In the space below, write a PAL program that reads in three numbers and prints out the largest of the three numbers (assume all three numbers are different).

Problem 8: In the space below, write a PAL program that reads in three numbers and prints out the numbers from smallest to largest

VALIDATION

PATT was administered to a CS1 class of 23 students on the first class day of the semester. One point was given for a correct solution, 2/3 of a point for a solution with correct logic, but incorrect syntax , 1/3 of a point for some logic being correct and 0 points for a completely incorrect solution. A midterm exam was administered to the same group and the PATT results were correlated with the results of the midterm exam. The midterm exam consisted of 10 short programming problems in C# programmed over a two hour period using Visual Studio. The midterm exam was embedded into a Visual Studio project that was provided to students at the start of the exam. Each solution was graded using test data and was assigned a grade of 1 if correct and 0 if incorrect. Test data was provided for students to test their solutions before turning in their VS project folder. A sample midterm exam question is shown below:

```
//Do NOT remove or comment the line below!
Console.WriteLine("******************** Problem 2 output ***************************");

//Only uncomment these two lines when testing your method or turning in a correct solution
//Console.WriteLine(partialHarmonic(2,5));  // 1/2 + 1/3 + 1/4 + 1/5 = 1.283333333333333
//Console.WriteLine(partialHarmonic(10,10)); // 1/10 = 0.1

/****************************************** Problem 2
* Write a method named partialHarmonic that takes as arguments an integer M >= 1 and and N >= M and
returns the sum 1/M + 1/(M+1) + 1/(M+2) ... + 1/N
* For example, if M=2 and N=5 the method call partialHarmonic(2,5) would return the sum 1/2 + 1/3 + 1/4 +
1/5 */
```

For a complete set of problems on the midterm exam, see http://jharris.mat.georgiasouthern.edu/midtermproblems. The results of PATT and the midterm exam are shown in table 1 below.

	# of Problems	N (students)	Range of score	Avg	SDev
PATT	8	23	1.67 - 7.67	4.74	1.96
Midterm Exam	10	23	1-10	5.63	3.24

Table 1: Results of PATT and Midterm Exam

PATT scores were correlated with the results of the midterm exam scores, student GPA, and student math SAT scores. The results are shown in table 2.

	N	Avg	SDev	*Correlation w/ PATT*
Math SAT	16	617.50	75.41	0.541
GPA	23	3.38	0.57	0.049
Midterm	23	5.63	3.24	0.881

Table 2: Correlation of PATT with Math SAT, GPA, and Midterm scores

It is interesting to note that there is almost no correlation between GPA and PATT scores. There was a fairly good correlation between PATT and midterm exam scores (0.881). In order to determine which subsets of questions might best be used to determine proficiency, each individual question score on PATT was correlated with the midterm score. Correlations between the scores on individual questions on PATT and the midterm scores, correlations between scores on PATT questions and the midterm exam scores ranked by correlation (most correlated to least correlated) , and correlations between averages of the top 2,3,4,5,6,7, and 8 questions (in terms of correlation) on PATT and the midterm scores are shown in table 3 below.

Question	Avg	SDev	*Correlation*		Question	*Correlation*		Average of scores	*Correlation*

1	0.96	0.21	0.230		7	0.898		Average All	0.881
2	0.81	0.24	0.560		6	0.868		Average 2 3 4 5 6 7 8	0.974
3	0.61	0.33	0.728		3	0.728		Average 2 3 5 6 7 8	0.970
4	0.75	0.29	0.444		8	0.718		Average 3 5 6 7 8	0.970
5	0.46	0.36	0.653		5	0.653		Average 3 6 7 8	0.967
6	0.56	0.48	0.868		2	0.560		Average 3 6 7	0.935
7	0.45	0.47	0.898		4	0.444		Average 6 7	0.872
8	0.26	0.35	0.718		1	0.230			

Table 3: Correlation of PATT questions with the Midterm Exam Scores

The highest correlation occurred between averaging the scores on all questions, except question 1, which was the easiest question on PATT (Avg. 0.96) and the midterm scores. Question one was a very basic question to determine if there was any understanding at all of PAL syntax. Every student, except one, answered question one correctly. The correlation of the average of scores for questions two through eight with the midterm scores was 0.974, which was the highest correlation for subsets of questions.

CONCLUSIONS AND FUTURE WORK

PATT has so far been shown to provide an accurate measure of programming proficiency in a CS1 course. It will be tested again in the future to see how it performs with different CS1 classes. If it performs as well as it did for this CS1 class, it will be used to identify students at risk in the beginning of the semester and provide the necessary help early so that the at-risk students have a chance of succeeding. It is our hope to eventually have an online version of PATT that will allow provide randomly assigned problems with automated grading starting a beginners level and working up to an advanced level within a given time period. This way we can determine exactly where each participant stands within a given category. It is envisioned to have a PAL IDE where the PATT can be given and PAL programs can be assigned, created, run and then submitted as solutions.

REFERENCES

[1] Aptitude for Programming Practice
http://www.cse.dmu.ac.uk/placement/AptitudeTest-Programming.pdf University of Kent Computer Programming Aptitude Test
http://www.kent.ac.uk/careers/tests/computer-test.htm

[2] Berger Aptitude for Programming Test (B-APT) http://www.psychometrics-uk.com/page26.html.

[3] Bornat, R., Dehnadi, S., Mental models, consistency and programming aptitude, January 2008, ACE '08: Proceedings of the tenth conference on Australasian computing education, Volume 78.

[4] Caspersen, M., Larsen, K., Bennedsen, J., Mental models and programming aptitude, June 2007, ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education.

[5] ExpertRating online Programming Aptitude Test http://www.expertrating.com/certifications/Programming-Aptitude-Test.asp.

[6] Hostetler, T., Predicting student success in an introductory programming course, September 1983, SIGCSE Bulletin, Volume 15 Issue 3.

[7] Imperia Program Aptitude Test http://www.niitimperia.com/program-aptitude-test.

[8] International Academy of Computer Training Programming Aptitude Test http://www.iact.ie/programming-aptitude-test/.

[9] The International Programming Aptitude Test http://www.winrowtesting.com/index.htm.

[10] IBM Programmer Aptitude Test http://ed-thelen.org/comp-hist/IBM-ProgApti-120-6762-2.html.

[11] The Language-Free Programmer/Analyst Aptitude Test http://www.linkedin.com/company/apr-testing-services/the-language-free-programmer-analyst-aptitude-test-524890/product.

[12] Lorenzen, T., Chang, HL., MasterMind©: a predictor of computer programming aptitude, June 2006, SIGCSE Bulletin , Volume 38 Issue.

[13] Scanlan, D., The mental abilities associated with programming aptitude, February 1988, CSC '88: Proceedings of the 1988 ACM sixteenth annual conference on Computer science.

[14] Simon, S., Cutts, Q., Fincher, S. The ability to articulate strategy as a predictor of programming skill, January 2006, ACE '06: Proceedings of the 8th Australasian Conference on Computing Education, Volume 52.

[15] Simon, S., Fincher, S., Robins, A., Predictors of success in a first programming course, January 2006, ACE '06: Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 , Volume 52.

[16] Testa, C., A new approach to programmer aptitude testing, June 1973, SIGCPR '73: Proceedings of the eleventh annual SIGCPS computer personnel research conference 98.

[17] University of Kent Computer Programming Aptitude Test http://www.kent.ac.uk/careers/tests/computer-test.htm

[18] Walden Programming Aptitude Test (Wolfe-Spence Programming Aptitude Tests) http://www.waldentesting.com/infotechtests/proapt.htm

[19] Wolfe, J., Perspectives on testing for programming aptitude, January 1971, ACM '71: Proceedings of the 1971 26th annual conference.Wolfe, J., An interim validation report on the Wolfe: programming aptitude test, April 1977, SIGCPR Computer Personnel, Volume 6 Issue 1-2.

[20] Wolfe, J., Wolfe programming aptitude test (school edition), June 1971, SIGCPR '71: Proceedings of the ninth annual SIGCPR conference.