A program design tool to help novices learn programming

Stuart Garner
School of Management
Edith Cowan University

The learning of software development is difficult for many students. Often students start writing programming code as soon as they receive a problem statement without paying sufficient attention to program design. One of the most common ways to design the logic of a program is to use pseudocode, and yet many students do not like to do this. Reasons for this include: it is another language to learn; they do not think that they are actually programming; they cannot test their designs as the designs are not executable; there is not a rigid syntax and so students are unsure whether their pseudocode meets an instructor's expectations. This paper concerns the development of a simple tool that helps students create pseudocode. The tool has been used and evaluated in an introductory programming unit of study. The results suggest that the tool was easy for students to learn and that it helped support their learning.

Keywords: Programming, design, pseudocode, tools

Introduction

This paper concerns the development, use and evaluation of a tool that helps novice programmers generate pseudocode designs for simple programming problems. It discusses: the difficulties of learning to program; what is meant by pseudocode; the development of the tool; and the evaluation of the tool with students.

Difficulty of learning to program

Students struggle with programming, and programming has continued to be a major factor contributing to the attrition of first year students from the computing courses (Miliszewska & Tan, 2007). Jenkins (2002) suggests that the learning of programming is a perennial problem. Students struggle as they try to master the subject and it is not uncommon for a student's first experience of programming to be so negative and stressful that it leads to academic failure or withdrawal.

Programming is a complex process involving many steps (Winslow, 1996) including:

- Studying a given problem statement / set of requirements and producing an algorithm, often in pseudocode, to solve that problem;
- Translating the algorithm into the programming code of a certain programming language; and
- Testing and amending the program until it meets the original set of requirements.

Frequently students combine steps one and two above and attempt to produce algorithms in the programming language that is being utilised in their course of study rather than in a design language such as pseudocode. Many instructors do not encourage the use of pseudocode with students and the majority of introductory programming texts include few, if any, references to pseudocode.

What is pseudocode?

Pseudocode is

> A detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language. Pseudocode is sometimes used as a detailed step in the process of developing a program. It allows designers or lead programmers to express the design in great detail and provides programmers a detailed template for the next step of writing code in a specific programming language (Techtarget, 2005).

Algorithms to solve a particular problem may well differ in their syntax dependent upon the designer. Robertson (2003) points out that:

> There is no standard pseudocode at present. Authors seem to adopt their own special techniques and sets of rules, which often resemble a particular programming language.

The fact that there is no standard is one of the reasons that many students bypass using it in their program designs and move directly to coding their solutions. Another reason for students disliking pseudocode is that the code is not executable and direct feedback of a design cannot therefore be obtained.

However research suggests that the use of pseudocode in the design of programs helps students in their learning of software development. Ramadhan (2000) synthesised certain empirical results and concluded:

- The underlying programming constructs can be learned independently of the surface structure or syntax of the programming language; and
- When users are introduced to programming using a pseudocode language they perform more accurately during problem solving and their programming knowledge is more transferable to other programming languages, and the languages that involve simple and few programming concepts to be learned reduce the mental load on users and help them in the process of program understanding and debugging.

Development of a program design tool

A technology supported tool to help students create pseudocode for the design of programs was developed by the author. The main requirements of the tool were: ease of use; to include a library of "standard" constructs; to include a facility to change the "standard" constructs in order to utilise a different syntax; and a to include a facility to gradually add pseudocode constructs to the tool, thereby allowing for a reduced cognitive load on students in the earlier stages of their learning.

The inspiration for the pseudocode tool came from a tool that is being used to help students learn scripting languages (Gibbs, 2002). The tool developed by Gibbs makes use of a freeware text editor named NoteTab (Notetab, 2007). NoteTab is much more than a text editor as it includes: a "clipbook" feature that lets a user create and organise textual clips; and macros that can range from text replacement to complete mini-applications that use a simple built-in scripting language.

Such a "clipbook" library was created by the author for use with NoteTab. The library was specifically designed to support the production of pseudocode for introductory programming in an event driven environment, eventual implementation being in Visual BASIC. The interface of NoteTab with the pseudocode library loaded is shown in Figure 1.

The left-hand window contains the pseudocode library and the right-hand window contains an example of the pseudocode that can be "generated". When students use the system, they double click on an item in the library and a macro is executed. The macros prompt the user for relevant information, generates text and then places that text in the right-hand window.

The design tool supports the standard statements that are necessary to create algorithms that will eventually be implemented in a programming language including: input and output statements; assignment statements; the selection and control structures; procedures and functions; etc.

The pseudocode shown in the figure uses a syntax that has been created by the author. The aim of the particular syntax utilised is to help students gain a better understanding of what particular statements in an algorithm "do". An example of the a pseudocode statement is

> Place the data from the text property of txtPrice into price after converting it to a number

which equates to the Visual BASIC statement

```
Price = CDbl(txtPrice.Text)
```

Use of the program design tool by students

The tool has been used over a number of semesters in an introductory programming unit in Visual BASIC at an Australian university and student feedback was obtained via a questionnaire. A majority of students

indicated that: the pseudocode generator was straightforward or easy to use; they usually created the pseudocode before the Visual BASIC code; the creation of pseudocode usually helped their understanding because the syntax of the statements made explicit the underlying mechanism of the underlying "machine"; and that they preferred to use the pseudocode generator before creating programming code in Visual BASIC.

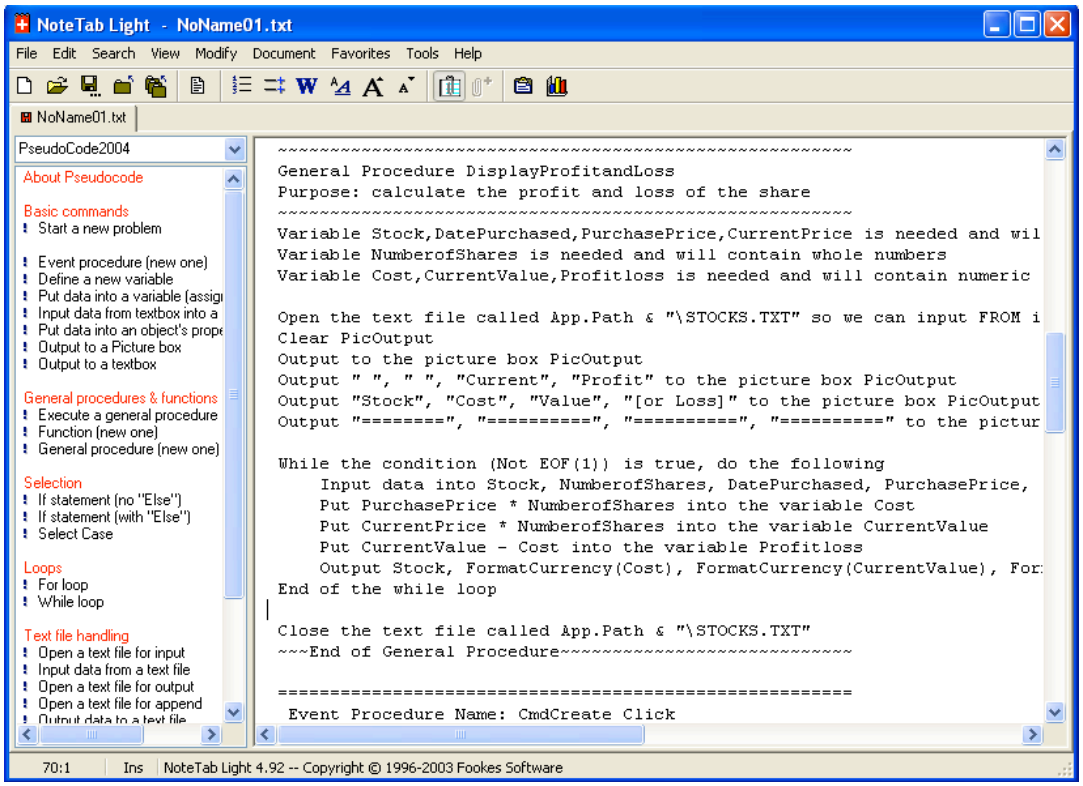

Figure 1: Program design tool interface

The results are pleasing as it is well documented that generally students, who do not use such a tool, do not create pseudocode before they create programming code. Also, the pseudocode generator seems to have met the objective of helping students in their understanding of how the underlying notional machine works.

Students were also asked to comment on the use of the pseudocode generator. The positive comments included:

- It helped me go through all the things needed in the question
- The pseudocode in general was very straightforward and easy to follow. The indentation of certain statements was useful as it reminded me to indent the statement when writing the actual Visual BASIC code.
- It helps you to think about the sequence of the program statements;
- It forces you to organise your thoughts in a clear logical manner. Without this I find I tend to go back and forth between deciding what I do and do not want in my code. The creation of the pseudocode gave me the necessary preparation time before commencing the programming task.
- Using the pseudocode helps in understanding what is needed to do in the VB code.
- It makes me think more carefully about the question and the variables I need for each question.
- Well... lets me organise a bit in computer instead of the brain. It reduced the typing work required.

The comments strongly support the use of the pseudocode generator as a tool that helps in the design process of algorithms that can then be implemented in a programming language. It seems to have helped students think about the problems to be solved and organise their solutions and designs in a coherent manner. There was one negative comment:

- It is useless!! Because, if we get the pseudocode wrong, we will not know if the program is correct. Using VB directly, we can monitor our mistake.

This comment concerns the fact that pseudocode is not executable and clearly this student believed that the cycle of coding and testing in a program development environment was more useful in his or her learning. This reflects the fact that some students always prefer to skip the careful thought processes required in program design and use trial and error techniques in their algorithm design.

Conclusions

This paper has discussed the design and use of a tool to support the generation of pseudocode for novice programmers. It has been used and evaluated with students in an introductory programming unit within an Information Systems major at an Australian university. The tool has been well received by students who believe that it is very useful in their learning and that it helps scaffold them in the program design process.

The results of the evaluation suggest that possible changes to the tool and its use might include:

- The actual programming language statements could be generated in addition to the pseudocode statements.
- A help file that includes information on all of the pseudocode statements could be included.
- A facility could be made available that allows students to easily change the wording, or syntax, of statements generated by the macro library. Such a facility should not require students to have to amend scripts that form the basis of the macro library.

References

Gibbs, D. C. (2002). *An interactive introductory programming environment using a scripting language*. Paper presented at the Ed-Media 2002, Denver, Colorado.

Jenkins, T. (2002). *On the cruelty of really teaching programming*. Paper presented at the 2nd LTSN-ICS one day conference on the teaching of programming, University of Wolverhampton, UK.

Miliszewska & Tan (2007). Befriending Computer Programming: A Proposed Approach to Teaching Introductory Programming. *Issues in Informing Science and Information Technology*, 4, 278-289.

NoteTab (2007). NoteTab. Retrieved April 4, 2007: http://www.notetab.com.

Ramadhan, H. A. (2000). Programming by discovery. *Journal of Computer Assisted Learning*, 16, 83-93.

Robertson, L. (2003). Simple program design. Thomson.

TechTarget (2005). TechTarget. Retrieved April 4, 2007: http://whatis.techtarget.com/definition/0,,sid9_gci213457,00.html

Winslow, L. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin*, 28(3).

Stuart Garner, s.garner@ecu.edu.au

Please cite as: Garner, S. (2007). A program design tool to help novices learn programming. In *ICT: Providing choices for learners and learning. Proceedings ascilite Singapore 2007*. . http://www.ascilite.org.au/conferences/singapore07/procs/garner.pdf