Augmenting Compiler Error Reporting in the Karel++ Microworld

Dr Chris Burrell
Waikato Institute of Technology
Chris.Burrell@Wintec.ac.nz

Dr Matt Melchert
Waikato Institute of Technology
Matt.Melchert@Wintec.ac.nz

Abstract

This paper illustrates the problems involved with novices learning to program and the role of the compiler in identifying and reporting on errors. Errors associated with the structure and syntax of the program are easily identified and can normally be reported on by the compiler in a form that may understandable to programmers with a good background knowledge. Once these problems have been identified another class of problem may emerge. That of simple user error in mistyping, or misremembering the names of programming elements that have been declared and defined earlier in the code. This paper describes an application of a cross correlation technique that has been used in transmitting data through space by NASA to looking for best-fit word matches in the symbol table produced as the program is compiled. A match with a pre-declared name and one with three typographical errors is shown.

Keywords: Computing education, microworld, compiler help, correlation techniques

1 Introduction

Educators concerned with teaching programming know that it is one of the hardest disciplines to learn, that each student is different and that most of the tools available are designed for experts rather than novices. Novice programmers make many mistakes whilst learning the language and the environment within which they work whilst developing the ability to problem solve in that environment, McGill and Volet (1997).

The practice of choosing suitable names when writing code and using techniques of capitalisation whether the language is case sensitive or not all need to be learned. When the errors associated with opening and closing brackets and language structure are accounted for many of the remaining errors that are made are, for one reason or another, typographical.

This quality assured paper appeared at the *20th Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2007), Nelson, New Zealand. Samuel Mann and Noel Bridgeman (Eds). Reproduction for academic, not-for profit purposes permitted provided this text is included. www.naccq.ac.nz

Typically a compiler does not does not check for spelling or similar forms of error. This research examines, adapts and implements a data communications error correcting technique in an attempt to provide more informative feedback to a novice programmer using the Karel++ microworld environment developed at Wintec.

1.1 Methodology

Anyone who has attempted to write a compiler knows how difficult it is to write meaningful help that is presented when an error is encountered. Anyone who has used a compiler knows how difficult it is to interpret the help offered in the context it is presented. A lot of background knowledge is often required to make effective use of the information. This research looks at the ability to find and present best matches words mistyped into the source code of a program. A technique is adopted, adapted and implemented with encouraging results.

2 The Karel++ Microworld Environment

The Karel++ microworld (Figure 1)enables programmers to write, compile and run object oriented code in a similar way to using a commercial Interactive development environment(IDE)(Burrell, 2001, 2002, 2003, 2004)

The microworld environment as implemented at Wintec works as a virtual machine that executes tokenised code constructed in an extended symbol table. The language is described in Backus–Naur form (BNF) and follows a declare before use structure. The normal error reporting procedures from the recursive descent compiler are augmented by a cross correlation pattern matching technique to help identify best fit mistyped words.

3 Novices Learning to program

Novice programmers learn a language and its rules at the same time as learning to solve problems and become conversant with the working environment. The student has to come to an understanding of the capabilities of the language and more immediately has to learn, and use correctly, naming conventions for objects, methods, attributes and variables.

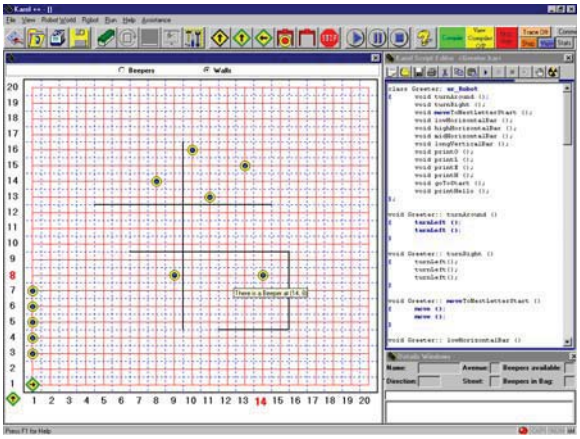

Figure 1The Karel++ IDE used at Wintec

3.1 The skills and abilities required

Two fundamental types of generally accepted knowledge are:

- Declarative: The ability to specify objects and events by the properties that characterise them.
- Procedural: The ability to identify, or perform, sequences of actions that achieve a desired result.

Knowledge about programming can be classified using these headings but additionally software developers also need to have effective command of three separate but interrelated types of programming knowledge. (Bayman & Mayer, 1988)

These are

- Syntactic: Knowledge of facts that are specific to a particular programming language, and the rules for their use.
- Conceptual: Understanding of computer programming constructs and principles.
- Strategic: Programming specific versions of general problem solving skills.

McGill and Volet (1997) produced a framework, that combines these concepts.

To novice programmers, the obstacles to be overcome when presented with their first problems can seem enormous. Almost everything to be learned in terms of the problem and the means of its solution are abstract. Learning to think and work in any recognised paradigm let alone to produce working program implementations is found to be very demanding by many, and daunting by some (personal observation).

With little relevant formal or informal knowledge about any sort of programming, novices are expected to form an understanding of the problem and then use the available tools and recently acquired techniques to develop and implement cohesive, structured and successful solutions. By progressing through a sequence of supportive and exploratory exercises the student is expected to construct their knowledge. The cognitive load is very high.

The process followed is essentially, edit the code using an editor, compile the code and debug the code until there are no more syntax errors and when compilation is finally successful, run the code the code to see if it does what it was supposed to.

3.2 Inattention to detail

Overlooked in the formal knowledge structure and one of the problems faced by many using computers and not only programmers are simple errors through inattention to detail.

One of the problems faced by novice programmers when learning a language and writing code is simple errors due to misspelling, capitalization, or typing in a name that they think has been used before but spelling part of it differently. Often the error checking and "help" capabilities of a compiler do not or cannot cope with this sort of error well.

In a commercial IDE simple spelling, typographical or capitalization errors often generate unhelpful error messages. This can be particularly confusing for a novice who may have insufficient background knowledge to identify the real cause of the trouble.

4 Compiler Operations

The compiler is designed to obey the rules and expectations of the language and apply them with the aim of producing runnable code. Any departure from the structure and rules embedded in the language design is detected and reported with an error message.

A computer language has a list of known reserved words. A programmer uses these and also generates their set of identifiers for such purposes as class and method names and the names of objects, attributes and other variables. As part of the compilation process a symbol table is generated that lists all reserved words and declarations.

The Karel++ language is similar to many in that it has a declare-before-use approach. Names become progressively available and are added to a symbol table as the compiler progresses through the code. When the compilation process finds a piece of text that it cannot understand or was not expecting, an error message is produced.

Errors detected at compile time are often the result of not finding a symbol, character or sequence of letters that exactly matches those expected. Problems with the structure of the program are dealt with by the nature of the compiler and reported on. There could be more simple explanations for the error, such as capitalisation and case sensitivity which the student needs to understand and master. The problems could also result from making typographical errors, or simply guessing the name of something that has been previously declared.

All the reserved words, pre-declared by the language, and are effectively in a symbol table but are not normally available to the user. The declarations made by the programmer are stored in a symbol table intended for inspection if needed. When there is no match to the text stored in the symbol table, a standard error message is generated. How close a match between the text-in-error is to anything that has been used before, is not considered.

The question could then become one of "How near a match is a word in the symbol table to the word just found" ?

4.1 Providing help from a compiler

A compiler processes statements written in a programming language with the intention of producing code that can be run by a machine. A recursive descent design was used for this implementation of Karel++. This type of implementation matches the definition of the language and naturally produces code reflecting the structure of the grammar it finds. When an error is encountered, there is an inherent ability to "back out" and produce suitable error messages that indicate where and why the structure of a program has failed. This approach does not cope with spelling and typographical errors. The technique described here is an attempt to better inform the novice about errors that may occur from mistyping.

Figure 2 below, shows only the first part of a typical piece of code developed in the microworld environment. At the end of this code snippet example is one word with several wrong characters. A typical compiler error could be expected to be "error XXX undeclared symbol encountered. A visual inspection shows that there is obviously some form of typographic or spelling error but the compiler is obviously not programmed to look for or cope with it.

5 Spell checking

In word processors spelling checkers are invoked when words are encountered that are not in the dictionary. The exact nature of the spell checking method is usually proprietary and not known, but experience has shown that a mistake in the middle of a word is enough to detract from the list of words offered.

Where it may be desirable to have a context sensitive spelling checker research has focussed on machine learning techniques to help "Winnow" (Golding & Roth, 1999) out correct choices from text.

Programming in the Karel++ environment does not need the sophisticated techniques used in other applications and can be approached with a more simple solution.

The problem could be likened to a signal or message being corrupted by noise. A more supportive environment can find a "best match" among the contents of the symbol table. Options can be presented when the programmer is pointed to the place in the original text file where the error was discovered.

```
class Greeter: ur_Robot
 {   void turnAround ();
     void turnRight ();
     void moveToNextLetterStart ();
     void lowHorizontalBar ();
     void highHorizontalBar ();
     void midHorizontalBar ();
     void longVerticalBar ();
     void printO ();
     void printL ();
     void printE ();
     void printH ();
     void goToStart ();
     void printHello ();
  };

  void Greeter:: yurnAtpund ()
```

Figure 2 Typical code segment for Karel++,

6 Spell checking

In word processors spelling checkers are invoked when words are encountered that are not in the dictionary. The exact nature of the spell checking method is usually proprietary and not known, but experience has shown that a mistake in the middle of a word is enough to detract from the list of words offered.

Where it may be desirable to have a context sensitive spelling checker research has focussed on machine learning techniques to help "Winnow" (Golding & Roth, 1999) out correct choices from text.

Programming in the Karel++ environment does not need the sophisticated techniques used in other applications and can be approached with a more simple solution.

The problem could be likened to a signal or message being corrupted by noise. A more supportive environment can find a "best match" among the contents of the symbol table. Options can be presented when the programmer is pointed to the place in the original text file where the error was discovered.

7 A lesson from data transmission and signal processing

When the transmission of information through a medium results in a poor signal to noise ratio it becomes difficult to determine the content of the message. There are several coding and communication techniques which can be used to try and recover the

original message. One method, that is useful when several messages are known to be possible but the correct one cannot be easily determined is to use the technique of cross correlation.

This technique looks for a similarity between the incoming message to a pre-stored expected messages. If the two are exactly the same, then the highest value of correlation or goodness-of-fit is achieved. Any differences between the received and stored signals generate much lower values.

The technique was first encountered by the author as part of the NASA Mariner 9 picture transmission system(*Instrumentation Units 11,12,13: Noise in Instrumentation Systems*, 1974). All message content was known but the transmission was likely to be corrupted by significant noise. Identifying which message (pixel value) was sent was the problem.

Information is recovered by "sliding" the digitized received signal past each potential message in a number of discrete steps. At each step, each pair of signal values is multiplied and all are summed to give a value of goodness (correlation). The highest value generated once all of the processing is complete identifies the best-fit correlation between the signal and the potential messages. By substituting the best match data for the corrupted sequence the original data can be recreated.

This correlation method is illustrated with a simple diagram Figure 3 A received sequence is compared with each of the possible error free chain codes. The best match has the highest correlation coefficient..

This technique is easily adapted to comparing sequences of characters looking for "best match" words.

8 Correlation with sequences of characters

In programming the "reference" sources only consist of the words defined in the language and the words defined by the programmer. In analyzing the program the compiler builds this reference list in the symbol table. The compiler can then check the contents of a symbol table to see if the string of characters just found are already in the symbol table, can be added to the symbol table or are erroneous.

If the word is not found in the symbol table and cannot be added, the correlation technique is invoked to find the best match. A Boolean operation compares each pair of characters as they are "slid" past each other, and routine calculates a sum for that position. The highest value obtained for the sequence is stored along with the word. The set of operations is repeated for all words in the symbol table and the resulting highest number for all of the words identifies the best match. A highest value can be obtained for any string found in the source code file but not stored in exactly that form in the symbol table. Sorting the words by the number they produce provides a list in "most likely " order.

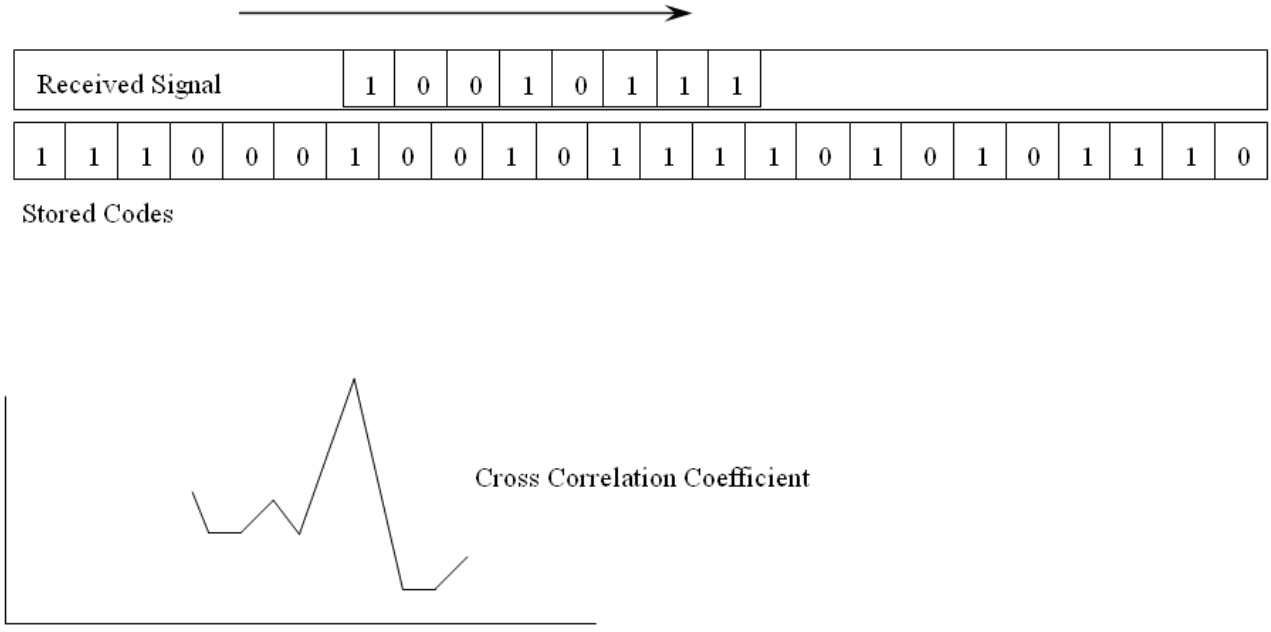

Figure 3 A received sequence is compared with each of the possible error free chain codes. The best match has the highest correlation coefficient.

In practice arrays or pointers can be used to effectively slide the words past each other and compare each character with its current counterpart.

The pseudo code Figure 4 shows how the program works.

There is too much information offered, by this implementation, when an error is found to include it all in this paper.

The quantity may seem a little daunting at first but it is logical. The output consists of a symbol table complete to the point of the error, with the source line of the declaration shown. A copy of the source code up to the point where the error is found is also given with a pointer to the position along the line the compiler has reached. The user also has the ability to move to the error position in the source code merely by mouse clicking in the editor window.

This routine has been successfully integrated with the simple recursive descent compiler in the microworld. Observing it in action in the classroom has shown that the dynamics of student tutor interaction has changed. Less time is spent working on simple typographical errors. Some students are inquisitive to know how the method works and a simple one button press reveals, immediately on the screen the extended symbol table generated so far and the tokenised code developed to that point. All is designed to increase awareness and understanding of the processes involved.

Informal feedback from novice programmer students has been positive. Feed back from advanced 3rd year programming major students has not been deliberately sought but some that have been used to more complex environments have initially expressed surprised and are impressed by the ability to offer solutions rather than just state problems.

```
Repeat for each word in the symbol table
      Align end-of-search-word with start-of-symbol-table-word.
                                                ‘start of comparison
      Repeat until start of symbol-table-word compared with end of search word
            For each character position for the search word
                  if there is a match
                        give a value of one,
                         else
                        give a value of zero.
                  endif
            endFor
            Total the ones, for this word in this position. ‘ calculate
            correlation coefficient
            Store the result, for this word in this position.
            Slide the search string one character on.
      endRepeat
      Find largest correlation coefficient for this symbol table entry and save
it
endRepeat
Sort all symbol table correlation coefficients by value’ The best match will
have the highest value.
Select (say) the largest 3 values
Return with the search-word and the 3 words with the largest correlation
coefficient.
```

Figure 4 The Psuedo Code for determing correlationDiscussion

9 Summary

Identifying programmer errors, of typographical, spelling and misremembered names as a form of data corruption triggered the approach to signal processing techniques of error detection and correction. Applying a concept used to improve the success rate of digital data transmissions from a satellite in orbit around Mars to text “discovered” by a compiler produced several benefits, the most important being a suggestion to the programmer of the nearest “fit” word.

The whole exercise could later be used as an educational experience enabling and enhancing at least the following

- Concepts of error detection and correction
- Codes and coding
- Physical errors through corruption of the data
- Physical errors by corruption at source
- A compilers need for accuracy

```
Error List and Symbol Table        <Greeter.kar> 14/03/2007              9:55:49 a.m.

 "1"       #2000-06-01#   #1899-12-30 15:05:41#


         Base Class Class         Method Name               Source Line

         *         ur_Robot       turnOff                   0
         *         ur_Robot       turnLeft                  0
..
         ur_Robot  Robot          facingSouth               0
         ur_Robot  Robot          facingEast                0
..                                                          1
         ur_Robot  Greeter        turnAround                2
         ur_Robot  Greeter        turnRight                 3
..
         ur_Robot  Greeter        printHello                14

 1       class Greeter: ur_Robot
..
 17      void Greeter:: yurnAtpund ()
         ........................^
Error 1 GPM05> The Class > Greeter < does not have the Method Name > yurnAtpund < Defined.
You may have intended to write > turnAround <

I have also found the following. . . .
. . . .
```

Figure 5 A subset of the error output

However, by then the lessons of structure and correctness, that this technique should have allowed the novice programmer to concentrate on, should have been learned.

10 References

Burrell, C. J. (2001, October 10 - 13). Visualising and interpreting individual student models developed whilst learning the foundations of object oriented programming. Paper presented at the 31st ASEE/IEEE Frontiers in Education Conference, Reno, NV.

Burrell, C. J. (2002, November). Micrososft/APNZ Staff Award. Paper presented at the The Association of Polytechnics in New Zealand and Tairawhiti Polytechnic (APNZ) Conference and Awards Dinner Celebrating Diversity: a regional perspective, Gisborne.

Burrell, C. J. (2003). Observing Novice Programmer skill development: A micro-world that supports object-oriented programming and usage data visualisation. Paper presented at the Third International Conference on Science, Mathematics and Technology Education: "Making Science, Technology and Technology Acessible to All", East London, South Africa.

Burrell, C. J. (2004). The acquisition and analysis of student models developed while learning programming skills in a microworld learning environment. Unpublished Doctor of Philosophy, Curtin University of Technology, Perth, Australia.

Golding, A. R., & Roth, D. (1999). A Winnow-Based Approach to Context-Sensitive Spelling Correction. Machine Learning, 34(Numbers 1-3), 107-130.

Instrumentation Units 11,12,13: Noise in Instrumentation Systems. (1974). Milton Keynes: The Open University Press.

McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing student's knowledge of programming. Journal of Research on Computing in Education, 29(3), 276-297.