A grading procedure for PL/1 student exercises

By J. F. Temperly and Barry W. Smith*

A procedure to supply test data for a number of undergraduate programming exercises in the PL/1 language and check the validity of the programs is described. The procedure provides diagnostic information to the student and performs all necessary output, as well as maintaining complete records of student performance on magnetic disc storage. The procedure differs from many previous grading routines in being called as a precompiled library subroutine, and is the first known grading procedure for PL/1. The initial set of class problems and specimen output listings are appended.

In view of recent interest in what have become known as "grading programs" (Perlis and Braden, 1965; Forsythe and Wirth, 1965; Berry, 1966), the following description of another such program successfully implemented on an IBM System/360 Model 50 computer may be of value. Whereas previous published work has been almost exclusively for programs written in ALGOL or its dialects, this is the first known grading program for PL/1 exercises.

Beginning in the third term of the 1966 academic year, introductory courses in digital computing have been given as part of undergraduate units in Statistics and Accounting at the Australian National University. The PL/1 language (IBM Corporation, 1966) has been used for both these courses and, especially for the longer course for statistics students, considerable emphasis has been given to the practical solution of problems.

A decision was made to use automatic grading by computer of all student attempts at class problems, for several reasons:

(i) the volume of work required, at least potentially, for tutors to carry out the grading and associated record-keeping;
(ii) the apparent difficulty of teaching input/output sufficiently early in a course for students to begin writing programs;
(iii) the ease with which input test values can be supplied to all students and results printed under control of a grading procedure; and
(iv) the intrinsic interest of the task.

While there are some obvious deficiencies in the PL/1 language in its present form, especially in relation to the very complex set of rules governing assignment statements and the evaluation of expressions involving data elements of differing attributes, and while even more obvious criticisms can be made of the efficiency of implementation of the language in the early versions of the IBM compiler for the System/360, the writers are convinced that the general form of the language represents a real advance on most previous languages. In particular, it is believed that PL/1 is a good vehicle for undergraduate courses in digital computing for the following reasons:

(i) The language is of very general applicability, and is not specifically tailored to the needs of any particular class of applications, whether scientific, commercial, textual or symbolic. It is thus equally apt for students in the physical sciences, accounting, pure mathematics, statistics or linguistics, these being the disciplines in which the greatest interest has so far been expressed in the undergraduate use of computers at this University.
(ii) If the intention is to teach more than mere coding or programming techniques, the richness of the language makes it particularly easy to illustrate fundamental notions such as those of data representation and structure, program structure, parametrization and recursion. No other language readily permits as comprehensive a set of examples as can be based on the use of PL/1.
(iii) The design of PL/1 permits the teacher to a very large extent to select his own subset of the language and to introduce new features and concepts in the order he prefers.
(iv) The student who has mastered PL/1 should have little difficulty in learning to use another procedure-oriented language (more readily than the converse). This is especially relevant because of the limited acceptance and availability of PL/1 at present.

Since it was decided to use PL/1 for teaching purposes, it was obviously desirable for the grading programs to be written in the same language for maximum compatibility, and the use of PL/1 for these programs provided useful additional experience of the language in a novel environment. The work described here represents one of the earliest major tasks using PL/1 undertaken at this installation. Few significant problems were presented by the use of PL/1 for grading; the initial difficulties encountered arose mainly from the use of a restricted pre-release compiler and from initial errors in system programs. On the other hand, the use of PL/1 for grading simplified the task in several ways, in particular the availability of a powerful means of creating and maintaining random-access files. In addition, given

* *Computer Centre, The Australian National University, Canberra, A.C.T.* 2600, *Australia.*

that the exercises were written in PL/1, it was possible to grade a mixture of problems, the test data and results of which were in a variety of formats, including arrays, structures, bit-strings and character-strings, as well as scalar numeric values, without concern for PL/1 implementation conventions. The range of examples that could be both coded and graded in an elementary course was extended to include some involving extensive character manipulation. In other words, there were no restrictions on the range of possible student exercises that could be given; in future, the set of exercises could be expanded by inclusion of problems of any degree of complexity, making explicit use of any PL/1 feature. The only forseeable limitations are those which may arise from the difficulty of checking the solutions to some problems—a difficulty that is language-independent.

Outline of the system

Two types of test run are provided. Each student is allowed an unlimited number of test or type "T" runs (although the number is counted) but only one grading or type "G" run for each exercise. Both provide test data, but the data for "T" runs are not intended to be exhaustive and do not cover all special cases.

The system to perform all the tasks associated with student exercises has three main parts:

(i) Functions which are common to the checking of all exercises, e.g. updating student records on disc and timing of the execution of programs.
(ii) Functions specific to each exercise, i.e. provision of data and the checking of results.
(iii) The summarization and analysis of records for student attempts, both by student and by exercise.

All three have been to some extent implemented in PL/1 on the University's IBM System/360 Model 50, the initial programs being compiled with a pre-release copy of version 1 of the PL/1 compiler.

The first part of the design was satisfactorily realized and presented no problems when put into use. However, at first only stream-oriented input/output was available and the maintenance of records of student performance on disc was unnecessarily slow. Record-oriented input/output has since made possible substantial improvement in a second version of the system.

The routines common to all exercises perform the following functions:

(i) checking of student number, exercise number and run type (i.e., test or grading run), with provision for an informative error message and job termination if these are invalid;
(ii) control of printer spacing for output (there is provision for one, two or three output test sets to be printed on a page, depending on the nature of the exercise);
(iii) calculation and printing of the execution time for each set of test data;
(iv) accumulation of the total time for each test or grading run;
(v) checking, for grading runs, that there has been no previous grading run for the same exercise for the same student, and updating on disc the number of test or grading runs for the student and exercise; and
(vi) for grading runs, updating on disc the number of supplied input data sets for which results were correctly returned by the program under test.

The second part of the system was completed for an initial set of fifteen different class problems of varying complexity (shown in Appendix 1). The addition of further exercises is planned. The general approach adopted resembles that taken at the Carnegie Institute of Technology for ALGOL exercises and described by Perlis and Braden (1965).

Initially, both parts (i) and (ii) described above were included in a single, large subroutine to be invoked by the students' main programs, using a different entry point for each exercise. This is perhaps the reverse of what one might expect of a supervisory routine, but has some advantages. If the supervisory routine performing the grading were the main procedure to which students' programs were supplied as data, not only would the compiler itself need to be invoked as a subroutine of the supervisor, but also problems of recovery would arise if a student program terminated abnormally in execution, a common enough occurrence. With the grading routine invoked by each student procedure, however, normal operating procedures can be followed; an abnormal termination causes the normal monitor to proceed to the next job, i.e. the next student program. Further, as Forsythe and Wirth (1965) point out, if one has the facility—available in PL/1 and FORTRAN but not usually in ALGOL—of placing the grading routines in relocatable object form in the system library, much unnecessary compilation is avoided.

Implementing the grading routine as a procedure called as a subroutine with appropriate parameters proved quite satisfactory so long as it was INTERNAL to the student's main procedure; however, because of an error in the company's software, since remedied, the grading routine could not at first be invoked successfully as an EXTERNAL procedure at all of its entry points.

Only a limited amount of subsequent analysis of records of student attempts was necessary for the first course, and the output from a sample analysis is given in Appendix 3.

The second version of the system which uses record-oriented input/output was a little different in design. While the student's program was still the main procedure, the testing routine was broken into seventeen separate EXTERNAL procedures called as subroutines, and these were stored as a library of load modules (relocatable binary object programs). There are thus now separate subroutines for each class exercise and two which perform most of the functions common to all exercises.

The combined effect of using the additional PL/1 language features supported in the more recent compilers, and of the redesign, was to reduce the time needed for each student exercise by a factor of two. Part of the improvement may be attributable to general improvements in company software and to consequent reductions in job step overhead time.

The grading routine

The groups of undergraduates taught vary widely in programming experience. Some in fact are experienced professional programmers doing part-time courses; most have no previous computer experience. As no previous experience was assumed, to avoid detailed, irrelevant and confusing explanations of job accounting conventions and the mechanics of grading, students are provided with mimeographed sheets showing all job control statements and any DECLARE and CALL statements needed specifically for grading purposes. These differ from exercise to exercise mainly in the number and type of arguments to be passed to the grading routine; consequently, a different entry point is used for each exercise.

For example, exercise 9 requires the determination of the highest common factor and lowest common multiple of two positive integers. Students are advised to include in their programs the declaration—

```
DECLARE CHECK09 ENTRY (FIXED BINARY, CHARACTER(1),
        FIXED BIN, FIXED BIN, FIXED BIN, FIXED BIN),
        (N1,N2,HCF,LCM) FIXED BINARY;
```

This declares the required entry point (CHECK09) to the grading routine and the attributes of its parameters (which do not otherwise concern the student), and the attributes of the supplied test values (N1 and N2) and of the computed results (HCF and LCM).

In almost all cases the student's program, after any initial steps, comprises a loop containing his algorithm, and at the beginning of this there is a CALL to the grading subroutine. In the case of exercise 9, students would be told that this is to be—

```
CALL CHECK09 (student-number,{'T'|'G'},N1,N2,HCF,LCM);
```

This supplies an initial set of test values (N1 and N2) on the first invocation, and also causes the result variables (HCF and LCM) to be set to zero to avoid the correct results being obtained by accident. At the end of the loop, the student returns control back to the beginning, i.e. to the CALL-statement. On the second and subsequent invocations of the grading program, all necessary checking of the results is carried out, the results obtained by the student are printed (and the correct results if these were not the same), and another set of test values is supplied if required—or, if all sets of data have been processed, the program is terminated.

Thus the subroutine specific to each exercise, where appropriate—

(i) provides test data for the student's solution;
(ii) sets all result variables initially to impossible values to ensure that correct results do not arise by accident;
(iii) prints the correct results for the test data; and
(iv) prints diagnostics and the student's results if these differed from the correct results. (An algorithm known to be correct is included for each exercise. For example, in the case above, the algorithm used in the grading subroutine CHECK09 is the well-known Euclidean algorithm (Birkhoff and MacLane, 1953).)

Where appropriate the sets of input data for "G" runs are designed to be "orthogonal" in the sense that each set independently tests a separate boundary condition or a general class of cases, and that collectively they cover all relevant conditions that the student is expected to allow for. Thus the CHECK09 subroutine provides the following test values—

N1	N2	HCF	LCM
11	1	1	11
21	9	3	63
36	25	1	900
56	49	7	392
81	81	81	81
111	121	1	13431

This approach to the selection of test cases appears appropriate for two reasons. First, the number of correct sets of results obtained by a student can be used as a reasonably fair indication of the merit of his solution, and avoids grading on an "all or nothing" basis. If a student algorithm fails to compute the correct highest common factor and lowest common multiple when the two integers are equal or when one is unity, but properly handles all other cases, he gains a score of 4/6. Secondly, "orthogonal" data sets reduce the number of cases needed to test a solution. This is especially important since a disc reference is needed after each set of test data is used; if this were not done, and the disc record were updated only at the end of all trial solutions, information can be lost when one of the data sets causes the student program to terminate abnormally, or if the job time limit is exceeded. With only one "G" run allowed for each student, this would be unfair.

Once the course has dealt with input/output features, students are encouraged to attempt some program testing of their own, by submitting problems to the Computer Centre in the normal way for batch processing. In this case they make no reference to the checking routines, use their own test data and print their own results. Once students are satisfied with their programs, whether after one or more "T" runs or after their own test runs or both, they must then submit them for grading in the usual way. Type "G" runs must not include any input/output statements (except in one specific problem that requires the printing of a table of constants in a suitable format).

Appendix 1 shows the set of exercises initially used, and Appendices 2 and 3, respectively, show specimen outputs from the grading routine and the analysis routine for partially correct results. A copy of the complete program can be provided on request.

Appendix 1

Initial set of fifteen student exercises (for statistics students)

Write PL/1 programs for the following:

1. Compute $n!$ $(0 \leqslant n \leqslant 13)$.
2. Given a real value x $(0 < x < 0{\cdot}5)$ compute
$$y = 1 + x + x^2 + x^3 + \ldots,$$
continuing until the term added is less than 10^{-4}.
(Do not for the exercise use the fact that $y = (1 - x)^{-1}$.)
3. Given a real value x $(|x| < 1)$, compute e^{-x} from the series
$$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} \ldots,$$
continuing until the term added is less than 10^{-4}.
4. Compute the terms of the Fibonacci series
$$F(0) = 0, F(1) = 1, F(n) = F(n - 1) + F(n - 2)$$
for $n = 1$ to 40 and store in a FIXED BINARY (31,0) array.
5. Place the values of the first n prime numbers $(n \leqslant 200)$ in an array of FIXED BINARY (31,0) attributes.
6. Program to evaluate quadratics:
Given successive sets of constants A_n,B_n,C_n, calculate
$$Y_n = A_n x^2 + B_n x + C_n$$
for $x = 0$ to 1 inclusively in increments of $0{\cdot}1$.
For each set of constants store the values of Y_n as a vector of 11 values (A_n,B_n,C_n and Y_n are all to be assumed FLOAT BIN.)
7. Given FLOAT BINARY values a, b, c, obtain the roots of the quadratic equation
$$ax^2 + bx + c = 0.$$
If they are real set a bit string of length 1 equal to 1. If the roots are complex, set the bit string equal to 0. (Note that it is quicker to obtain the value of the second root by using the fact that the sum of the roots is $-b/a$.)
8. Given two square matrices of dimension (10×10), compute the matrix product and its transpose.
9. Compute the highest common factor and lowest common multiple of two positive integers.
10. A customer tenders an amount A in either decimal or £ s. d. currency for a purchase of value V, again specified either in decimal or £ s. d. currency. Compute the *exact* change in the appropriate currency which minimizes the number of notes and coins. Do not assume that the customer is either honest or conversant with the currency laws. Maximum transaction is $1,000 or £500 and no halfpennies occur.

Input: a structure

```
1—INPUT (2) (Subscript 1 denotes amount tendered;
             Subscript 2 the value of purchase)
    2—CODE CHAR (1) (D=Decimal) (L = £ s. d.)
    2—AMOUNT IF DECIMAL FIXED DEC (6,2)
    2—AMOUNT IF £ s. d.
        3—POUNDS FIXED DEC (3,0)
        3—SHILLINGS FIXED DEC (2,0)
        3—PENCE FIXED DEC (2,0)
```

Output: a structure

```
1—OUTPUT
    2—CODE CHAR (1) X = Satisfactory Purchase
                    * = Insufficient Tendered
                    Q = Amount Tendered not
                        in acceptable form
    2—ARRAY (12)—all elements FIXED DEC (2)
```

Subscript	1	denotes	no.	@	$20
,,	2	,,	,,	,,	$10
,,	3	,,	,,	,,	$2
,,	4	,,	,,	,,	$1
,,	5	,,	,,	,,	50c.
,,	6	,,	,,	,,	20c.
,,	7	,,	,,	,,	10c.
,,	8	,,	,,	,,	5c.
,,	9	,,	,,	,,	2c.
,,	10	,,	,,	,,	1c.
,,	11	,,	,,	,,	3d.
,,	12	,,	,,	,,	1d.

Note: 1. If the currency tendered is not the currency of the purchase price, the amount tendered must be an integral multiple of 5 cents or 6d.
2. If change is required that is not an integral multiple of 5 cents or 6d., it is given in the currency of the purchase price.
3. The amount tendered must not be less than the purchase price.

11. Given an array of 30 FLOAT BINARY values, compute their mean, standard deviation, variance, skewness and kurtosis. Store these values in the above order in 5 successive locations in an output FLOAT BINARY array.
12. Given the x and y co-ordinates of two points $P_1(x_1,y_1)$, $P_2(x_2,y_2)$ which are opposite vertices of a square, determine whether a third point $P_3(x_3,y_3)$ is contained within the square, is on the square or is outside the square. Output a single CHARACTER
(I = inside, O = on, E = outside).
13. Using your own output statements, construct a table showing for values of $n = 0,1,\ldots,10$ the following functions:
$$n, n^2, n!,$$
$$e^{-m}\frac{m^n}{n!} (m = 1,2,\ldots 9).$$
(For checking of your program, store the results in an array of dimension (11,12).)
14. Given three fixed integer decimal values which respectively represent day, month and year (e.g. 14, 7, 1946 might represent your birthday) for any date since the inception of the Gregorian Calendar until the year 9999, determine the day of the week and output a single decimal integer value (0 = Sunday, 1 = Monday, . . ., 6 = Saturday, 9 = invalid date).
The day, month and year will be the first three locations of a FIXED DECIMAL array and the result is to be stored in the fourth location of that array.
(Every year which is exactly divisible by 4 is a leap year, except for those divisible by 100 which are not, excepting

again those divisible by 400 which *are* leap years. You need to know also the day of the week for at least one date.)

15. You are given a piece of natural language text as a PACKED array of 300 single characters. The characters may include any letter of the alphabet, blanks or the punctuation marks full stop, comma, semi-colon or apostrophe. Words are preceded by blanks (except for the first word) and followed by a blank, full stop, comma or semi-colon. Words may contain an apostrophe. Sentences conclude with a full stop. Count the number of—

    (1) Sentences.
    (2) Words.
    (3) Occurrences of the word "I".
    (4) Occurrences of the letter "I".
    (5) Occurrences of words beginning with "I".

    If any vowel occurs more frequently than "I", output it as a single character, else output "I".

Appendix 2

Specimen output from the grading routine

```
EXERCISE 2—TABLE OF FIRST 150 PRIMES
STEP TIME 3·400 SECONDS
PROGRAM CORRECT LAST STEP
CORRECT RESULTS ARE

      2      3      5      7     11     13     17     19     23     29
     31     37     41     43     47     53     59     61     67     71
     73     79     83     89     97    101    103    107    109    113
    127    131    137    139    149    151    157    163    167    173
    179    181    191    193    197    199    211    223    227    229
    233    239    241    251    257    263    269    271    277    281
    283    293    307    311    313    317    331    337    347    349
    353    359    367    373    379    383    389    397    401    409
    419    421    431    433    439    443    449    457    461    463
    467    479    487    491    499    503    509    521    523    541
    547    557    563    569    571    577    587    593    599    601
    607    613    617    619    631    641    643    647    653    659
    661    673    677    683    691    701    709    719    727    733
    739    743    751    757    761    769    773    787    797    809
    811    821    823    827    829    839    853    857    859    863

EXERCISE 5—COMPUTE N FACTORIAL
STEP TIME 0·020 SECONDS
PROGRAM CORRECT LAST STEP
CORRECT RESULT IS 2 FACTORIAL = 2

STEP TIME 0·020 SECONDS
PROGRAM TESTED GAVE INCORRECT RESULT 5 FACTORIAL = 0
CORRECT RESULT IS 5 FACTORIAL = 120

STEP TIME 0·000 SECONDS
PROGRAM TESTED GAVE INCORRECT RESULT 8 FACTORIAL = 0
CORRECT RESULT IS 8 FACTORIAL = 40320

TOTAL TIME USED BY PROGRAM TESTED = 0·040 SECONDS

EXERCISE 10—BUSINESS TRANSACTION PROBLEM
WARNING—NO CHECK IS MADE THAT INPUT DATA VALUES ARE UNCHANGED
STEP TIME 0·020 SECONDS
PROGRAM TESTED GAVE INCORRECT RESULT
CHANGE MADE UP AS FOLLOWS
0 x $20, 0 x $10, 0 x $2, 0 x $1, 0 x 50C, 0 x 20C, 0 x 10C, 0 x 5C, 0 x 3D, 0 x 2C, 0 x 1C, 0 x 1D
TRANSACTION RESULT CODE = X
CORRECT RESULTS ARE
CHANGE—MADE UP AS FOLLOWS
19 x $20, 1 x $10, 2 x $2, 0 x $1, 0 x 50C, 2 x 20C, 0 x 10C, 1 x 5C, 0 x 3D, 0 x 2C, 1 x 1C, 0 x 1D
TRANSACTION RESULT CODE = X
INPUT DATA
PRICE = $105·54                TENDERED AMOUNT = $500·00
```

```
STEP TIME 0·020 SECONDS
PROGRAM TESTED GAVE INCORRECT RESULT
TRANSACTION RESULT CODE = Q
CORRECT RESULTS ARE
CHANGE—MADE UP AS FOLLOWS
0 x $20, 0 x $10, 2 x $2, 0 x $1, 0 x 50C, 2 x 20C, 0 x 10C, 1 x 5C, 0 x 3D, 0 x 2C, 0 x 1C, 1 x 1D
TRANSACTION RESULT CODE = X
INPUT DATA
PRICE = STG 52/15/ 5          TENDERED AMOUNT = $110·00

STEP TIME 0·020 SECONDS
PROGRAM CORRECT LAST STEP
TRANSACTION RESULT CODE = Q
INPUT DATA
PRICE = STG 52/15/ 5          TENDERED AMOUNT = $105·54

STEP TIME 0·020 SECONDS
PROGRAM CORRECT LAST STEP
TRANSACTION RESULT CODE = *
INPUT DATA
PRICE = STG 52/15/ 5          TENDERED AMOUNT = $100·00

TOTAL TIME USED BY PROGRAM TESTED = 0·080 SECONDS
```

Appendix 3

Specimen analysis by student

```
STUDENT NUMBER 2

 EXERCISE      NUMBER OF       SCORE ON        TIME USED         COMMENTS
  NUMBER       TEST RUNS      GRADING RUN     GRADING RUN
                                RAW  %          SECONDS

    1          NO ATTEMPT     NO ATTEMPT
    2          NO ATTEMPT     NO ATTEMPT
    3              5          5 100·00           0·00
    4              1          5  50·00           0·06
    5          NO ATTEMPT     NO ATTEMPT
    6          NO ATTEMPT     NO ATTEMPT
    7              2          3 100·00           0·01
    8              1          NO ATTEMPT
    9          NO ATTEMPT     NO ATTEMPT
   10          NO ATTEMPT     NO ATTEMPT
   11              2          NO ATTEMPT
   12              3          NO ATTEMPT
   13              3          NO ATTEMPT
   14              1          NO ATTEMPT
   15              1          NO ATTEMPT

SUMMARY—STUDENT 2

NO ATTEMPT MADE TO TEST-RUN 6 EXERCISE (S)
12 EXERCISE (S) NOT GRADED
AGGREGATE SCORE FOR 3 EXERCISE (S) = 13
PERCENTAGE SCORE FOR 3 EXERCISE (S) = 72·22
```

References

BERRY, R. E. (1966). Grader programs, *Computer Journal*, Vol. 9, p. 252.

BIRKHOFF, G., and MACLANE, S. (1953). *A Survey of Modern Algebra*, New York: Macmillan, pp. 16–20.

FORSYTHE, G. E., and WIRTH, N. (1965). Automatic Grading Programs, *Comm. A.C.M.*, Vol. 8, p. 275.

IBM CORPORATION (1966). *IBM System/360 Operating System: PL/1 Language Specifications*, IBM Systems Reference Library, Form C28–6571–4, New York: IBM Corporation.

PERLIS, A. J., and BRADEN, R. T. (1965). *An Introductory course in computer programming*, Monograph No. 7—Discrete System Concepts Project, Pittsburgh: Carnegie Institute of Technology.