An Automatic Grading Scheme for Simple Programming Exercises

J. B. HEXT AND J. W. WININGS
University of Sydney, Sydney, Australia*

A discussion is given of alterations that were made to a typical university operating system to record the results of programming exercises in three different languages, including assembly language. In this computer-controlled grading scheme provision is made for testing with programmer-supplied data and for final runs with system-supplied data. Exercises run under the scheme may be mixed with other programs, and no special recognition of exercises by the operators is necessary.

KEY WORDS AND PHRASES: automatic grading program, programming exercises
CR CATEGORIES: 1.5, 2.43, 4.39

Introduction

In 1966 the University of Sydney's KDF 9 computer was running student exercises under a conventional batch-processing system. With the numbers of exercises amounting to several hundred per week, and threatening to run into thousands, it became desirable to provide some means of recording, checking, and summarizing results. To meet this need, the Basser Automatic Grading Scheme (BAGS) was developed.

The idea of a computer-controlled grading scheme was not new. Earlier work by Hollingsworth [1] had already shown its value, and other reports described how the basic idea could be elaborated for exercises in ALGOL [2, 3]. A later paper by Temperly and Smith [4] describes a versatile system for exercises in PL/1. BAGS, however, differs from most other schemes in two major respects: first, it can handle exercises in several different languages; second, it requires no special action by the operators. Whereas most other schemes embed their exercises in some larger programs, BAGS is simply part of the standard operating system and its exercises are run as normal, batch-processed jobs.

The basic requirements of the scheme were as follows:

(1) It should handle exercises in ALGOL, in MINIGOL (a subset of ALGOL) and in the KDF 9 Assembly Code.

(2) It should not place any additional burden on the operators.

(3) It should record every attempt at an exercise, with sufficient data for calculating a mark.

(4) It should provide summaries on request for specified classes and exercises over a given period.

In the following sections implementation of the scheme and its method of grading are described, and experience in its use is discussed. Sample exercises and results are also given.

Implementation

By 1966 the batch-processing system had evolved into the following sequence of operations:

(1) Loading a batch of source programs onto magnetic tape in parallel with the processing of other jobs. These programs could be in any of nine languages and could be presented on 5-channel tape, 8-channel tape, or cards. A variety of codes could be used, representing the standard set of symbols used by the manufacturer's software.

(2) Converting this file to the standard code and to a fixed format on another reel.

(3) Automatic processing of the file without operator intervention. Complete hardware protection of the monitor makes this possible for programs written in assembly code as well as in higher level languages.

The changes to this basic system necessary to include a general and flexible automatic grading scheme were almost trivial. The format conversion program was altered to insert, at the option of the programmer, one of several sets of data made available for a particular exercise. Processors for three of the available languages were modified to note the progress of the job; and the job control program, which supervises the automatic processing of the file, was made to write a four-word record of the performance of each job onto magnetic tape. Separate programs were written to form a cumulative file of records and to interrogate that file to produce a report for given classes and exercises. These operations are described in more detail below.

Insertion of Data. A twelve-character program identifier contains all control information required for a job in the system. Six characters (including a check digit) identify the class or an individual research project; one character identifies the individual in the class; two characters specify the language, and one the exercise. If the first of the remaining two characters is a "D" and the second is a decimal digit, the system inserts a set of data following the

* Basser Computing Department

user's program. This data is a sequence of characters, as if supplied by the programmer himself. The examiner may provide as many sets of data as he likes for a given exercise. He may also choose, at the time the data is established, whether one of these sets is to be selected at random or whether the programmer is to specify which of the sets he requires.

Language Processors. There are currently twelve compilers, interpreters, simulators, assemblers, and macro-processors incorporated in the system. Of these, three have been modified to preserve data from student jobs for subsequent grading. One is an ALGOL interpreter characterized by excellent run-time diagnostics and slow execution speed. The second is MINIGOL, a compatible subset of ALGOL with good diagnostics and good execution speed. The third is the input-output package used for assembly language programs; catastrophic errors cause this package to be entered, making it difficult, though not quite impossible, for the cunning student to prevent a record being saved. The grading data is preserved in general purpose registers at job termination, and includes the twelve character program identifier, the number of the data set, the central processor time used during execution, two 48-bit results, and a mark of 0 for a compilation failure, 1 for a run-time failure, and 2 for successful termination.

Job Control Program. There is provision in the KDF 9 system for repeated overwriting of a fixed length block on magnetic tape. This feature is utilized to allow the record of each job to be stored on the system library tape, thus avoiding the necessity of dedicating one of the four tape drives to this purpose. On completion of a job, the control program checks the program identifier to see whether it is instructional; if so, the block for recording marks is read from the system tape, the new four-word record (including a 16-bit sum check) is added, and the block is rewritten. It proved possible to devise a format for this block which makes it invisible to the other systems programs normally used in maintaining library tapes.

There is space for 800 records in this block. Normally the marks block is dumped to the cumulative file on another reel daily. If the block fills during a batch, the control program can call in a segment to dump the block at that time. The control program can also recognize if it is using a standard library tape (i.e. without provision for the marks block) and if so, it can inhibit recording. The date on which the block was initialized and the date it was dumped are recorded on the cumulative file.

The recording process is quite independent of class, language, exercise, or data set. It can thus be used by a number of different classes simultaneously, each using a different set of exercises. Throughput achieved in this system is up to 15 jobs per minute.

Method of Grading

The maximum mark for an exercise is 5. One mark is gained for each of the following criteria:

(1) The program compiled successfully.

(2) The program ran to completion.

(3) The first result was correct.

(4) The second result was correct.

(5) The above four points were satisfied in sufficiently short central processor time.

All the information necessary for assessing this mark is recorded by the operating system as described above.

The program which calculates the mark is known as the Report Program. For each exercise, the Report Program contains five constants: these are the two correct results, a tolerance for each result, and a time allowance. If an exercise may take one of several data sets, there is a separate result set for each.

A request to the Report Program gives a list of classes and the exercises to be summarized for each. It also gives two dates—blocks of marks are ignored if recorded outside these two dates. A class is specified by a deck of cards: the first card gives the class number, the second lists the exercises to be checked, and the remainder list the members of the class. There is one card per student, giving his one-letter identifier and his name. This is the only point at which BAGS is given the actual names of the students.

The Report Program lists results one page per class. A sample page is shown in Figure 1. In the main table there is one line per student and one column per exercise, and each entry has the form i/j. This states that the student made i attempts at the exercise and that his best mark was j. The ideal mark, of course, is 1/5, indicating that his program satisfied all five criteria first time. An additional

BASSER AUTOMATIC GRADING SCHEME

SUMMARY FOR CLASS 1636 FOR PERIOD 28/02/68 TO 01/07/68

		WE	MA	M2	TA	TE	TOTAL	GRADE
JAUL M	A	2/1	4/5	4/5	3/3	-	13/14	36.90
KAROLIS C	B	1/5	1/5	1/5	1/5	2/5	6/25	96.67
KEELING J A	C	4/2	5/5	6/5	2/5	1/2	18/19	50.78
KENYON P H	D	2/3	1/5	2/0	-	-	5/8	30.00
KIDD C	E	-	1/5	2/2	-	1/0	4/7	26.67
KITE D F A	F	3/4	1/5	2/5	6/5	5/2	17/21	62.54
KNAGGS H J	G	3/5	2/5	2/5	3/5	3/2	13/22	67.62
LANDAU L	H	6/2	2/5	5/5	11/3	9/2	33/17	38.85
LUCKY U	I	1/5	1/5	1/5	1/5	1/5	5/25	100.00
LUTHER D	J	3/3	2/5	3/3	4/5	51/3	63/19	47.40
MCFADYEN B E	K	3/4	1/5	3/5	3/5	4/2	14/21	65.00
MARR A J	L	6/2	2/5	3/1	1/5	11/2	23/15	46.19
MILLER R L	M	7/5	1/5	2/5	4/5	12/2	26/22	60.76
MITCHELL L J	N	5/3	1/5	1/5	3/5	5/2	15/20	65.40

CLASS AVERAGE 56.77

NUMBER OF RUNS WITH RESULT

0	-	7	17	8	56	88
1	10	-	5	8	13	36
2	22	1	3	6	31	63
3	4	-	1	10	3	18
4	5	-	-	-	-	5
5	5	17	11	10	2	45

TOTAL RUNS 255

NOTE - I/J MEANS I RUNS WITH BEST MARK J

GRADE IS AVERAGE OF (100 * J) / (4 + I)

FIG. 1.

column gives his total number of attempts and his total mark, i.e. $\Sigma i/\Sigma j$. The final column gives his grade, calculated according to the formula

$$\text{Grade} = \text{average of } \frac{100 \times j}{4 + i}$$

the average being taken over all the exercises being marked. This formula will give 100 percent to the student who scores 1/5 for every exercise. He is penalized for every mark lost and for every additional attempt. After six attempts, for example, a successful run scores only 50 percent. The average grade of the class is displayed beneath the table in order to encourage an element of competition between the classes.

The results also include a table giving statistics on the attempts made at each exercise. The first row gives the number of attempts at each exercise which scored 0 marks, followed by the total over all exercises; the second row gives the figures for attempts scoring 1 mark; and so on. A final figure gives the total number of runs made by the class. This table gives some indication of the difficulty of the exercises, and the totals give a second summary of the performance of the class.

Discussion

The experience has shown that a suitable operating system can be extended to incorporate a simple grading scheme without undue effort or disruption. The system requires no extra work by the operators and only a minimum of additional detail for the students. Exercises are run as normal programming jobs and the recording process is completely automatic.

The overhead involved for each exercise is quite small. The recording takes about 2 seconds, this being the time for adding a four-word result to the block on tape. When disks are used instead, the time will be much less. Other jobs are not affected at all; nor are they restricted in their available core or any other facility. The only other cost is in the daily updating and printing of the marks file.

It will be apparent that the grading criteria, based only on two results and a time, are considerably less sophisticated than those of some other schemes. However, it has been found that with suitably designed exercises two results and a time provide adequate checking for most purposes. More detailed criteria were considered unnecessary, since the grades are never used in assigning credit for the course: they are required only for general feedback purposes. (In assigning credit, too many uncomputable factors are involved, such as detailed technique, annotation, etc.) The advantages of the simple criteria are the ease of implementing the scheme and of supplying exercises to it: all that is required in adding a new exercise is the provision of data sets (if any) and corresponding results.

The Report Program could, of course, be extended in various ways to provide other statistics and output. In particular, additional credit should perhaps be given to a student whose program can handle all the different data sets; this would encourage him to allow for exceptional conditions in the data. The Report Program could also print out warning messages to students whose grades are too low, or the reverse to those who do well. It is a fairly routine piece of data-processing and such changes would not affect the operating system. The only change that is planned in the operating system is the dumping of marks on disk instead of on magnetic tape.

It is easy, of course, for a student to cheat—the obvious way is for him to copy someone else's program. Less blatantly, he could test all his programs using a different project number and then run his correct versions under his own number. Alternatively, he could find out the correct results by some devious means and assign them directly in his program. Tests for possible cheating can be incorporated in the Report Program, based on the size of the compiled program and the time of execution. But complete safeguards are impossible and such loopholes in the system are another reason why credit cannot be attached to any grades produced by BAGS. Our experience has not found cheating to be any problem.

The benefits of BAGS are along two lines. Firstly, it provides information to the teacher on how well a class is progressing. Students who fall behind in their practical work can be prodded, and those who do especially well can be commended. Secondly, the publication of weekly records provides an incentive to the student which previously was lacking. If the mark 1/5 is sufficiently glamorized, he will be encouraged all the more to write his programs carefully and accurately.

During the past term, over 3000 runs, covering 24 different exercises, have been recorded for our main third-year course. Several smaller classes have also been monitored. With 1500 first-year students learning MINIGOL in the next two terms, and other new courses looming ahead, these figures will further increase. The use of BAGS, coupled with TV lectures, will greatly ease the burden that they bring.

Acknowledgments. The implementation of BAGS has been the combined effort of many people. In particular, credit is due to D. Blatt for work on the input of cards and the supplying of data sets; to Messrs. Haddon, Hore, and Newey for work on the compilers; to D. Roudenko for work on the Report Program; and to M. Ratner for calculating all the correct results. A final tribute should be paid to the long-suffering students who endured the initial trials of the system.

APPENDIX A

Three sample exercises are given below, partly to illustrate the application of BAGS and partly for their general interest as programming exercises. The first is to be programmed in MINIGOL (AM), the second in ALGOL (AW), and the third in KDF 9 Assembly Code (AT). In each case, x and y refer to the two required results: in MINIGOL and ALGOL they must be the first two declared variables and must be type *real*. In Assembly Code, they

must be left as standard floating-point numbers in two designated registers.

EXERCISE AMB

The following function ln* $(1 + X)$ is an approximation for ln $(1 + X)$ in the range $0 \leq X \leq 1$:

$$\begin{aligned}\ln^*(1+X) &= 0.9974{,}442X \\ &\quad - 0.4712{,}839X^2 \\ &\quad + 0.2256{,}685X^3 \\ &\quad - 0.0587{,}527X^4\end{aligned}$$

Compute the error term, i.e. ln $(1 + X)$ − ln* $(1 + X)$, for $X = 0(0.02)1.0$. Set

x = the mean of their absolute values;

y = the absolute value of the greatest error.

Draw (by hand) a graph of the error against X.

EXERCISE AWE

Declare a real procedure

Simpint (f, a, b, n)

which integrates the function f over the range (a, b) using Simpson's rule with n intervals. This rule is given by the approximation

$$(h/3) \times (f(a) + 4f(a + h) + 2f(a + 2h) + 4f(a + 3h) + \cdots + 2f(b - 2h) + 4f(b - h) + f(b))$$

where $h = (b - a)/n$ and n is even.

Declare the real procedure

trap (x) = 0.92 × cosh (x) − cos (x)

and integrate it over $(-1, 1)$ using 2, 4, 8, 16, ⋯ intervals until two successive results differ by less than 10^{-9}. Set

x = the final result;

y = the final number of intervals.

Print out the results of the successive approximations: what would the result have been if the accuracy required was 10^{-6}? Any comments?

EXERCISE AT4

Read an integer n from data, followed by an $n \times n$ matrix A listed by rows (floating-point). Denote by $R_i(C_i)$ the sum of the absolute values of the elements in row i (column i). Set

x = trace (A), i.e. $A_{11} + A_{22} + \cdots + A_{nn}$;

y = maximum $(R_1, R_2, \cdots R_n, C_1, C_2, \cdots C_n)$.

(x = sum of eigenvalues, y = upper bound on magnitude of eigenvalues.)

RECEIVED AUGUST, 1968; REVISED NOVEMBER, 1968

REFERENCES

1. HOLLINGSWORTH, J. Automatic graders for programming classes. *Comm. ACM 3*, 10 (Oct. 1960), 528–529.
2. NAUR, P. Automatic grading of students' ALGOL programming. *BIT 4* (1964), 177–188.
3. FORSYTHE, G. E. AND WIRTH, N. Automatic grading programs. *Comm. ACM 8*, 5 (May 1965), 275–278.
4. TEMPERLY, J. F. AND SMITH, B. W. A grading procedure for PL/1 student exercises. *Comput. J. 10* (Feb. 1968), 368–370.

Algorithms

J. G. HERRIOT, Editor

The following algorithm by Bartels and Golub relates to the paper by the same authors in the Numerical Analysis department of this issue, on pages 266–268.

This concurrent publication in Communications follows a policy announced by the Editors of the two departments, J. G. Herriot and J. F. Traub, in the March 1967 issue.

ALGORITHM 350
SIMPLEX METHOD PROCEDURE EMPLOYING LU DECOMPOSITION* [H]

RICHARD H. BARTELS AND GENE H. GOLUB (Recd. 2 Aug. 1967 and 5 June 1968)

Computer Science Department, Stanford University, Stanford, CA 94305

* This project was supported in part by contracts NSF GP948 and ONR NR 044 211.

KEY WORDS AND PHRASES: simplex method, linear programming, LU decomposition, round-off errors, computational stability

CR CATEGORIES: 5.41

```
procedure linprog (m, n, kappa, G, b, d, x, z, ind, infeasible, unbounded, singular);
  value m, n; integer m, n, kappa; real z;
  array G, b, d, x; integer array ind; label infeasible, unbounded, singular;
comment linprog attacks the linear programming problem:
                maximize d^T x
                subject to Gx = b and x ≥ 0
```

Details about the methods used are given in a paper by Bartels and Golub [*Comm. ACM 12* (May 1969), 266–268].

The array $G[0{:}m-1, 0{:}n-1]$ contains the constraint coefficients. Array $b[0{:}m-1]$ contains the constraint vector, and $d[0{:}n-1]$ contains the objective function coefficients (cost vector). The computed solution will be stored in $x[0{:}n-1]$, and z will have the maximum value of the objective function if *linprog* terminates successfully. Error exit *singular* will be taken if a singular basis matrix is encountered. Error exit *infeasible* will be taken if the given problem has no basic feasible solution, and exit *unbounded* will be taken if the objective function is unbounded. If *kappa* = 0, problem (2) of the referenced paper will be set up and phase 1 entered. If $1 \leq$ *kappa* $\leq m - 1$, problem (4) of the paper will be set up and phase 1 entered. The last *kappa* columns of G will be preceded by the first $m -$ *kappa* columns of the identity matrix to form the initial basis matrix. If *kappa* = m, phase 2 computation will begin on problem (1) with variables numbered $ind[0], \cdots, ind[m-1]$ as the initial basic variables and variables numbered $ind[m], \cdots, ind[n-1]$ as the initial nonbasic variables. Hence each component of *ind* must hold an integer between 0 and $n - 1$ specified by the user. Finally, if *kappa* > m, problem (3) will be set up, and phase 2 computation will begin with variables numbered $ind[0], \cdots, ind[m]$ as the initial basic variables and variables numbered $ind[m+1], \cdots, ind[n+kappa-m-1]$ as the initial nonbasic variables. This option is of interest only because *linprog*, upon successful termination, leaves all variable numbers recorded in