ALGOL68 INSTRUCTION AT OKLAHOMA STATE UNIVERSITY

G.E. Hedrick*
Department of Computing and
Information Sciences
Oklahoma State University
Stillwater, Oklahoma 74074
U.S.A.

At Oklahoma State University ALGOL68 is taught to students whose background in programming consists primarily of programming in FORTRAN. Frequently, they have had some experience with PL/1 although it is not so extensive as their experience with FORTRAN. PL/1 is usually the only block structured language to which the students have been exposed. The students who study ALGOL68 are typically seniors, although there are some graduate students, and an occasional lowerclassman. Almost none of these students has any experience with ALGOL 60 or any ALGOL-like language at the time he begins his study of ALGOL68. About half of them have had limited experience with formal specification of programming languages.

The students learn ALGOL68 in one of three formats depending upon the number of students enrolled at a given time. The most elaborate treatment is given when there are enough students to justify a faculty member spending one-quarter of his time teaching the class as a special topics lecture course. This is the mode of instruction that is preferred by the students. A smaller group of students covers the same material in a seminar format rather than a lecture format; the students in the seminar do not have the same access to faculty as do students in a special topics lecture course. The final mode of instruction is individual study; it is this instructional format which is used when only one or two students enroll for ALGOL68. In this format the students work almost entirely on their own except that they do have access to a faculty member when they cannot adequately resolve their questions.

In each of the three modes, the students are given six programs which they are to copy and to run on either of the two machines available to them (an IBM 370/158 and an IBM 1130). These programs are complete and correct when presented to the students. By running them the students learn the non-ALGOL68 requirements, such as JCL, etc., required to execute the programs, and they achieve a familiarity with the form of ALGOL68 listings. Each of these six programs is designed to point out a feature of ALGOL68 with which they may not be familiar from their past experience with FORTRAN and/or PL/1.

*The author's participation in this conference is supported, in part, by NSF Grant MCS 76-06090.

In each of the three instructional formats the students are assigned a set of twelve programs to write also. Eight of these are presented in the appendix. These twelve programs must be written during the semester in which the student is enrolled for the course. The programs are chosen from different areas of computer science including numerical analysis, data structures, logic, and other areas. Some of these programming assignments are given in the appendix. The students cover the greatest depth of material in the lecture course and the least amount of material in the individual study course, although the programming assignments are the same for all three modes of instruction.

The primary texts for the ALGOL68 course are Lindsey and van der Muelen's Informal Introduction to ALGOL68 [5] , and Pagan's A Practical Guide to ALGOL68 [6]. The students are expected to obtain a copy of the ALGOL68 report [8] for supplemental reading, and are given a set of class notes. It is anticipated that the text Grammars for Programming Languages by Cleaveland and Uzglais [1] will be used in the future as supplemental reading also. Neither of the primary texts is used in such a way that it forces a structure on the course. In fact, they are required primarily for reference material. The ALGOL68 report costs as much as most computer science textbooks, but neither of the stated texts are as expensive as one should expect a computer science textbook to be. The cost to the student for all three textbooks is approximately the same as it would be for a course that requires two "average priced" textbooks. The author feels that these costs to the student are justified for two reasons: (1) they give the students a written statement of the syntax and semantics of the language; this can be used even when no instructor is available; and (2) no serious student of ALGOL68 should be without a copy of the formal report [8] since it is the final authority on the syntax and the semantics of the language. Hopefully, the required texts will be useful to the student as he continues his studies in graduate school.

The students begin their study with a small amount of memorization of the basic concepts and constructions. This is followed immediately by a survey of the general program structure for ALGOL68 programs. Program structure is examined by first studying straight line programs, then by looking at programs which make use of various types of control structures, rewriting each of these programs

several times in successively more concise forms. Such exercises serve to demonstrate both the ease of programming in ALGOL68 and the power of the language.

Following the study of the general program structure for ALGOL68 programs the students are introduced to the nested structure of the language in a unique way. The contour model, slightly modified, is presented to the students as a run-time environment for ALGOL68 programs. The contour model is a list structured model of the static and dynamic portions of programs written in a block structured language. In the static portion there is one "algorithm contour" for each block or procedure. In the dynamic portion there is one "record contour" for each execution of each block or procedure. The blocks are linked together to show the relationships among the contours. This model must be modified to be able to handle generators, names, and values which are not associated with range entry and range exit [2]. The introduction of the contour model at this point allows the students to obtain both an intuitive feel and a formal specification of the dynamic nature of the run-time environment for a language with nested structure. The study of the contour model is extensive using the papers by Johnson [4] and by Hedrick [2]. The students examine the Johnston paper [4] thoroughly, using examples which are ALGOL68 programs. Each example program is analyzed in depth, and to the point of enabling the students to specify what display is changed at what time during the execution of the algorithm as specified by the ALGOL68 program. At the conclusion of the study of the contour model, most of the students are familiar with what to expect at run-time from ALGOL68, and as an added benefit, also have a better understanding of nested languages in general.

After the study of the contour model the students make a study of the assembler code equivalent to given ALGOL68 programs. This serves much the same purpose as the study of the contour model in that it enhances their understanding of the compiler and helps them to know what to expect at run-time. Many educators might say that the use of a assembly language in this context is inappropriate, but this use of assembly language actually improves the student's ability to program in ALGOL68.

The next item studied by the students is how to read the ALGOL68 report [8]. This tends to be a traumatic experience for most students. Indeed it is helpful not to let them look at the report until after the first three or four sessions of study of van Wijngaarden grammars. Although it is not required of the students, Peck's ALGOL68 Companion [7] presents this topic in such a way the students rapidly learn how to read van Wijngaarden grammars. At that point the study of the report begins. The course will probably be modified to incorporate portions of the text of Cleaveland and Uzgalis [1] prior to the start of the study of the report [8]. Once into the study of the report, the early part is read in order, and the later part is studied in view of specific particular programs. The particular programs are, for the most part, those that the students have constructed earlier in the course.

A great deal of time is spent on writing actual ALGOL68 programs of various degrees of complexity. The students construct the programs while studying the other parts of the course in parallel. The programs range from a simple program to find the zeros of a quadratic polynomial at the beginning of the course, to some game-playing which requires sophisticated tree searching at the end of the course. In order to help the students write these programs, there are separate units on program design and program testing based on the article from the Journal of Data Education [3]. These units are not specific to ALGOL68 but are part of our attempt to instill good programming habits in all of our graduates independent of the language that they use to implement their programs.

It appears that the ALGOL68 instructional methods described in this paper have been quite successful. The students who have completed the course not only have a knowledge of the programming language, ALGOL68, but they also demonstrate a greater knowledge of the underlying data structures required during the execution of block structured languages than do other students at the same academic level in the computer science department. Students who have taken the ALGOL68 course demonstrate a better grasp of theoretical program description languages as well. There appears to be no significant difference among the three modes of instruction in view of the student's knowledge of the material at the end of the course. The students seem to prefer the lecture format to the other formats, however. In view of the faculty time required, the individual study format should be preferred, but the lecture format would be preferred for the comfort of the students. The amount of student effort required is another point in favor of the lecture format.

The problems encountered in the teaching of this course include the lack of a single text for the students to follow and the use of a compiler which is still being developed. It is possible to eliminate most of the first problem by adopting Pagan's text [6] and studying the topics from the text in order. Supplementary material would continue to be required for the study of run-time data structures, van Wijngaarden grammars, and transput.

The second problem is attributable to the fact that the OSU ALGOL68 compiler has been, and is being used for the course, and this compiler is in the development process. Some of the less-used features became available after a problem had been assigned. The fact that this developmental compiler is used is no longer much of a problem since the features of ALGOL68 implemented are now relatively static and changes to the compiler usually are transparent to the users.

This course has been offered once or twice a year every academic year since 1973-1973. Many of the students who have taken this course have continued their studies in graduate school. Some have even written theses which are based on ALGOL68. Thesis topics have included such things as the MODE algebra, dynamic data structures for modelling program execution, and transput topics.

REFERENCES

[1] Cleaveland, J. Craig and Robert C. Uzgalis. Grammars for programming languages. New York: Elsevier, c1977.

[2] Hedrick, G.E. An adaptation of the contour model as a runtime environment for ALGOL68. The SOKEN KIYO, v. 6, #3, Jan. 1977.

[3] Hedrick, G.E. Program planning and testing. The Journal of Data Education, v. 16, #1 Oct. 1975.

[4] Johnston, John B. The contour model of block structured processes, in Tou, Julius T. and Peter Wegner (eds.), Data Structures in Programming Languages, SIGPLAN Notices v. 6, #2, Feb., 1971.

[5] Lindsey, C.H. and S.G. van der Muelen. Informal Introduction to ALGOL68. New York: American Elsevier, c1971.

[6] Pagan, Frank G. A Practical Guide to ALGOL68. New York: John Wiley and Sons, c1976.

[7] Peck, John E.L. An ALGOL68 Companion. Vancouver, B.C.: University of British Columbia, 1972.

[8] van Wijngaarden, Aad (ed.) et al. Revised Report on the algorithmic language ALGOL68. New York: Springer-Verlag, 1976.

APPENDIX

Programming Assignments for an Introductory Course in ALGOL68

Note. These problems have been selected for the ALGOL68 constructions they cause the students to use. Several have come from introductory texts for other languages. The author regrets that he is not able to identify the original source of each problem.

Assignment #1

Problem:

1A. Construct an ALGOL68 pro-ram which will find the zeros of a polynomial using the method of bisection (see Conte & de Boor, p. 28)

1B. Construct an ALGOL68 program which will find the zeros of a polynomial using Newton's method (see Conte & de Boor, p. 28).

Notes:

1. Return program decks with your output.
2. Variables of mode COMPL are allowed.
3. In written form the operations for +:=,-:,*:=, etc. terminate with AB; e.g. PLUSAB for "plus and becomes, "MINUSAB for "minus and becomes," etc.

Assignment #2

Notes:

1. The IS and INST operators are not presently available on the OSU compiler.
2. Individual programmers do not have access to the heap in the OSU ALGOL68 compiler. (You may not use global generators in your programs.)
3. Strings, as such, cannot be declared, but you may declare row of character variables. FLEX is not permitted.

Problem:

Construct an ALGOL68 program which will simulate the SAMC computer.

Author's note:

SMAC is a simple 10 instruction, 20 location machine. It handles only integers, i, in the range $-999 \leq i \leq 999$.

Assignment #3

Notes:

1. The OSU ALGOL68 compiler uses reserved words for keywords so you cannot use keywords (e.g. FI) for variable names. This is called reserved word stropping.

Problem:

Construct an ALGOL68 program which will

1. Read a value, N
2. read N,(x,y) pairs of input values
3. print the pairs in order from left-to-right; bottom to top as they would appear on an x-y coordinate system.

Assignment #4

Problem:

A. Construct an ALGOL68 program which will

1. Read a list of 0-40 names
2. Alphabetize the list
3. Print the alphabetized list with last names and first (and middle initials).

B. Sample Input

Jean-Paul Jonz
Mary Ellen Vouz
Philip Richard Carpenter

C. Sample Output

Carpenter, P. R.
Jonz, J. P.
Vouz, M. E.

Assignment #5

Notes:

1. Please sequence your decks before you submit them with the sample output for grading.
2. When you find something that the compiler does not support and you feel like it should be supported, please let me have a copy of the deck, a copy of the output, and an explanation of why you think the compiler should support a particular construction. (e.g. The compiler should support because the text indicates that it is valid ALGOL68.)

3. Certain constructions that were valid according to the old ALGOL68 report are not longer valid. For example THEF was valid according to the old report but the construction starting with THEF is not valid according to the new report.

Problem:
Let p be a polynomial of the form

$$p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_kx^k$$

This polynomial, p, can be represented as

p = (k, a[0], a[1], a[2], . . ., a[k])

Construct a computer program that will accept a polynomial p as input and produce a polynomial q; as output. With p represented as above q is to be computed and represented as

q = (k = 1, a [1], 2*a[2], 3*a[3], . . ., k*a[k])

Problem:
You have a "little black book" that contains the names, addresses and telephone numbers of your friends. For each one you have information about their likes and characteristics. For this problem your black book contains only names of members of the opposite sex, and you keep the black book on computer cards.

1. Make up five ficticious people and rate them on a scale from one to five according to the table (you may add to the table, if you wish) below.

Characteristics	rating
appearance	
compatibility	
conversion	
financial situation	
morals	
scholarship	

Likes	rating
Classical music	
pop music	
movies	
TV	
parties	
sports	
world affairs	

2. Based on the rating scheme design three algorithms to choose a date from your little black book. These should include methods to choose a date for
   a) the homecoming football game and homecoming dance
   b) the FLATLANDER FRAT-RAT BRAWL
   c) a relaxing evening.
3. Write an ALGOL68 program that will accept your little black book as input; and tell you who to call first when you want that date.
   Input:
   1. Ratings tables for each person in your black book
   2. Type of date you want

   Output:
   Name and phone number of the first person you want to call.

Assignment #7

Program 7A:
Construct an ALGOL68 program which will find all three digit positive integers that are equal to the sum of the cubes of their digits.

Program 7B:
Construct an ALGOL68 program which will read a positive integer, N, and print the complete factorization of N (all prime factors). If N is 12 then the output should be 2*2*3.

Program 7C.
Construct an ALGOL68 program which will read a number, its radix, and a new radix; the program should then produce the representation of the number in the new base.

Problem:
Assume that you have the tabular approximation of a function at integral values of the ordinate; such as,

x	y
x_1	$f(x_1) + \varepsilon_1$
x_2	$f(x_2) + \varepsilon_2$
.	.
.	.
.	.
.	.
x_n	$f(x_n) + \varepsilon_n$

For this problem $\{x_1, x_2, x_3, \ldots, x_n\} \in I$. We will represent $f(x_i) + \varepsilon_i$ by y_i; where, $f(x_i)$ is the true function values at x_i and ε_i is the error in our value. (This situation frequently occurs when we are taking data for an experiment.) For this problem we will make the additional assumptions that y_i, $f(x_i) + \varepsilon_i \in R$, and that $x_1 < x_2 < \ldots < x_n$.
For this problem you are to write an ALGOL68 program that will find approximations to f at input points x, $x \notin I$. The approximations are to be made using each of four methods of interpolation. The methods are: 1) linear interpolation, 2) Newton's backward interpolation, 3) Aitken's interpolation*, and 4) Lagrangian interpolation. You can find these methods in any numerical analysis book. In particular, you might want to look at

Grove, Wendell E. Brief numerical methods.
Englewood Cliffs: Prentice-Hall, c1965.

Input
Input for this program should be 1) a positive integer N, which indicates the number of points in the table; 2) a set of N (x,y) pairs which forms the tabular representation of the function; and 3) a set of x value ($x \notin I$), where function is to be evaluated.

Output
Output is to be the tabular representation of the function, a copy of each input x value and the corresponding y value for each interpolation method used.
For example, your output could appear as

ORIGINAL FUNCTION

x	y
0	4.0
1	5.0
2	7.0
3	9.2
4	16.0
5	26.9
6	32.1

*Also called Aitken's process of repeated interpolation.

INTERPOLATED FUNCTION VALUES

x	LINEAR INTERPOLATION	NEWTON'S BACKWARD INTERPOLATION
0.5	4.5	4.7
4.3	17.0	18.0

AITKEN'S INTERPOLATION	LAGRANGIAN INTERPOLATION
4.8	4.4
18.1	17.9

(Note: These values are not necessarily correct.)

Test cases

Make sure your test cases are complete; e.g., What happens where f(x) is undefined? What happens when $x \in I$ and x is in the table? What happens when $x \in I$ and x is not in the table?, etc. Answer each of these questions and any similar questions you can think of about the operation of the program. Don't forget the cases where you have correct input and get correct output. For each case, is the action your program takes reasonable? Why or Why not? If not, how can you correct it?

Documentation

Full and complete documentation is expected.