PROGRAMMING AND COMPUTER SOFTWARE

A translation of *Programmirovanie*

Volume 4, Number 5 September-October, 1978

CONTENTS

	Engl.	Russ.
PROGRAMMING THEORY		
Program Compositions and Composition Programming – V. N. Red'ko	303	3
Formalization of Language – V. A. Tuzov	318	25
Compiler-Writing Technology: A Projection Approach – V. Sh. Kaufman	327	36
Correctness of Graph Schemes of Parallel Algorithms – Yu. P. Korablin	334	45
Parallel Decomposition of Algorithms – A. I. Belousov	340	53
COMPUTER SOFTWARE AND SYSTEM PROGRAMMING		
Algorithms for Real-Time Management of a Distributed Computing System Working in Interactive Mode – A. V. Gavrilov and V. I. Zhiratkov	348	62
SOFTWARE INFORMATION SYSTEMS		
Chain-Progressive Organization of Overflow Records in Randomized-Structure Files – V. A. Litvinov and A. A. Kokorin	353	69
PROGRAMMED SIMULATION AND COMPLEX SYSTEMS		
KIMDS – A Program Suite for Simulating Digital Systems of General Type – Yu. I. Mitrofanov and A. N. Ivanov	357	74
PROCESSING SOFTWARE INFORMATION		
An Algorithm for Organizing Hash Tables – R. M. Akchurin	364	84
BRIEF COMMUNICATIONS		
NEATPL – An Aid to the Debugging of PL/1 Programs – N. N. Bezrukov	366	87
A Text-Data Packing Module – R. G. Khannanov	371	94

The Russian press date (podpisano k pechati) of this issue was 9/28/1978. Publication therefore did not occur prior to this date, but must be assumed to have taken place reasonably soon thereafter.

NEATPL – AN AID TO THE DEBUGGING OF PL/1 PROGRAMS

N. N. Bezrukov

UDC 681.3.06:51

A service program designed for the structured printout of the texts of PL/1 programs and other operations (automatic identifier replacement, control of the listing printout process, translation of compiler diagnostic messages, etc.) is described as an aid to program debugging.

As the volume and complexity of programs grow they become increasingly difficult to comprehend, debug, and modify. Consequently, worldwide efforts are being aimed at devising methods and facilities to aid in the design, debugging, and generally keeping track of complex programs [1]. These methods and facilities may be nominally divided into two groups: general-purpose, i.e., applicable to a fairly broad class of programming languages, and special-purpose, i.e., oriented toward one specific language.

In the present article we describe the NEATPL service program, which is designed to aid in comprehending the logical structure of PL/1 programs. When the disk operating system (DOS) and tape operating system (OS) for computers of the Unified System (ES) used in COMECON countries are employed in conjunction with PL/1 language translators, NEATPL provides a means for representing the standard translator listing in a more compact and workable listing. The program utilizes a number of ideas of Conrow and Smith [2]. However, while the main ideas remain intact, the specific principles of transformation of the source text in NEATPL differ from those used in [2].

The service program has two versions: NEATPL1 and NEATPL2. NEATPL1 is a preprocessor, and the input information for it is the text of the user program. In the course of operation the program reads the source text, transforms it, and transfers the transformed program text into the system listing printer and then onto tape or disk for subsequent translation. The input for NEATPL2 is the translator log, written on tape or disk. NEATPL2 reads the log, transforms it, and transfers the transformed listing into the system printer.

The capabilities of NEATPL for the generation of a listing exhibiting the logical structure of a program are most completely utilized when the use of the GOTO statement is limited in the program, i.e., when the program has a hierarchical structure. Experience has shown that the application of NEATPL encourages the programmer to continually refine the logical structure of his programs.

General Description of Program Capabilities. The primary function of NEATPL is to generate a compact (two columns per list) listing mirroring the logical structure of the user program. Here the user has a number of options for controlling the form of the listing. Moreover, NEATPL can execute certain other ancillary functions facilitating the debugging process. The required operations are specified by inserting into the input program comments of the form

/*: condition — 1 condition— 2...*/,

where condition—1, condition—2, etc. are keywords used to specify conditions.

Following is a concise description of the most important conditions and keywords used to specify them. Keywords used to void a condition are indicated in parentheses. The PROFILE and MACRO conditions refer only to NEATPL1, and the RUSSIAN condition only to NEATPL2.

1. SHIFT = ⟨integer⟩ (SHIFT = 0). If the SHIFT condition is specified, statements entering into a loop are shifted by the designated number of positions to the right relative to statements not belonging to the given loop.

2. COMMENT (NOCOMMENT). If the COMMENT condition is specified, the listing will include all comments existing in the source program. If the NOCOMMENT condition is specified, comments existing in the source program are not included in the listing.

Translated from Programmirovanie, No. 5, pp. 87-93, September-October, 1978. Original article submitted August 1, 1977.

0361-7688/78/0405-0366$07.50

3. PROFILE (NOPROFILE). If the PROFILE condition is specified, instructions for computing the number of executions of each statement of the source text are included in the listing and in the resultant text.

4. MACRO (NOMACRO). Specification of the MACRO condition makes it possible to replace identifiers in the source program text by new identifiers or character strings.

5. RUSSIAN (NORUSSIAN). If the RUSSIAN condition is specified, then in the derivation of each diagnostic message NEATPL2 determines its number, selects the translation text of the message with the given number from the file of diagnostic messages ERMES, and prints the translation text after the English text of the diagnostic message.

6. NEAT (NONEAT). If the NEAT condition is specified, NEATPL prints each statement on a new line. If the NONEAT condition is specified, NEATPL transfers during printout to a new line with the beginning of each (including empty) card or in the event of line overflow.

Like translators, NEATPL permits the first-position symbol to be used for print control. Besides the CTLASA control symbols, NEATPL responds to the symbols ¤, < , and > . The symbol ¤ elicits transfer of a given card into a subheading, the symbol > shifts all subsequent strings by the designated number ("SHIFT") of positions to the right, and the symbol < shifts them SHIFT positions to the left. This makes it possible to form a relief of level numbers in the declaration of structures.

Transformation of Text of Source Program; DO Relief. The following transformations are employed to make the program text more comprehensible.

1. If the first statement on a card does not contain labels, then regardless of the position from which it was entered on the card, it is printed in the position

$$N = 7 + SHIFT * (NEST + 1),$$

where NEST is the nesting depth of the loop including the given statement. If the statement does not enter into a loop NEST = 0. If the first statement on a card contains a label, the label is printed at the beginning of the line.

2. In statements contained on a given card all nonsignificant blanks other than those which facilitate reading of the program are deleted. This operation is applicable to blanks after : and ; as well as before and after ! and &.

3. If the NEAT condition is specified, each new statement is printed on a new line.

These transformations produce in the printout a special relief (see Fig. 1), which we call the DO relief. The DO relief facilitates error search in a program, because it exhibits by and large the logical structure of the program. This feature is attributable to the fact that PL/1 statements entering into the THEN and ELSE parts of an IF statement when there are more than one of them are usually formatted as a DO cluster. Hereinafter we refer to a listing with DO relief for brevity as a relief listing.

Listing Format. An example of a listing generated by NEATPL is given in Fig. 2. Inasmuch as the program displays occupy only one column, the second column is not shown in Fig. 2.

Each column of the listing begins with a heading. As in translators, the first card of the program is transferred into the heading. Below the heading is a subheading, into which is transferred the first of the cards contained in the given column, with the symbol ¤ in the first position. The first three symbols in each line represent the index of the program statement. For NEATPL2 this index always corresponds to the numbering in the PL/1 translator. In the case of NEATPL1 the correspondence does not carry over to cases in which the translator has inserted a ; into the source text in the course of error correction. The next two symbols are set aside for the value of the loop nesting depth. These values correspond to the values generated by the OS ES PL/1 compiler when the NEST option is specified. The card text fills up the remaining symbols. If the number of symbols to fill up a given line exceeds the allowed number and the text of the source card is not used up, then syntactically correct transfer of the text to the next line is executed.

Identifier Substitution in Source Program Text. If the MACRO condition is specified, NEATPL1 has the capability of replacing identifiers of the source program by arbitrary character strings (i.e., of executing an elementary form of macrosubstitution). The syntax of the substitution-specifying statements coincides with the syntax of the assignment of statement of the PL/1 OS ES translator preprocessor:

% < identifier > = arbitrary string ;

```
S N TEST: PROC;
T E /*************************************************************/
M Sn/* Fig. 1. EXAMPLE OF DO RELIEF.                             */
T T /*************************************************************/

1       TEST: PROC;
2       IF...THEN DO;
4 1        DO I=1 BY 1 WHILE(A...A & B...B);
5 2           C...C;
6 2           IF D...D ! F...F THEN DO;
8 3              E...E;
9 3              DO F=1 BY 1;
10 4                G...G;
11 4                IF H...H THEN DO;
13 5                   K...K;
14 5                   DO L=1 BY 1;
15 6                      M...M;
16 6                      END;
17 5                   END;
18 4                END;
19 3             END;
20 2          END;
21 1       END;

22    END TEST;
```

Fig. 1

For example, suppose that the following substitution statements are given:

```
%PAPER= 'SYSPRINT';
%CALL = ' ';
%OUT = 'PUT FILE(PAPER)EDIT(CARD)(A)';
```

and the statement CALL OUT occurs in the program text; then this statement is replaced by the string

```
PUT FILE(SYSPRINT)EDIT(CARD)(A);
```

Thus, it is possible to replace calls for procedures without parameters by a linear code for the purpose of program optimization. We highlight the following in the set of other possible macrosubstitution applications:

1. It is often necessary to replace a variable identifier by another that more accurately exhibits its function or contains fewer symbols. The situation arises when in the course of development of a program its size becomes such that it has to be separated into two or more modules. Inevitably in this case a number of variables must acquire the attribute EXTERNAL. However, the length of external-variable identifiers must not exceed 6 (DOS) or 7 (OS) symbols.

2. Sometimes a program will use constants that are subject to frequent changes. Of course, these constants can be replaced by variables, but in a number of situations it is impossible to do so because of efficiency considerations or translator limitations. In this event the macrocapabilities of NEATPL can be used to perform additional program parametrization.

3. A built-in stenographic notational system can be created for structures that recur frequently in programs. Suppose, for example, that the statement

```
PUT FILE(OUT)SKIP EDIT...
```

recurs frequently in a program. Then by specifying the substitution statement %EXTRACT = 'PUT FILE(OUT) SKIP EDIT'; we can write in the program EXTRACT('REPORT')(A); After NEATPL processing we obtain

```
PUT FILE(OUT)SKIP EDIT ('REPORT') (A);
```

```
S N PRIMES: PROC OPTIONS(MAIN);
T E /*****************************************************/
M Sn/* PROGRAM PRINTS FIRST 100 PRIMES                   */
T T /*****************************************************/

1       PRIMES: PROC OPTIONS(MAIN);
      /* PROGRAM PRINTS FIRST 100 PRIMES       */
      /* IN 5 COLUMNS 20 LINES EACH COLUMN     */
2     DCL PRIME(100)/* ARRAY OF PRIMES        */
        FIXED BINARY INITIAL(2,3),
        LAST/* INDEX OF LAST PRIME IN 'PRIME'  */
        INITIAL(2),
        CUR/* CURRENT NUMBER TESTED */ FIXED BINARY,
        #STR/* NO. OF NUMBERS IN COLUMN */ FIXED BIN INIT(20);
3       DO CUR=5 BY 2 WHILE(LAST<=100);
4 1       DO K=1 TO LAST WHILE(MOD(CUR,PRIME(K))-=0);
5 2         END;
6 1       IF K>LAST THEN DO;
8 2         LAST=LAST+1; PRIME(LAST)=CUR;
10 2        END;
11 1      END;
12      PUT SKIP EDIT('FIRST 100 PRIMES') (A);
13      DO K=1 TO #STR;
14 1      PUT SKIP;
15 1      DO L=0 BY #STR TO LAST-#STR;
16 2        PUT EDIT(PRIME(K+L))(F(10));
17 2        END;
18 1      END;
19    END PRIMES;
```

Fig. 2

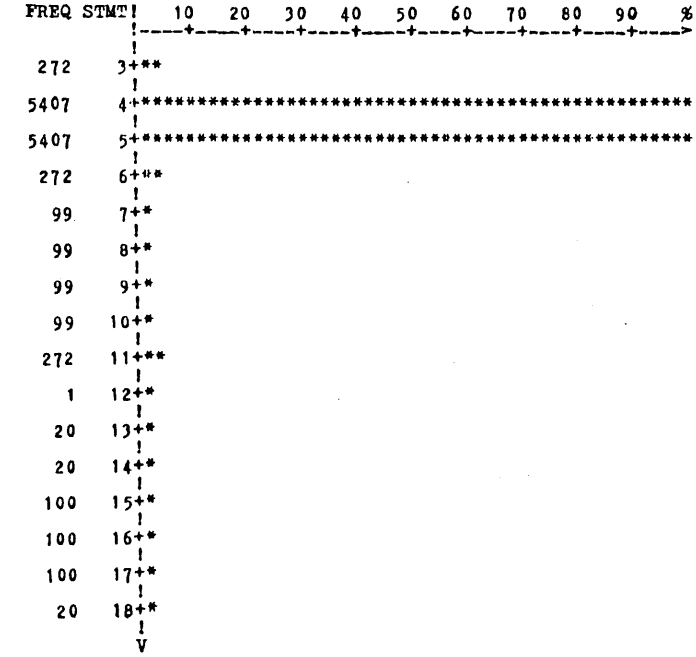

Fig. 3

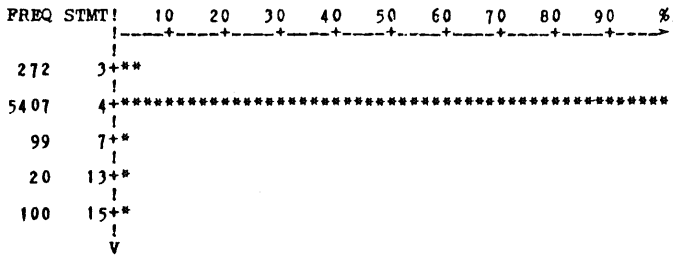

Fig. 4

Generation of Program Profile. The profile of a program is a function whose argument is the index of a statement in the program and whose value is the number of executions of the given statement from the start of the program to the point in time at which the profile is generated. Profiles obtained at different points of the program and for different source data sets contain important information about the behavior of the program.

Recently there have appeared a number of automatic systems that either insert counters at necessary locations in a program or interrupt program execution at definite time intervals in order to determine the most frequently executed part of the program [4]. NEATPL implements a simple system of the first type. For the generation of a program profile it is required to specify the PROFILE condition and to process the source program through NEATPL. In this case counters are inserted only after DO, BEGIN, and PROCEDURE statements and at points of the source text where comments of the form /*%...*/ occur. A statement calling the profile-printout routine is generated at locations of the source program where comments of the form /*?...*/ occur.

Since the profile obtained by means of NEATPL most completely mirrors the course of execution of programs not containing GOTO statements, we call it a structure profile. Its analysis greatly facilitates program optimization. It is important to bear in mind, however, that in PL/1 different types of statements differ considerably in execution time and the same statement is executed in different times when different types of data are used. The profile enables the programmer to concentrate his attention on the most frequently executed parts of his program. The results of an analysis of FORTRAN programs have shown that if the computing time greatly exceeds the input–output time, then for the majority of programs ≈3% of the codes use almost 50% of the machine time [4]. Therefore, for programs of this type it suffices to work on improving precisely that critical 3% in order to achieve a significant speedup of the computations. Consequently, even such a primitive profile as that generated by NEATPL will make it possible to conserve appreciable time and effort for more useful activities.

Figure 3 gives a profile of the PRIMES program, and Fig. 4 gives the structure profile of the same program. The profile is useful not only for optimizing internal programs, but also for the checkout and analysis of outside programs. In checkout the profile indicates which branches of a program have not been tested in a given text. For the analysis of outside programs the profile aids in isolating important parts of the program from nonessential parts.

Conclusion. NEATPL can be used, beginning with the early stages of program development. Experience has shown that without NEATPL the search for logical errors in a program usually begins with runs on test data. With the use of NEATPL this type of error is identified in the acquisition of an unanticipated DO relief structure. This process does not require a syntactically correct program, and so errors of this type can be localized and corrected before the first test-data run.

The programmer is aided considerably in the checkout phase by the program profile, the analysis of which often enables him to locate untested program branches. When development, debugging, and checkout have been completed, a new deck with the DO relief fixed in it can be obtained by means of NEATPL. Here it is possible to replace unfavorable identifiers.

Despite the fact that NEATPL is designed mainly for the processing of source text prior to translation or translation logging, it can serve a highly auspicious auxiliary function as a documentation facility, because a person seeing the program for the first time can look at the logical structure of the program without worrying about constructing a flowchart. The relief listing combined with the program profile can provide considerable aid in comprehending outside programs.

The time for NEATPL to process a program text or translation log amounts to only 5-10% of the translation time. Consequently, the application of NEATPL yields only a slight increase in the program running time.

The NEATPL program has been in operation since April of 1976, and the experience gathered in that period has demonstrated its considerable effectiveness for the debugging of programs of great complexity and size. In particular, since NEATPL is written in PL/1, after creation of the first version the entire remaining debugging operation was carried out with the use of the relief listing.

In connection with the teaching of programming in colleges and universities NEATPL can serve as an error-locating aid to both students and instructors. On the one hand, students can use the relief listing to locate more errors on their own. On the other hand, errors that the students are not skilled enough to locate can be found more rapidly by the instructor with access to the relief listing of the student program. NEATPL reliably protects the instructor against carelessly written student programs by converting them into accurate listings with a clearly perceptible logical structure. The application of NEATPL in computing centers, besides easing the work of programmers, in our opinion, will provide programmers with deeper insight into the programs of their colleagues.

LITERATURE CITED

1. Large-System Debugging Techniques [in Russian], Statistika, Moscow (1977).
2. K. Conrow and R. G. Smith, "NEATER2: a PL/1 source statement reformatter," Commun. ACM, 13, No. 11 (1970).
3. D. Knuth, "Structured programming with go to statements," Comput. Surv., 6, No. 6 (1974).
4. D. Knuth, "An empirical study of FORTRAN programs," Software — Prac. Exper., 1 (1971).

A TEXT-DATA PACKING MODULE

R. G. Khannanov

UDC 658.012.011.56:681.3.06

A brief description is given of a family of modules for packing text data into graphs of appropriate size while observing appropriate grammatical-association rules.

The UKLAD module is used for packing text data into a graph of appropriate size in accordance with grammatical rules; the module splits up the text into appropriate elements on the basis of formal rules that do not involve the semantics of the words or the component parts. In that sense, the algorithm is essentially equivalent to that of [1], but there are two major differences.

1. A word delimiter may be not only a single space but also any combination of spaces, as well as a full stop, comma, semicolon, colon, slash, or dash; this means that texts containing common typing errors are acceptable, e.g., the omission of a space after a comma or the insertion of more than one space between words. If required, the user can compile his own table of delimiters.

2. The module inhibits hyphenation not only of ST and SK but also of any arbitrarily specified set of diphthongs and/or triphthongs.

Other styles of the module have been developed for packing foreign-language texts (English, German, French, and Spanish). The modules have been written in ES assembler, which means they can be used with the IBM/360 software, and also with systems of similar architecture (ASVT, S-4, SIEMENS, etc.). Also, a similar module has been written in the input language of the Minsk-32 (the UKLAD module).

LITERATURE CITED

1. V. I. Abramov and G. D. Frolov, "Computer-aided editing," Tsifrov. Vychisl. Tekh. Programmir., No. 6 (1971).

Translated from Programmirovanie, No. 5, p. 94, September-October, 1978. Original article submitted July 6, 1978.

0361-7688/78/0405-0371$07.50