**Commission of the European Communities**

**Joint Research Centre - Ispra**

# Computing Centre Newsletter



**July 1977 ● No 13**

# Contents


Note of the Editor 2

The Operating System OS/MVT: An Overview 3

Statistics on computer utilization: June 12

Utilization by objectives and accounts 13

Table of equivalent time, summary per month and cumulative 15

Note to the Users 20

Inquiry 21

## Note of the Editor

The present Newsletter is published monthly except for August and December.

The Newsletter includes:

- Developments, changes, uses of installations
- Announcements, news and abstracts on initiatives and accomplishments.

The Editor thanks in advance those who want to contribute to the Newsletter by sending articles in English or French to one of the following persons of the Editorial Board.

## Note de la Rédaction

Le présent Bulletin est publié mensuellement excepté durant les mois d'août et décembre.

Le Bulletin traite des:

- Développements, changements et emploi des des installations
- Avis, nouvelles et résumés concernant les initiatives et les réalisations.

La Rédaction remercie d'avance ceux qui veulent bien contribuer au Bulletin en envoyant des articles en anglais ou français à l'un des membres du Comité de Rédaction.

## Editorial Board / Comité de Rédaction



## Computing Centre References

| | | Room | Tel. |
|---|---|---|---|
| *Manager* | J. Pire | 1816 | 732 |
| Adjoined | G. Gaggero | 1874 | 787 |
| *Computer Room* | P. Tomba | 1857 | 797 |
| Adjoined | A. Binda | 1857 | 797 |
| *Peripherals* | G. Nocera | 1825 | 767 |
| *System Group* | D. Koenig | 1839 | 742 |
| Adjoined | P.A. Moinil | 1841 | 704 |
| *Informatics Support* | G. Gaggero | 1874 | 787 |
| o General Information | G. Hudry | 1873 | 787 |
| o Program Information Service | G. Gaggero | 1874 | 787 |
| Adjoined | S. Leo Menardi | 1884 | 721 |
| o Graphics and Support to Users | H.I. de Wolde | 1890 | 753 |
| Adjoined | A. Pollicini | 1882 | 743 |
| Application Packages | A. Inzaghi | 1887 | 755 |
| Programming Languages | C. van den Muyzenberg | 1848 | 781 |

Editor : Sylvia R. Gabbai
Layout : Paul De Hoe
Graphical and Printing Workshop, JRC Ispra

## The Operating System OS/MVT : An Overview

*D. König, P. Moinil*

### Introduction

The following article presents excerpts of a number of manuals in the references which the interested reader may consult for further details.

The actual operating system running on the IBM 370/165-Model 1 of the computing centre is OS/MVT release 21.8 in conjunction with the HASP-II system, release 3.1 (see 1). The release 21.8 of OS is the most recent release and there will be no more future releases of OS/MVT from IBM. The actual operating system was introduced at the end of the last year to replace the operating system OS/MFT for mainly two reasons:

1. to be able to offer to the users of the computing centre more and better services (for instance an advanced time-sharing service)
2. to facilitate the operating of the enlarged configuration (i.e. three megabytes of core storage).

Since the more recently developed IBM operating systems utilizing virtual storage techniques for name space management cannot run on an IBM 370/165 Model 1 CPU OS/MVT 21.8 will be the final version of the operating system for the actual machine.

The operating system consists of a *control program* and a number of *processing programs.* The elements of the control program are

- job management
- task management
- data management
- recovery management

1) HASP – Houston Automatic Spooling Priority system

They will be described in the following section. The elements of the processing programs are

- languages (such as ALGOL, Assembler, COBOL, FORTRAN, PL/1,...)and
- service programs (such as utilities, linkage editor, emulators, etc.).

They will not be described in the sequel. Descriptions can be found in the appropriate manuals listed and described in ref. 6.

**Functions of the OS/MVT Control Program**

*Job management*

The job management programs control the job flow through the systems and all operator communications. In general, they do the following:

*Analysis of the input stream:* scanning the input data to indentify control statements; analysing and interpreting control statements; preparing the necessary control tables that describe each job to the system.

*Allocation of I/O devices:* ensuring that all necessary I/O devices are allocated; ensuring that direct access storage space is allocated as required; ensuring that the operator has mounted any required tape and direct access volume.

*Overall scheduling:* selecting jobs for execution on a priority basis.

*Transcription* of input data onto, and user output from, a direct access device.

*Communication* between the operator and the system.

The functioning of the MVT priority scheduling system and the different parts of the control program involved therein are depicted in Fig. 1.

*Task management*

The difference between a program and a task is that a program is a sequence of instructions whereas a task is the work to be done by the execution of a program. The task management programs supervise the execution of all work done in the system. In the multiprogramming environment they control the allocation and use of CPU, main storage and programming resources among competing tasks on a job class and priority basis. They receive a job step (each job step is executed as a task) from the

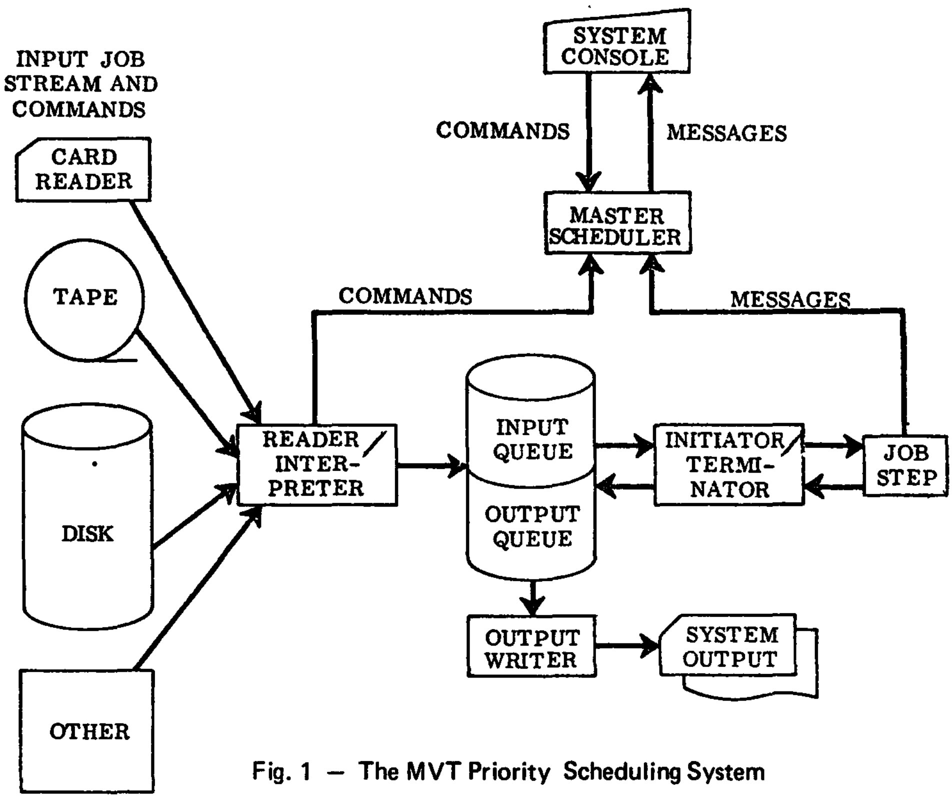


Fig. 1 – The MVT Priority Scheduling System

1. Your programs, defined as job steps by the job control language, enter the system through an input stream from some input device.
2. The reader/interpreter reads in control statements for one or *more* jobs and places them on the input work queue. The input on each queue is arranged by job class, the initiation of a job within a queue is determined by the priority within the class.
3. The job with the highest priority is selected for execution by the initiator/terminator.
4. The initiator/terminator turns your job step over to the task management programs, which supervise its execution.
5. The master scheduler accepts and takes action on commands.
6. Output is written (by job priority) when the job has terminated and while other jobs are being processed, if output data sets are being processed by the system output writer.

job scheduler which initiates a task (see Fig. 1). The functions of the task management programs are:

• Interruption supervision. The analysis of interruptions to determine what supervisor processing is required.

• Main storage supervision. The allocating and freeing of main storage, and recording of what use is being made of any portion of main storage.

• Timer supervision. The setting and maintaining of the interval timer from information provided in timer macro instructions.

• Contents supervision. The loading of programs into storage, the recording of what programs are currently in main storage, and what characteristics these programs possess.

• Task supervision. The recording of what tasks are currently in the system, their status, priorities, the programs they require, and the order in which these tasks have to be performed.

Each system resource is controlled by a different part of the control program, the so-called resource manager, and the allocation of the resources to competing requests is done by enqueuing the requests and servicing one at the time. Tasks which cannot proceed further because their resource request cannot be handled immediatly are placed into a wait-state and the completion of their request is posted to them. A more detailed description of the resource allocation and the synchronization primitives can be found in ref. 1.

*Data management*

The data management programs are primarily responsible for moving information between main storage and external storage and maintaining it in external storage. They are capable of locating data, preparing main storage areas for it, reading it, and writing it. Their major functions are

– assign and release space on direct access volumes
– maintain the catalog
– perform the I/O support (open, close, end-of-volume) processing
– process I/O operations.

Data is stored on external storage in data sets (named collection of data whose extent (physical boundaries) is known to the system) which reside on volumes (reel of tape, disk pack, drum). The data in a data set can be organized sequential, indexed sequential, direct or partitioned. A partitioned data set consists of independent groups of sequentially organized data sets each identified by a member name. A partitioned data set is often referred to as a library. More details on data organization and access methods can be found in the references.

Usually the data sets are referred to by their names and in order to find a data set the system maintains a catalog of data set names and associated control blocks on a direct access device. This system catalog resides on the system residence volume and all searches start here. In the general case of a non-simple data set name there exist a hierarchy of pointers which have to be followed in the search procedure. An example of the search procedure which has to be followed to find a data set with the name TREE.FRUIT.APPLE is given in Fig. 2 (see next page).

More details on the labelling conventions of direct access volumes, on the organization of the volume table of content (VTOC) and on the management of volumes in the MVT system can be found in ref. 1.

*Recovery management*

When a machine malfunction occurs, recovery management routines record critical machine and program data, and (in some cases) attempt to recover from the error.. Depending on the specific routine and type of error, recovery takes place at one of four levels:

*Functional recovery*
: resumption of the task at the point where the error occurred. Machine or recovery management facilities correct storage errors, retry unsuccessful instructions and I/O operations.

*System recovery*
: termination of the task affected by the error, permitting system operation to continue.

*System-supported restart*
: re-IPL (initial program loading) using system restart facilities.

*System repair*
: total system halt for manual repairs, aided by recovery management records.

More details about the different functions of the recovery management routines can be found in ref. 3.

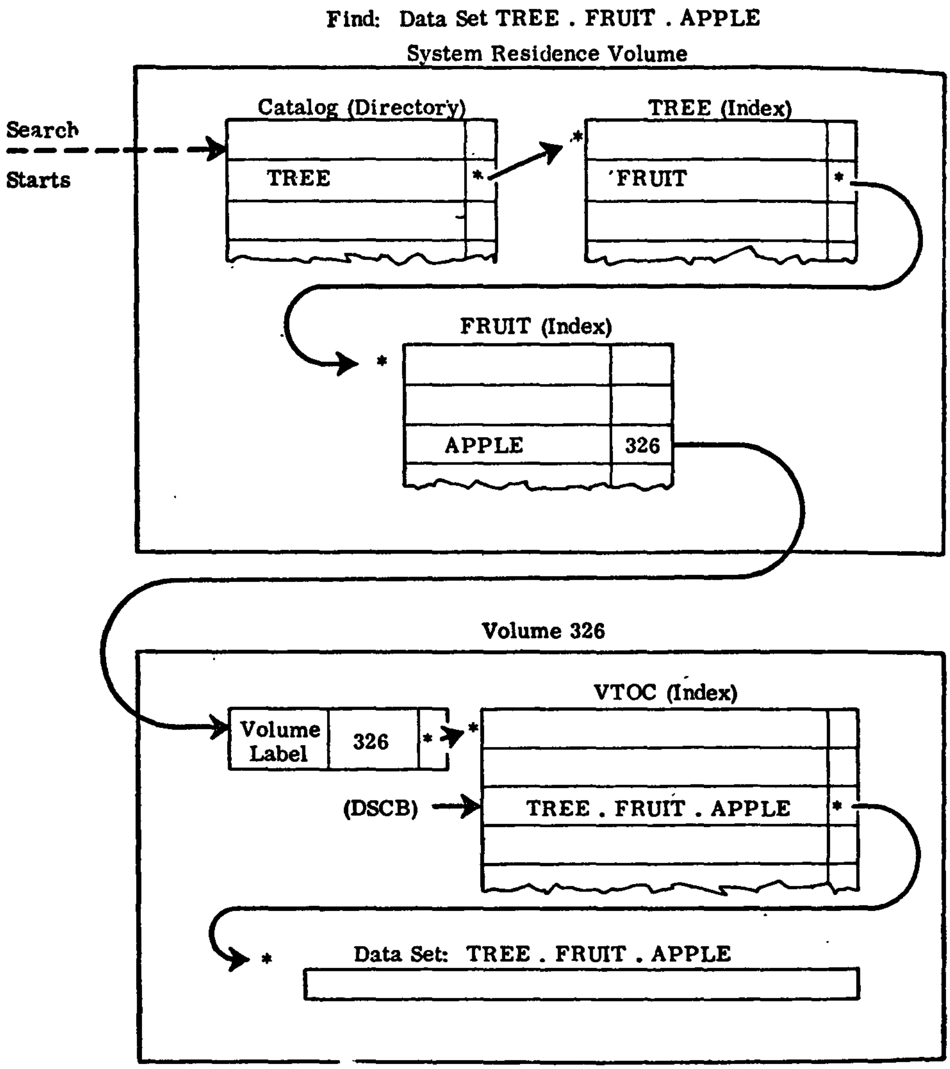


Fig. 2 — Catalog search procedure

*Summary*

In Fig. 3 the relations of the four functions of MVT are depicted. It should, however, be understood that this is a superficial picture and that simplifications tend to contain errors if the details are considered.

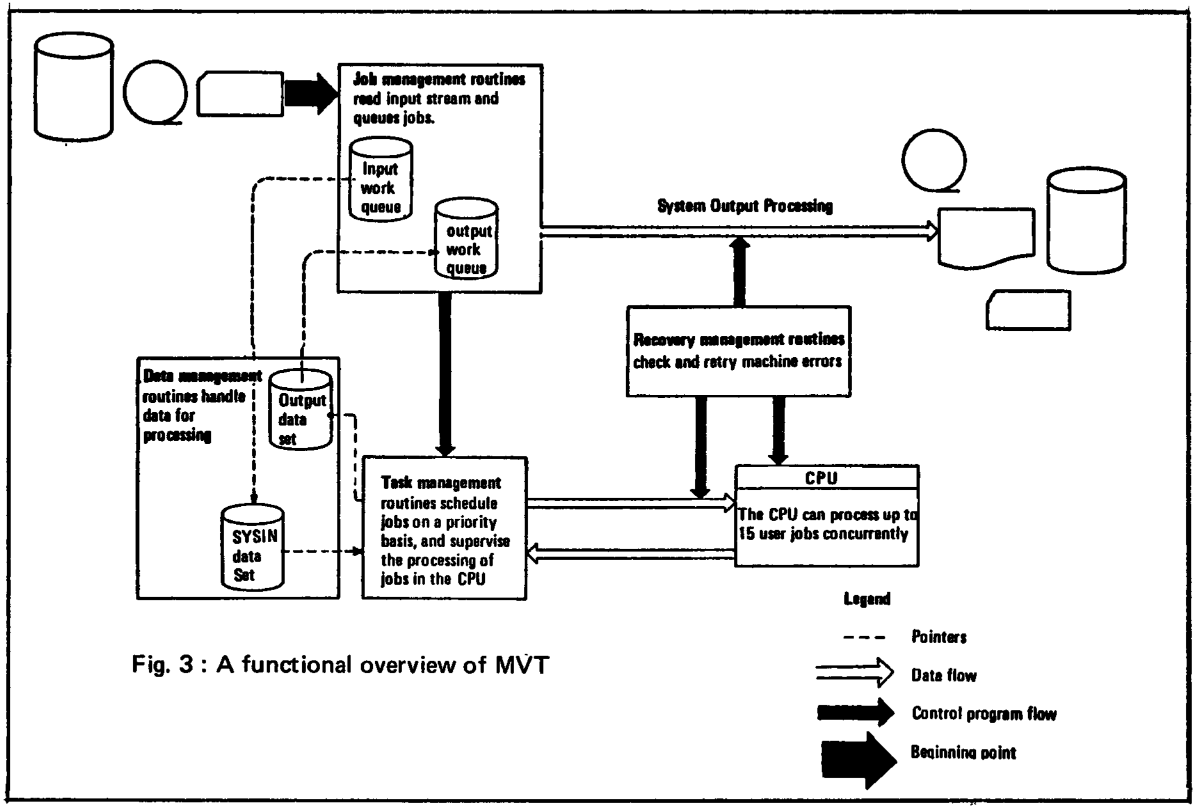


Fig. 3 : A functional overview of MVT

**The Functions of HASP**

The HASP-system operates a compatible extension to the MVT operating system to provide specialized supplementary support in the areas of job management, data management, and task management. HASP appears as a transparent "front-end" processor to OS. Via its SPOOLing functions – they are normally associatedwith OS input readers and output writers (see Fig. 1) – it acts as an automatic scheduler and operator of OS and performs the peripheral functions associated with batch job processing.

HASP has four major processing stages which account for its four major external functions:

*Input stage:* This stage reads jobs simultaneously from an essentially unlimited number of various types of on-line card readers, tapes and remote terminals into the system. These jobs are then entered into a priority queue by job class to await processing by the next stage.

*Execution stage:* This stage takes jobs on priority bases from the queue established by the input stage and passes them to OS/MVT for processing. Input cards are supplied to the executing program as required and print and punch records are received and written onto HASP intermediate storage (spooled).

*Print stage:* The purpose of this stage is to transcribe the printed output from the spool to real printers. An essentially unlimited number of various types of printers and remote terminals can be operated simultaneously.

*Punch stage.* This stage transcribes the punch records from the spool to real card punches. An essentially unlimited number of various types of punches and remote terminals can be operated simultaneously.

Since all of the above functions can occur simultaneously and asynchronously, a continuous flow of jobs may pass through the system.

Summarizing, the use of HASP offers the following advantages:

- Improved performance – Any improvement is of course dependent upon the configuration and job mix and can only be determined by actual measurement.

- Improved operational procedures – HASP acts as an automatic interface between the operator and OS/MVT, to perform various OS control functions which have to be done by the operator directly if HASP is not used. OS-Readers, Writers and Initiators for instance are started and scheduled automatically by HASP.

– Increased system function – The use of HASP provides functions which are not otherwise available. These include:

dynamic task ordering based on CPU-I/O characteristics

the inclusion of relevant console messages in each job's output;

the capability of any job to introduce another job into the HASP queue via an internal reader;

many additional operational control functions and various other functional enhancements (see ref. 2)

– High-performance Remote Job Entry (RJE) – The support for Binary-synchronous CPU-workstations employs an advanced technique called MULTI-LEAVING which provides for simultaneous operation of all devices on a remote workstation.

All these functions are accomplished via transparent operation to both the operating system and to the user program. This means neither the operating system nor the user program has to be changed to benefit from the above mentioned advantages.

**References**


1) IBM Operating System /360 Concepts and Facilities (excerpts) in: S. Rosen (Ed.) Programming Systems and Languages, pp. 598-646 McGraw-Hill, 1967

2) The HASP System, February 26, 1977 – Houston Automatic Spooling Priority, IBM Contributed Program Library

IBM Manuals

3) IBM System/360 Operating System: Concepts and Facilities, GC28-6535-X

4) IBM System/360 Operating System. Introduction. GC28-6534-X

5) IBM System/360 Operating System: MVT Guide. GC28-6720-X

6) IBM System/360 and System/370 Bibliography. GA22-6822-X



*The Newsletter is available at:*

**Mrs. A. Cambon**
Support to Computing
Bldg. 36 - Tel. 730

*Des exemplaires du Bulletin sont disponibles chez:*

**Mme A. Cambon**
Support to Computing
Bât. 36 - Tel. 730

## Statistics of computing installation utilization

### Report of computing installation exploitation for the month of June

| | YEAR 1977 | YEAR 1976 |
|---|---|---|
| Number of working days | 22 d | 18 d |
| Work hours from 8.00 to 24.00foı | 16.00 h | 16.00 h |
| Duration of scheduled maintenance | 17.84 h | 24.17 h |
| Duration of unexpected maintenance | 14.61 h | 3.83 h |
| Total maintenance time | 32.45 h | 28.00 h |
| Total exploitation time | 319.55 h | 262.00 h |
| CPU time in problem mode | 143.71 h | 128.86 h |
| **Teleprocessing:** | | |
| CPU time | 3.93 h | 1.18 h |
| I/O number | 854,000 | 218,000 |
| Equivalent time | 9.85 h | 2.71 h |
| Elapsed time | 351 h | 133 h |
| **Batch processing:** | | |
| Number of jobs | 10,392 | 8,056 |
| Number of cards read | 2,895,000 | 2,614,000 |
| Number of cards punched | 166,000 | 177,000 |
| Number of lines printed | 28,265,000 | 23,617,000 |
| Number of pages printed | 6,321,000 | 524,000 |

BATCH PROCESSING DISTRIBUTION BY REQUESTED CORE MEMORY SIZE

| | 100 | 200 | 300 | 400 | 600 | 800 | 1000 | 1400 | total |
|---|---|---|---|---|---|---|---|---|---|
| Number of jobs | 2443 | 3833 | 2048 | 1114 | 451 | 77 | 5 | 34 | 10005 |
| Elapsed time (hrs) | 56 | 174 | 198 | 149 | 150 | 34 | 0.7 | 4 | 766 |
| CPU time (hrs) | 3.7 | 19 | 33 | 24 | 47 | 11 | 0.2 | 1 | 130 |
| Equivalent time (hrs) | 17 | 51 | 66 | 64 | 64 | 16 | 0.5 | 1.6 | 280 |
| Turn around time (hrs) | 0.4 | 0.7 | 1.3 | 2.5 | 2.2 | 2.9 | 1.9 | 2.1 | 1.1 |

PERCENTAGE OF JOBS FINISHED IN LESS THAN

| TIME | 15' | 30' | 1h | 2h | 4h | 8h | 1D | 2D | 3D | 6D |
|---|---|---|---|---|---|---|---|---|---|---|
| % year 1976 | 41 | 58 | 73 | 86 | 95 | 98 | 99 | 99 | 99 | 100 |
| % year 1977 | 46 | 64 | 80 | 90 | 97 | 99 | 99 | 99 | 100 | |

**Utilisation of computer center by the objectives and appropriation accounts for the month of June**

| | | IBM 370/165 equivalent time in hours |
|---|---|---|
| 1.20.2 | General Services - Administration - Ispra | 34.85 |
| 1.20.3 | General Services - Technical - Ispra | 0.97 |
| 1.90.0 | ESSOR | 5.59 |
| 1.92.0 | Support to the Commission | 7.32 |
| 2.10.1 | Reactor Safety | 133.89 |
| 2.10.2 | Plutonium Fuel and Actinide Research | 6.73 |
| 2.10.3 | Nuclear Materials | 4.88 |
| 2.20.1 | Solar Energy | 0.68 |
| 2.20.2 | Hydrogen | 0.05 |
| 2.20.4 | Design Studies on Thermonuclear Fusion | 1.49 |
| 2.30.0 | Environment and Resources | 12.18 |
| 2.40.0 | METRE | 6.63 |
| 2.50.1 | Data Processing | 53.37 |
| 2.50.3 | Safeguards | 0.13 |
| | TOTAL | 268.76 |
| 1.94.0 | Services to external Users | 21.89 |
| | **TOTAL** | **290.65** |

**EQUIVALENT TIME TABLE FOR ALL JOBS OF THE GENERAL SERVICES - Monthly and Cumulative Statistics**

| | January | February | March | April | May | June | July | August | September | October | November | December |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year 1976 | 84 | 82 | 101 | 77 | 57 | 64 | 73 | 54 | 61 | 59 | 36 | 46 |
| accumulation | 84 | 166 | 267 | 344 | 401 | 465 | 538 | 592 | 653 | 712 | 748 | 794 |
| Year 1977 | 44 | 74 | 78 | 32 | 26 | 36 | | | | | | |
| accumulation | 44 | 118 | 196 | 228 | 254 | 290 | | | | | | |

**EQUIVALENT TIME TABLE FOR THE JOBS OF ALL THE OBJECTIVES AND GENERAL SERVICES - Monthly and Cumulative Statistics**

| | January | February | March | April | May | June | July | August | September | October | November | December |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year 1976 | 206 | 237 | 270 | 241 | 229 | 248 | 249 | 223 | 233 | 244 | 159 | 150 |
| accumulation | 206 | 443 | 713 | 954 | 1183 | 1431 | 1680 | 1903 | 2136 | 2380 | 2539 | 1689 |
| Year 1977 | 135 | 218 | 312 | 193 | 180 | 269 | | | | | | |
| accumulation | 135 | 353 | 665 | 858 | 1038 | 1307 | | | | | | |

**EQUIVALENT TIME TABLE FOR THE JOBS OF THE EXTERNAL USERS - Monthly and Cumulative Statistics**

| | January | February | March | April | May | June | July | August | September | October | November | December |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year 1976 | 18 | 19 | 28 | 16 | 25 | 32 | 14 | 11 | 27 | 31 | 29 | 12 |
| accumulation | 18 | 37 | 65 | 81 | 106 | 138 | 152 | 163 | 190 | 221 | 250 | 262 |
| Year 1977 | 13 | 14 | 18 | 16 | 13 | 22 | | | | | | |
| accumulation | 13 | 27 | 45 | 61 | 74 | 96 | | | | | | |

**EQUIVALENT TIME TABLE FOR ALL JOBS OF ALL USERS - Monthly and Cumulative Statistics**

| | January | February | March | April | May | June | July | August | September | October | November | December |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Year 1976 | 233 | 271 | 313 | 280 | 277 | 281 | 260 | 245 | 273 | 287 | 206 | 172 |
| accumulation | 233 | 504 | 817 | 1097 | 1374 | 1655 | 1915 | 2160 | 2433 | 2720 | 1926 | 3098 |
| Year 1977 | 158 | 241 | 314 | 242 | 202 | 294 | | | | | | |
| accumulation | 158 | 399 | 713 | 955 | 1157 | 1451 | | | | | | |

# Testing by Assertions

*H. Fangmeyer, K. Hanke, C.L. van den Muyzenberg*

## Introduction

"Correctness" is an evermore requested property a computer program should possess. Different approaches to prove correctness have been developed during the past but each of them demonstrate the complexity of the problem rather than give a solution.

There is a common feeling that correctness can only be proved if a precise specification of what the program should do is given.

On the other hand program correctness may even not exist per se since a program can only be considered together with its input space which often is not well defined.

There are algorithms e.g. to invert matrices which work only well when the input matrices are not "ill-conditioned". But ill-conditioning is a very vague property which can't always be easily recognized. Thus, a program may be correct but doesn't work for some input data.

The activity "Software Engineering" concentrates on these problems and it is hoped that some valuable contribution can be given during the course of the actual pluriannual research program.

This paper presents an easy to realize testing aid (consisting of macro instructions) using assertions for PL/1 programmers. The approach is a first step versus a more complex solution which aims at using the predicate calculus as a specification language. Problems of consistency and completeness are not yet considered.

Assertions can be introduced everywhere in the program. When an assertion is encountered its logical expression is evaluated and in respect to its value an action can be chosen by the programmer.

Assertions should be inserted after each logical section of the program to test if the result of the calculus corresponds to the specification expressed by a logical predicate calculus expression.

**Introduction to the Predicate Calculus**

It is well known, that a predicate is a kind of function whose values are propositions. Therefore, functional notations are used in the predicate calculus except in cases where some other notation is in common usage. The predicate $a < b$ could also be written as $L(a, b)$ in which L expresses the predicate "is less than". The variables of the function are called objects or individuals. The range of the objects for which the predicate is valid is called their domain. To express the fact that a predicate is valid for all objects of the domain the notation

$$\forall i\ (L(a_i, b))$$

is used and read :

"For all objects $a_i$ of the previously defined domain A, $a_i$ is less than b"

Likewise the symbol $\exists$ is used to express the existence of at least one object out of a domain which has the predicate stated. Thus

$$\exists i\ (L(a_i, b)$$

is read :

"There exists at least one object $a_i$ in the domain A which is less than b".

The validity range of the quantifiers $\forall$ and $\exists$ is defined by parentheses.

The whole predicate formula is assigned a proposition (FALSE/TRUE). In the first example the formula has the value 'TRUE' only when the predicate is satisfied for all objects $a_i$ otherwise FALSE. The formula of the second example has the value TRUE if one object $a_i$ can be found in the domain A so that the predicate is TRUE otherwise it is FALSE.

To complete, a predicate formula is inductively defined as follows:

A predicate is a formula;

If A and B are formulae then $(A) \subset (B)$*, (A) & (B), (A) v (B) and $\neg(A)$ are formulae;

If it is a variable and A(i) is a formula, then $\forall i\ (A(i))$ and $\exists i\ (A(i))$ are formulas.

These are the only formulae of the predicate calculus.

*) *The implication "$\subset$" cannot be expressed directly in the macro language presented here below.*

**A Testing Aid**

The predicate calculus is already widely used in higher level programming languages. It is here introduced as a testing aid.

A series of macros has been developed and included in the module called ASSERT (see figure 1).

ASSERT evaluates the predicate formula (p.f.) given as argument and assigns a proposition (TRUE or FALSE) to a system inherent logical variable SLV. The same proposition can also be assigned to a logical variable (second argument), if present, in order to be used in subsequent ASSERT-statements as part of a predicate formula.

Ex.

```
ASSERT(A > B & B¬= 0, LX);
```

The predicate formula A > B & B¬= 0 is evaluated and its proposition assigned to SLV and to the user defined logical variable LX.

By the use of ALLDO and EXDO full predicate formulae can be expressed.

ALLDO stands for the universal quantifier ($\forall$) while EXDO represents the existential quantifier ($\exists$). The range of validity of a quantifier is determined by ENDAX.

Ex. The predicate formula

$$\forall i\ (\exists j\ (AR(i,j) = -1))$$

postulates that in each row of the array AR is at least one column with an element having a value of -1.

In the macro-language this is written as follows:

```
ALLDO (I = N1 TO N2);
EXDO (J = M1 TO M2);
ASSERT(AR(I, J) = -1);
ENDAX;
ENDAX;
```

(N1, N2 and M1, M2 being the limits of the variable I and J respectively).

As can readily be seen, ALLDO and EXDO open a parenthesis while ENDAX closes the innermost still open parenthesis.

The system variable SLV is assigned TRUE, if the predicate formula is satisfied otherwise FALSE.

*N.B.* The analysis of the predicate formula is stopped as soon as one row is found for which no element has the value -1.

IFTRUE and IFFALSE are two macros which act on the logical system variable SLV. They can be used to take specific actions in respect to its value. Since there is only one SLV these macros refer to the last active ASSERT.

Generally these two macros will be used to print specific messages.

Ex.

```
ALLDO(I = 1 TO 100);
ASSERT(V(I) = 0);
ENDAX;
IFFALSE(PUT DATA (V(I)); PUT('xxxERRORxxx');STOP;);
IFTRUE (PUT('ALL ZERO') SKIP(2););
```

*N.B.* In order to maintain the logic, IFFALSE and IFTRUE must follow the ENDAX macro instruction of the sub-formula to which it belongs.

Finally there are some convenient macros which render the tool flexible:

ACTASSERT(O) indicates that all assertions will be suppressed.

ACTASSERT(1) indicates that all assertions are included into the source program (default)

ACTASSERT(2) indicates that the assertions will be included as comments

The SETASSERT macro will only be expanded if ACTASSERT(1).

SETASSERT(0) the proposition of the p.f. is assigned to the system variable SLV

SETASSERT(1) the proposition of the p.f. is assigned to the system variable and a standard error-message is generated if SLV = FALSE (default).

SETASSERT(2) the proposition of the p.f. is assigned to the system variable, a standard error message is printed if SLV=FALSE and a 'PUT DATA'-statement is generated.

SETASSERT(3) is similar to SETASSERT(2); after the 'PUT DATA' -statement a 'STOP'-statement is generated.

SETASSERT(4) similar to SETASSERT(1); after the error message a 'STOP'-statement is generated

Further useful macro-functions to facilitate the representation of p.f. are available:

| | |
|---|---|
| SUBRANGE | checks the validity of an index value; |
| TRUE· | generates the proposition TRUE |
| FALSE | generates the proposition FALSE |
| ONLY | a function which operates on logical arrays. |

Interested users may contact the authors for further details on these functions.

The only restriction of the macro-language is that not more than 10 nested ALLDO/EXDO-statements can be coded.

The module ASSERT can easily be accessed by using the following JCL-statements:

```
//  EXEC  PLPCLGS
//CMP.SYSIN  DD  *
*PROCESS M;              )
  %INCLUDE ASSERT;       }  This part must be repeated for every
    <procedure>          )  separately compiled procedure
```

We advise the user of the assert-module to use separate variable names for the ALLDO/EXDO-macros (bound variables) in order not to get into conflict with the variables of their procedures.

The PL/1 programmer of our installation will find the tool very helpful in writing reliable software. We therefore recommend its use and hope to get some suggestions to further develop it.

## Bibliography


– Parr, F.N., Lehman M.M., State-of-the-Art Survey of Software Reliability Final Report, May 1977, Research Contract No. 613.76.05 SISPE

– Kleene, S.C., Introduction to Metamathematics North Holland Publishing Co. 1952.


| Name | Syntax |
|---|---|
| 1. ASSERT | ASSERT (< predicate formula without quantifier > [< logical variable >] ); |
| 2. ALLDO | ALLDO (<PL/1-format of the DO-statement without the keyword 'DO'> ); |
| EXDO | EXDO (<PL/1-format of the DO-statement without the keyword 'DO'> ); |
| ENDAX | ENDAX; |
| 3. IFTRUE | IFTRUE (< sequence of any number of PL/1-statements> ); |
| IFFALSE | IFFALSE (< sequence of any number of PL/1-statements > ); |
| 4. ACTASSERT | ACTASSERT (< an integer between 0 and 2 > ); |
| SETASSERT | SETASSERT (< an integer between 0 and 4 > ); |

Figure 1

**Note to the Users**

Due to reasons beyond the control of the involved people an information meeting on the time-sharing system TSO which was scheduled for June 20, 1977 could not take place. This meeting is now planned for *tuesday September 27, 09.00 h in the amphitheatre of the CETIS.* All users who intend to use TSO in the future are invited to come. All users presently using the PSQ/FILEDI system are recommended to come.

*D. König*

The Editorial Board invites you to fill the form on page 23 of the No. 11 issue and reprinted on the next page.

## INQUIRY

*Name* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*Surname* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Are you still interested in receiving
the Computing Centre Newsletter? ☐ no ☐ yes

If yes, is your address correct? ☐ no ☐ yes

If no, please write it in capital letters:

*Address:* . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Can you please indicate the subjects you found more interesting and point out other items you would like to be treated?

| | | | |
|---|---|---|---|
| **Data base** | . . . | **Portability** | . . . |
| **High level languages** | . . . | **Structure programming** | . . . |
| **Simulation techniques** | . . . | **Time sharing** | . . . |
| **Network** | . . . | **Graphics** | . . . |

**Others** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Please send to:

**Mrs. A. Cambon**
**Support to Computing**
**Building 36, Tel. 730**

Les personnes intéressées et désireuses de recevoir régulièrement "Computing Centre Newsletter" sont priées de remplir le bulletin suivant et de l'envoyer à

**Mme A. Cambon**
**Support to Computing**
**Bât. 36, Tel. 730**

**Nom** ..................................................

**Adresse** ..................................................

..................................................

**Tel.** ................

---

The persons interested in receiving regularly the "Computing Centre Newsletter" are requested to fill out the following form and to send it to:

**Mrs. A. Cambon**
**Support to Computing**
**Building 36, Tel. 730**

**Nom** ..................................................

**Address** ..................................................

..................................................

**Tel.** ................