Second Edition

MACHINE-INDEPENDENT ORGANIC SOFTWARE TOOLS (MINT)

M.D. Godfrey, D.F. Hendry
H.J. Hermans, R.K. Hessenberg

TP31
G583
E2.

MACHINE-INDEPENDENT ORGANIC SOFTWARE TOOLS (MINT)

M.D. Godfrey
Sperry Univac, Blue Bell, Pennsylvania

D.F. Hendry
Computer Science Department, California Institute of Technology

H.J. Hermans
Sperry Univac, Blue Bell, Pennsylvania

R.K. Hessenberg
Micronology Ltd, London

2nd Edition

1982

ACADEMIC PRESS

A Subsidiary of Harcourt Brace Jovanovich, Publishers
London New York
Paris San Diego San Francisco
São Paulo Sydney Tokyo Toronto

ACADEMIC PRESS INC. (LONDON) LTD.
24/28 Oval Road
London NW1

United States Edition published by
ACADEMIC PRESS INC.
111 Fifth Avenue
New York, New York 10003

First edition published 1980

British Library Cataloguing in Publication Data

Machine-independent organic software tools.—2nd ed.
1. MINT (Computer programs)
I. Godfrey, M.D.
001.64'25 QA76.6

ISBN 0-12-286982-6

Printed in Great Britain by
Whitstable Litho Ltd, Whitstable, Kent

MACHINE-INDEPENDENT ORGANIC SOFTWARE TOOLS

Preface to the Second Edition

This edition is different from the first edition in two substantial ways. First, several corrections have been made. The most significant of these is the correction of the operation diagram for DICMATCH which appeared on page 202. The other corrections are all minor, being either typographical or obvious. The second difference is the introduction of several improved or new components. The significant improvements are: the INCH primitive which replaces GETSTR, a new virtual memory arrangement which permits use of 64K words of virtual memory and permits more efficient allocation of object text, a new portable format which is more compact and which is checksum and sequence checked, consolidation of the OPENxF primitives into the one new primitive OPENF, and provision of more powerful and selective diagnostics at the virtual machine level. These changes will not cause substantial incompatibility with respect to current source text. Where appropriate, routines are provided which allow continued operation of old constructs, such as a *function* GETSTR which uses INCH. In other cases, obvious changes should be made in source text which is based on the First Edition.

Chapter 12 has been expanded to include both MINT techniques and examples. Chapter 15 now contains a description of the MINT implementation on an Apple-II system instead of the Intel 8080 implementation.

During the period since the first edition we have benefitted substantially from discussions with and contributions from R. N. Riess. In particular, he contributed the EMULATE primitive which is described in Chapter 6.

This edition has been produced by the Sperry Univac COMADS system, as was the first edition. Thus, the process of production of the new edition was to write the new text, edit it into the first edition files, apply the usual spelling checking and analysis tests, run proof copy, correct and run final camera-ready copy. It is again a pleasure to acknowledge the help of Richard H. Acquard who is responsible for the COMADS language processor. In addition, Paul J. Pontinen, who has responsibility for the implementation of COMADS on the COMp 80 microfilm processor, has been particularly helpful in providing additional processing capabilities. These enhancements have improved the appearance of the result, and the ease of its production.

We have been pleased by the reactions of readers during the two years since the publication of the first edition. There have been numerous requests for copies of the system. We have found in practice that most potential users can accept the system on ANSI-format magnetic tape. Due to the problems of formats of cassettes and floppy disks, and our own access to facilities, we have had to restrict availability to magnetic tape, a floppy disk suitable for bootstrap loading into an Apple-II+ system, or, in special cases, transmission over communication lines.

Michael D. Godfrey
June 1982

Preface to the First Edition

The tools described in this monograph are intended to improve the efficiency of computer use and increase the value of the written instructions (termed *software)* which control the operation of computing machines. This is achieved through simplification and generalization of basic constructs, and through separation of the written software from the machines on which the software may operate.

It is intended that this monograph serve several purposes. First, it represents a complete summary of a body of research and development which has been underway since the late 1960's. Second, the content and level of presentation are such that the text may be used for advanced undergraduate or graduate courses in design and implementation of languages, virtual machines, or simple stack based processors. In addition, the text contains information which should be of interest to professional software writers or system designers. The example implementations of the system can be used as trial implementations for study, or may be used as the basis for production application implementations. In practice, these tools have been found to be highly effective for a wide range of applications on machines of widely differing structure. We hope that this monograph will help others to make effective practical use of these tools and techniques.

Until recently these tools were called the SNIBBOL system. While SNIBBOL is as good a name as any other (better than most we could think of), confusion with SNOBOL and other possible misleading associations led us to change the name to MINT (Machine–INdependent Organic Software Tools).

MINT has been put to practical use at several places. This practical use has been essential to the development of the system and, we hope, productive in its own right. The initial development of MINT took place in the early 1970's while D. F. Hendry was at the University of London Institute for Computer Science. MINT was used there as a part of the M.Sc. course. Many further uses have occurred in more recent years. We are aware of MINT implementations for about ten different computer systems.

Many users of MINT are known to the authors. Many of these have contributed significantly to further development of the system. We would like to acknowledge this help even though it is not feasible to list all the individuals who have made such contributions.

D. F. Hendry has been responsible for most of the basic concepts of MINT as it exists today. The current compiler implementation was created by Hendry. Initially R. K. Hessenberg tested and corrected the compiler, as well as contributing helpful insights and improvements. Subsequently, the compiler has been modified and extended by H. J. Hermans and M. D. Godfrey. Hessenberg and Hendry wrote the initial version of the Sperry Univac Series 1100 interpreter (described in Chapter 16) with some help from Godfrey, who has subsequently modified and extended the implementation. Hermans wrote the Intel 8080 interpreter which is described in Chapter 15. An initial MINT manual was prepared by Hendry and Hessenberg. That manual was extensively used in the preparation of Chapters 2 through 11 and Chapter 13 of the present monograph. The completion of the monograph in its present form has been carried out by Godfrey and Hermans.

This entire text, including all Tables and Figures, was prepared by means of a Sperry Univac computer–based documentation system (COMADS). It is hoped that the text reflects the quality of this system. The system greatly facilitated the writing task, as many time–consuming activities, such as proof–reading, were carried out by the computer. The fact that the entire document is stored in the computer has allowed use of the actual source files where language text is given. Thus, all such text has been processed as source text by the MINT system, and therefore checked for correctness. The use of computer–based tools did not completely remove the need for human assistance. Specifically, Richard H. Acquard has been extremely helpful in giving advice and providing support concerning the operation of the COMADS system.

This monograph is unusual in that the complete source text of the system (compiler, virtual machine, syntax analyzer, other text, and examples) are given. This demonstrates the compactness and readability of the system. By agreement with Academic Press Inc. (London) Ltd. the authors retain the copyright of this machine–readable source text.

Copies of the system in machine-readable form may be obtained by writing to me. When such a request is made, it is essential to state the required medium from the following choices:

1. Industry standard magnetic tape, 9-track, 1600 bpi, ASCII coded card images.

2. Standard cassette tape, ASCII coded images.

3. Another recording device which has a standard RS-232C interface. In this case the recipient must provide the recording device and the recording medium.

There will be a charge made in order to cover the cost of copying.

Michael D. Godfrey
May 1980

8063498

Contents

The love of economy is the root of all virtue.
G. B. Shaw

1. The MINT System

1.1. Introduction

The MINT system is a set of tools to facilitate communication with, and operation of, computers. These tools provide a high level software environment which is machine–independent and open–ended. The machine–independence implies that the MINT system, and MINT based applications, are readily portable to many machines. The open–endedness implies considerable flexibility in altering or extending the language facilities. The language itself allows sequences and expressions at as high or as low a level as is desired.

MINT is implemented in terms of a *Virtual Machine* which allows exactly the same (virtual) environment to exist regardless of the actual machine on which the system is operating. This Virtual Machine is referred to as the VM(M) Virtual Machine, and the instructions which the Virtual Processor executes are the VM(M) instruction set. Careful definition of this Virtual Machine contributes to the portability, compactness, efficiency, and verifiable correctness of the system.

1.2. Scope

The scope of MINT is very wide both in terms of machines on which it may operate and in terms of potential applications. At present MINT operates on such machines as the Apple–II and on large general–purpose mainframe systems such as the Sperry Univac Series 1100. Applications which have been written entirely in MINT include a number of compilers and assemblers for both small and large machines, language interpreters, a text editor, and interactive dialogue systems. These implementations were all relatively low cost in terms of development and implementation effort when compared to similar efforts using conventional techniques. The resulting programs are readable, and portable to any new machine.

In addition to its direct usefulness as a set of development and implementation tools, MINT can be an effective means of communication. MINT written text is precise, compact, and readable. The MINT Virtual Machine is a simple and carefully structured machine which displays the essential features of a stack–based (or zero–address) machine architecture. Thus, the MINT system provides an effective means of communication between people, between machines, and between people and machines.

The emphasis on effectiveness of communication makes MINT suitable for teaching computing principles and techniques. The system may be used to teach or learn about stack–based architecture, virtual machine design and implementation, compiling, macro structure, parsing, and concepts and techniques of portability. In this text we have not attempted a strict separation of these subjects. This is because we feel that they are not reasonably separable. Much of the effectiveness and interest in a system such as MINT derives from the structural relationships of the components, rather than from the components themselves. Thus, in this text, we have tried to develop an understanding of how MINT fits together. This may initially seem to impede learning, where compartmentalization is always a strong temptation. However, we believe that the end result will be found to be beneficial. The complete MINT system is more significant than the sum of its parts.

1.3. Purpose

The purpose of MINT is to facilitate the analysis and transformation of structured symbolic information. An example of such analysis and transformation is a conventional language compiler. Other examples include text–editing routines (such as those given in Section 12.6) or interactive dialogue systems for specific applications.

In order to satisfy a wide range of possible requirements, the system is organized in the form of a set of general–purpose tools. These tools may be used for many purposes, including the development of new tools. The system itself is constructed by means of the tools which it provides for general use. The open and modifiable structure of the system is essential to its generality, and allows a holistic approach to many problems which previously required ad–hoc solution methods.

Due to its compact structure, MINT is well suited for use in very small machines. An eight bit processor with 32K (here, and throughout, K is used to mean 1024) bytes of storage is sufficient for many purposes. However, the system also operates effectively on large–scale systems.

The complete machine independence of the MINT language permits the writing of systems which may have wide applicability and permanent value.

1.4. Background

Historically, computing has developed from a primary interest in the algorithms required for solution of numerical problems. The earliest forms of computing were characterized by relatively large algorithmic programs which operated on relatively small quantities of data. As computing technology developed there was a tendency to apply the tools which were developed for this structure to other, often non–numerical, problems. At the same time, the volumes of data, both numerical and symbolic, began to grow very rapidly. At present it is frequently the case that the amounts of data to be processed far exceed the size of the processing programs. Usually, the purely numerical processing accounts for only a very small part of the total. Thus, it is natural to question the basic structure of current computing tools, based as they are on conditions which no longer prevail. It is clear that if programs are used to process very large quantities of data, the value of the program and the importance of correctness of the program are increased. It is also evident that much of the complexity of current computing derives from the attempt to develop algorithms which express symbolic transformations. Finally, the slow and cumbersome operation of early computers provided the incentive for investment in improved efficiency of program execution. Early programs were often operated without substantial change for long periods. Thus, substantial effort in writing and understanding the program could be justified. The speed and flexibility of current computers imply that the limiting factor in their productive use is the rate at which information which is understood by humans can be precisely and correctly communicated to and from the computing system.

These background considerations have led us to attempt to develop new language tools which are based on the view that data transformation is the fundamental task, and that machine