COMPUTER
SYSTEMS
ENGINEERS

SALARY: TO $15,000

Several immediate openings exist at Hughes-Fullerton for Computer Systems Engineers qualified in the design and testing of large scale digital systems—and in the preparation of proposals for large scale digital computers.

These professional assignments involve such R & D areas as:

- **Solid state digital circuitry involving millimicrosecond logic**
- **Microwave carrier digital circuits**
- **Sub-microsecond core memory**
- **Thin film storage techniques**
- **Functional circuit concepts**
- **Micro-miniaturization concepts**
- **Tunnel diodes** ▪ **Microwave parametrons** ▪ **Circuit organization for maximal-speed computing.**

Located in Southern California's Orange County (the nation's fastest growing electronics center), Hughes-Fullerton offers you: a stimulating working environment; private or semi-private offices; long-term stability.

CALL COLLECT TODAY!
For complete information on these challenging assignments, call us collect **today!** Ask for:
Mr. B. P. RAMSTACK at: TRojan 1-4080, ext. 3741.
Or, airmail resume to: HUGHES-FULLERTON R & D, P. O. Box 2097, Fullerton 1, California.

HUGHES

Letters to the Editor

Dear Editor:

The presently high interest in character sets and the graphic representation of symbols has again focused attention on the most confusing graphics used in communication.

The sixteenth letter of the English alphabet (O) and the arabic numeral for zero (0) are for practical purposes the same graphic. There have been many proposals for establishing a clear difference in their graphics. Within certain specific systems these proposals have been carried out. Some of these are shown below:

Letter	*Numeral*
O	O
Ø	O
⊙	O
ơ	O
O	Ø
O	⊙

Utter confusion is indicated.

A close look at the problem, however, indicates that originally it was half again as bad. The numeral zero, the letter "O" and the eighteenth letter "Q" all had the same graphic. The "Q" problem was solved forever by simply always following it with a "U". The letter "O" problem is also solved by always following it by an "H".

Nohw that we are freed frohm this unhohly dilemma we may lohohk fohrward toh prohgress in the mohre impohrtant aspects ohf the cohmputing sciences.

F. A. Williams
IBM, New York

Dear Editor:

There appears to be a certain amount of programmer interest, currently, in recursively-defined functions and in self-calling subroutines to evaluate these functions. The ability of some new programming systems to handle such situations has been claimed. Programmers without these powerful tools, however, wonder about other ways of handling the same functions. In the interest of practical, economical computing, and with the help of some basic facts from the theory of recursive functions, the following suggestions are made.

When to recur? Never if you can avoid it. Recursive definitions provide a neat way for mathematicians to define functions, and are very convenient for proving things about functions by mathematical induction. However, they are a poor form in which to specify functions for computation. The difference between computing a function from a recursive definition and from a closed form or analytical expression is exactly the difference between looking up its values in a serial memory and in a random access memory.

Unfortunately, however, recursion cannot always be avoided. It can be proved that certain functions cannot be defined in terms of certain sets of primitive functions except by recursion or something equivalent to it. For example, on most computers the four operations of arithmetic are available as primitive functions, and such functions as exponentiation and the factorial can be defined in terms of these only with the aid of recursion. Even here, other computing devices such as tabulated values or analytical approximations (for example, Stirling's formula for large factorials) may well prove more economical than computing from a recursive definition.

How to recur? From the bottom up if possible. That is, to compute a recursively-defined function $f(n)$, start with $f(0)$ and work upwards by increasing the variable until n is reached. This can be done by a perfectly straightforward subroutine. A subroutine which starts with n and operates by decreasing the variable and re-entering itself until it reaches 0 is bound to be slower. It has the same n-level hole to climb out of as the straightforward subroutine, but the recursive subroutine must first take additional time to back itself into the hole, level by level.

There is a very large and useful class of functions, the "primitive recursive" functions, for which it is always possible to recur from the bottom up. The vast majority of interesting functions evaluated on computers are certainly primitive recursive (although they may not always be recognized as such). On the other hand, there do exist computable functions (not primitive recursive) which cannot be computed from the bottom up by means of their recursive definitions in terms of the usual primitive functions. These functions are very likely to be pathological in other ways as well. A practical scheme for computing their values may require non-trivial efforts by a recursive function expert, a numerical analyst, or both.

There are circumstances, other than the computing of functions from recursive definitions, in which the problem arises of a subroutine operating simultaneously on more than one level. Usually the question is one of producing extremely general and unrestricted program components, which may be interconnected with complete freedom. The preceding remarks of course do not apply here.

H. G. Rice
System Development Corporation
Center for Research in System Sciences
Santa Monica, California

Dear Editor:

We have been studying the "Report on the Algorithmic Language ALGOL 60" both because it pertains to an area of particular interest to us, and because we wish to participate in the interchange of algorithms via the *Communications*. We feel that the Committee has done a commendable job in describing the language. However, a Talmudic scrutiny of the report has provoked from us a number of remarks. Some of these remarks are intended as corrections, or as points for clarification. Still others are criticisms of ALGOL 60. The points are ordered not by importance, but rather by correspondence to the pertinent sections of the report.

1. A somewhat involved but not uninteresting set of definitions is given for strings in section 2.6.1. However, not only are no operations on strings defined, but it is implicitly stated (4.7.5.1) that strings may occur only as parameters to machine language procedures. If so, why the involved definitions? One wonders if the proponent of the string definitions got his foot over the threshold, only to have the door slammed upon it. As a matter of general interest we would like to know what the full proposals were. We strongly favor the inclusion of defined operations upon strings, since without them many of the algorithms that we would like to submit to the *Communications* can be expressed at best awkwardly.

2. We feel that a distinction should be made between labels that are prefixed to statements or blocks, and the references to these labels which may occur in **go to** statements or switches. This distinction would correct the ambiguity that occurs in section 4.1.3 where it is stated that "labels . . . are local to the block in which they occur." If the preceding quotation is to be taken literally, one may never exit from a block with a **go to** statement in which a label is the designational expression. This, however, would appear to contradict section 5, paragraph 3, which unqualifiedly permits "an exit from a block . . . by a **go to** statement".

3. We regret the omission of parallel for clauses as recommended by the Share Algol Committee. (H. Bratman, et al,

(*Please turn to Page 506*)

COMPUTER RESEARCH ENGINEERS & LOGICAL DESIGNERS

SALARY: TO $15,000

Rapid expansion of the Computer Laboratory at Hughes-Fullerton has created several attractive professional opportunities for qualified Computer Research Engineers and Logical Designers. These positions require active participation in broad computer R & D activities in connection with Army/Navy computer systems and a *new* logi-scale, general-purpose computer: the MPC-1200. This multiple processor computer utilizes advanced solid-state circuitry, gating and resolution times in the millimicrosecond regions; combines synchronous and asynchronous techniques for maximum speed and reliability.

These professional assignments involve broad areas of logical design, programming and system conception. Fields of interest include:

▪ Distributed computers ▪ Advanced arithmetic processing techniques ▪ Mechanized design ▪ Asynchronous design techniques ▪ Utilization of parametrons in computers ▪ Studies in the utilization of multiple processor computers.

These professional assignments involve such R & D areas as:

▪ Solid state digital circuitry involving millimicrosecond logic
▪ Microwave carrier digital circuits
▪ Sub-microsecond core memory
▪ Thin film storage techniques
▪ Functional circuit concepts
▪ Micro-miniaturization concepts
▪ Tunnel diodes ▪ Microwave parametrons ▪ Circuit organization for maximal-speed computing.

Located in Southern California's Orange County (the nation's fastest growing electronics center), Hughes-Fullerton offers you: a stimulating working environment; private or semi-private offices; long-term stability.

CALL COLLECT TODAY!
For complete information on these challenging assignments, call us collect **today!** Ask for:
Mr. B. P. RAMSTACK at: TRojan 1-4080, ext. 3741.
Or, airmail resume to: HUGHES-FULLERTON R & D, P. O. Box 2097, Fullerton 1, California.

HUGHES
HUGHES AIRCRAFT COMPANY

Editor's Comments on Compilers

The following comments are meant to provide additional information and not to detract from the value of Dr. Blatt's complaints (see Opinions, p. 501) which, while specifying FORTRAN, refer to many others as well. Certainly it is about time that more compiler builders started designing translators for the good programmer.—A.J.P.

[a] Actually many such compiler manuals have been written by users, though they have not been widely distributed, e.g., a FORTRAN manual by Westinghouse, and an IT manual by Texas Instruments. However, to accent the author's complaint, most of these manuals are intended to further isolate the occasional user from the machine. Is it not obvious that in the next few years —if not already—there will be a large educated audience who will be able to use—and probably insist upon—more control over the manipulation of their codes originally composed in an ALGOL-like form.

[b] Actually there has been a large number of compilers built which used the stated principle as the design motivation, e.g., IT, RUNCIBLE, GAT, and CORREGATE for the IBM 650, and MAD for the IBM 704, to name an admittedly partial list. CORREGATE permits modifications to the object code in source language with only these modifications retranslated. An ALGOL translator is being built at Oak Ridge, and in several centers in Europe, to function as type B compilers. Indeed, one at Mainz translates at the paper tape input speed and the object code commences running when the tape has been completely read. Ironically there have been some complaints that ALGOL—as a language—has been heavily organized so as to permit type B translators to be built.

[c] The translator GENIE, being built at the Rice Institute, is an example of a system where certain machine properties can be exploited in codes written in GENIE. In general, they do not appear to be too difficult to translate into actions on other machines than the one for which GENIE was designed.

[d] The author here refers to the assembly problem, many of whose aspects are independent of the form of the source code. Systems such as the authors would like to see are currently being built, e.g., the ACT system designed for the Signal Corps.

LETTERS (continued)

Recommendations of the SHARE ALGOL Committee, *Comm. Assoc. Comp. Mach. 2* (Oct. 1959), 25–26.

4. The absence of the return statement as defined in ALGOL 58 necessitates the use of an artifice such as a labeled dummy statement, in the event that the last written statement in a procedure body is not necessarily the last executed statement. We believe that the committee should offer some justification for its action in this matter.

5. We notice that there is no stop statement in ALGOL 60. Admittedly, "stop" may mean all things to all translators, but there should be some standard method for denoting the termination of a dynamic statement sequence.

6. Section 4.7.6 appears to be incomplete and unnecessarily complex, by virtue of the following reasoning:

(a) If a quantity is non-local to a procedure body, it must be local to some block which includes the procedure body, else the procedure body is not completely defined.

(b) Hence, "a procedure statement written outside the scope of any non-local quantity of the procedure body" is *ipso facto* outside the scope of the procedure body, and is accordingly undefined. (The scope of a procedure body may be defined analoguosly to the scope of a label, where the procedure identifier in the heading and the same identifier in a statement correspond respectively to a particular label and a reference to it.)

Thus, section 4.7.6 seems to be a special case of the principle of scope, and might be emended to read as follows:

"A procedure statement is defined if and only if it occurs within the scope of the procedure body, and the procedure body is completely defined."

7. We regret that no provision for the specification of initial data was made. If ALGOL was designed primarily for the communication of algorithms rather than for machine implementation, then we concede that such a provision is unnecessary.

8. Section 5.4.4 implies that a function designator may occur *only* as the left part of an assignment statement, lest the procedure be activated recursively. Was this the intention of the Committee?

Any comments from the Committee members or from interested bystanders would be welcomed.

H. ISBITZ W. DOBRUSKY
RUTH ANDERSON D. ENGLUND
E. BOOK H. MANELOWITZ
H. BRATMAN SONYA SHAPIRO
System Development Corporation
Santa Monica, California

Note of Amplification

E. F. CODD

In parts 1 and 2 of the paper "Multiprogram Scheduling" (June 1960 issue, pp. 347–350), the term "space-shared" is used. It seems desirable to clarify the scope of this term, particularly as in one instance the term was altered to "space-(memory)-shared."

The term "space-shared" applies not only to the internal storage (e.g., core) but also to auxiliary storage and input-output devices (e.g., drums, disks, tape units, card readers, and printers). In the case of tape units, all tape units of a given type constitute a single (composite) space-shared facility for which the natural unit of space is a single tape unit. A similar remark applies to card readers and printers.

It should be emphasized that the scheduling algorithm described in Part 3 of the paper handles in one operation any number of different facilities and is not a scheme for internal storage alone.