Program composition and editing with an on-line display

by HARVEY BRATMAN, HIRAM G. MARTIN, and ELLEN CLARK PERSTEIN
System Development Corporation
Santa Monica, California

INTRODUCTION

An Interactive Programming Support System (IPSS) has been under development at System Development Corporation since 1965.* The purpose of the system is to permit all of the programming processes—composition (in a procedure-oriented language), editing, execution, testing, and documentation—to be carried out as parts of a single, coordinated activity centered around an interactive compiler. IPSS attempts to unify techniques that are usually embodied in separate functional programs, so that the programmer need not know which particular program is performing a specific task. The system is intended for a time-sharing environment, with user interaction via a small tabular display or typewriter-like terminal.

In the development of IPSS, emphasis has been placed on the functions of program composition and editing, particularly via an interactive display. The current object of this work has been to investigate the use of a small tabular display by a professional programmer to compose and edit programs and to retrieve dynamic information about the state of his program. This paper reports the results of three aspects of that work.

- It describes how the display is used, how the display compares with a typewriter-like terminal, and how the user interacts with the display to accomplish the tasks of program composition and editing.
- It describes the advantages of doing syntactic analysis concurrently with program composition and editing, how program information is obtained, and how IPSS does the analysis and editing.
- It describes the software interface between IPSS and the display hardware, how IPSS can be used with various displays, and what the characteristics of an "ideal" display are.

Background

The Interactive Programming Support System operates under SDC's new time-sharing system (known as ADEPT-50) on an IBM 360 Model 50 computer. Since one of the major purposes of ADEPT-50 is to support large-scale program development, testing, and debugging, IPSS is considered to be an essential element of the ADEPT-50 system.

A small, tabular display (rather than a typewriter-like terminal) was chosen as the primary interactive device in this work for the following reasons. First, we are looking forward to the near future when relatively inexpensive display devices are expected to be available. These devices could be installed directly in the programmer's office or work area, and could be connected to a central time-sharing system. Second, a display provides faster response by presenting many lines of output in the same time a hardcopy device requires to print one line. Thus we hope to eliminate the need for the programmer to search through numerous pages of program listings, hoping to find some single piece of information, but not knowing if the listing really corresponds to the current state of his program. Instead, he will be able to display all the information he wishes to see, and will be assured that it reflects the true state of his program at that instant. Third, the display scope minimizes the amount of typing the programmer must do to compose and edit his program. The programmer is able to use the "replace-character" feature of a display scope to modify parts of his program, rather than having to retype whole lines. Previous systems (such as the Q.E.D. Time-Sharing Editor developed at the University of

*This work has been supported in part by the Advanced Research Projects Agency of the Department of Defense.
California, Berkeley, or the editing routines used under the M.I.T. Compatible Time-Sharing System show the ingenuity and complexity required to perform similar functions with a typewriter terminal.

The display device used in this project is an IBM 2250. This device is more powerful and expensive than necessary for this work, but we can use it to simulate the characteristics of displays such as the Sanders 720, Computer Communications Inc. 300 TV display, or RCA video data terminal—which are typical of the devices we are considering.

The current system is limited to the use of the JOVIAL programming language (since this is the most readily available language at SDC). The operational principles derived in this work, however, are applicable to other languages such as FORTRAN or PL/I. The user can choose either the 2250 display or an IBM 2741, a typewriter-like device, as his interactive medium. However, some IPSS functions such as automatic program display and minimization of programmer typing can be done only on a display scope; other functions may be too time consuming to be used often on the IBM 2741. The current capabilities of IPSS allow the user to compose and edit his program interactively. All program statements are checked for syntactic correctness as they are input, and correct statements are automatically displayed to show the current status of program composition. The programmer is informed of errors in syntax, which he can correct immediately. Syntactic checking combined with editing insures that the edited program is free from the more common programming errors. Thus, the programmer need not be afraid of unpleasant surprises when his program is compiled. The programmer can display parts of his program, he can obtain current "where set" and "where used" information about variables in his program, and he can obtain all the references to any program label.

Use of the display

The display is used to convey information both from IPSS to the user, and to IPSS from the user. IPSS either requests input and waits for the user to respond, or it outputs information in response to user actions and again waits for the next user response. User responses may be answers to specific questions from IPSS, program statements, or requests for IPSS action.

The face of the display has been organized into three logical "blocks" to facilitate these uses (see Figure 1). The number pair shown in Figure 1 represents block and field numbers; e.g., 3,0 represents block 2, field 0. Each line is composed of one field except for block 1, where the line is divided into two fields. A "block" is roughly defined as a physical segment of the display surface, set aside for some specific function. Block 1 is the area where IPSS requests user actions. IPSS signals the user when it is ready for input by displaying two question marks (?) in position 1,0 (block 1, field 0) and by placing a cursor in position 2,0. A descriptive message is written in position 1,1. Block 2 is the work area where the user responds to IPSS input requests. Since the cursor has been placed in position 2,0, typing on the keyboard causes input to appear there. The user is restricted by IPSS from typing into block 3. Block 3 is the area where IPSS displays information requested by the user or automatically generated in response to some user action. Every line but the first contains one field of 72 characters. The first line is divided into two fields of 2 and 72 characters each. Block 1 consists of 1 line, block 2 consists of 5 lines, and block 3 consists of 47 lines. The size of the blocks can be varied for experimental purposes.

The IBM 2250 has been available to us only since February 1968. (Prior to that time, we used the IBM 2741 to test many IPSS functions.) Hence our ideas on how to use the display are still evolving as the result of experience. We have decided on some general principles to guide us in the use of block 2, the user work area, and block 3, the program's output area. To meet the stated goals of IPSS, the minimum sizes for blocks 2 and 3 are 2 and 20 lines, respectively. Hence any practical display scope should hold about 25 lines of at least 50 characters each.

We wish to minimize the amount of typing the user must do. As shown below, IPSS will, in the case of error correction and editing, regenerate program statements on block 2 so that the user can modify them as necessary, and does not have to retype entire statements. The user can add new displays to block 3 without destroying the display already there. He can change a current dis-

<table>
<thead>
<tr>
<th>2</th>
<th>72 Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>1,1</td>
</tr>
<tr>
<td>2,0</td>
<td></td>
</tr>
<tr>
<td>2,1</td>
<td></td>
</tr>
<tr>
<td>2,2</td>
<td></td>
</tr>
<tr>
<td>2,3</td>
<td></td>
</tr>
<tr>
<td>2,4</td>
<td></td>
</tr>
<tr>
<td>3,0</td>
<td></td>
</tr>
<tr>
<td>3,1</td>
<td></td>
</tr>
<tr>
<td>3,2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3,46</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3,47</td>
<td></td>
</tr>
<tr>
<td>Block 3 (47 lines)</td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 1—Display organization

From the collection of the Computer History Museum (www.computerhistory.org)
play to make room for a new display, and thus make effective use of the display as a substitute for leafing through program listings.

Display interaction in program composition

An example of program composition is shown to describe the use of the display and also to point out some advantages of a display over a typewriter-like terminal. The example corresponds to the program shown in Figure 2. Figure 3 shows how the display is used in program composition. Block 3 on the display surface contains an automatic display of the program thus far composed. Statement numbers corresponding to the last statement on each line have been added. The program label is formatted on the same line with a statement so that its statement number does not appear. For editing purposes, INITIAL. is statement 2 and XX = 10000$ is statement 3. Block 1 indicates that IPSS is ready for more input. The user has typed more of his program in block 2. There are two incorrect statements in this example. They will be discussed later. When the user is satisfied with the contents of block 2, he sends it to IPSS by pressing a "SEND" key.

Figure 4 shows the results of IPSS action on the input.
put. Correct statements 4 and 5 have been deleted from block 2; they have been reformatted and added to block 3. The incorrect statement

GOYOVER$

is flagged by an asterisk (*) in block 1 and the type of error is explained at the top of block 3. The cursor has been placed in block 2, ready for user input. The user can correct the statement to read

GOTO OVER$

by positioning the cursor under the Y, typing T and sending the entire block 2 again to IPSS. He does not have to retype the rest of his previous input because IPSS has restored it to block 2. If he had been using a typewriter-like terminal, he would have had to retype the entire incorrect statement and everything after it. (The statement $XX = XX + 18$ is logically incorrect. It will be used later to illustrate editing and debugging.)

Display interaction in program editing

The user can change his program in several ways: he can delete lines, replace lines, insert new lines, copy lines, and move lines. He can also change “signs” within statements in a particular area of his program. These tasks are all done by typing the appropriate edit commands in block 2. The user can save himself extra typing if the change consists of modifying statements. The command

#COPY M,N$

brings lines M through N to block 2. The user can then treat the lines as if he had typed them there. For example, suppose that after compilation, execution, and debugging, the programmer finds that statement 7 is logically wrong and must be replaced. Figure 5 shows the response of IPSS after the following sequence. The programmer types and sends the command

#COPY 7$

The response by IPSS is to bring statement 7

$XX = XX + 1$

to block 2. The user then types the command

#REPLACE 7$

He positions the cursor under the “+” and changes it to a “−.” When he sends block 2 to IPSS, statement 7 will be replaced. Block 3 will be modified to show the replacement.

The length of the program is not limited to the length of block 3. When the program exceeds the length of block 3, lines at the top are removed and the new lines appear at the bottom. The line identifiers M,N in the command

#COPY M,N

can refer to any lines of the program, visible or not. The seemingly more straightforward technique of moving the cursor to the appropriate position in block 3 has not been included in IPSS. The technique of determining which lines have been changed in block 3 is difficult in the type of display equipment we are considering. We have avoided the use of expensive, complex hardware such as light pens and function keys. We are assuming just the ability to move a cursor and send an interrupt to the main computer. (The TVEDIT program, which runs under the THOR Display Based Time-Sharing System at Stanford University, provides an example of what can be done with more complex equipment).

Other display interaction

Consider the previous example of program editing.
Suppose that in the course of debugging, the programmer knows that the variable $XX$ is being set incorrectly, but he does not yet know where. Assume that the program in question is a large one and that the programmer does not remember every place the identifier $XX$ is used.

The following sequence of commands shows how the programmer can make use of saved program information, and can use displays to find and correct errors in program logic (see Figure 6).

The user first types

\#DISPLAY SET USED $XX$

IPSS clears the display and outputs at the top of block 3:

$$XX \ S3 \ U5 \ B7$$

This indicates that the identifier $XX$ is set in line 3, used in line 5, and both set and used in line 7. The user then types**

\#ADD DISPLAY 3

which adds statement 3 to the current display, as follows:

0003. INITIAL. $XX = 10000$

He then types**

\#ADD DISPLAY 7

which adds statement 7

0007. $XX = XX + 1$

Upon reflection, he determines that statement 7 is wrong and changes it as previously described.

**Other IPSS commands**

This paper will not describe in detail all the commands available to the user. In brief, the user can display all or part of his program using either statement numbers or symbolic names for the areas. He can request that the current program display be rolled up or down like a scroll. He can display the names and corresponding identifying statement numbers of every procedure and statement label. He can display the statement number of every reference to any identifier. In addition, any display can be directed toward a hard-copy printer.

The editing commands are described in the following section.

When the programmer is satisfied with his program, he can compile, execute and debug it using facilities available in the ADEPT-50 Time-Sharing System. Eventually, we plan to include compilation of modified program segments and execution-time debugging in IPSS.

**Techniques of program editing used by IPSS**

The previous sections of this paper have described the user interaction with IPSS via a display scope. The display scope was chosen for the convenience of the user and because it offers facilities that—on other devices—are cumbersome or impossible. The techniques IPSS uses to carry out the user composition and editing requests are, in general, independent of what interactive device is employed. Their only requirements are that they are efficient in their use of time and do not impose undue restrictions on user actions.

Editing is permitted in any part of a program and in any order that the user desires. The program is blocked, and blocks are output to disc as required. IPSS keeps a record of what statement numbers are included in a disc record, and permits insertions or deletions of any length anywhere in the program. The disc records need not be in order according to statement numbers. This makes it easy to insert statements anywhere in the program,
since new disc records can be added at the end. An attempt is made to combine two records into one after statements are deleted, when this is possible without causing extra input/output. A disc record freed in this way becomes available for reuse whenever needed. Output of records is minimized by not outputting until a user request forces it (unknowingly, of course, since he has no concern with the blocking and input/output of his program.) Input of records is minimized by remembering what disc records are currently in core.

Editing is done on program segments where a program segment is a single statement or a consecutive series of statements identified by its starting statement number or starting statement label, and by its ending statement number or ending statement label. (The ending statement need not be given if only one statement is involved.) Such program segments may be deleted from the program, moved to (inserted after) another program statement, copied after another program statement (this is similar to moving except that the statements are also retained in their original place), or replaced by a new series of statements that the user inserts.

The statement numbers assigned by IPSS during program composition permit up to 1000 insertion statement numbers between statements. During editing, IPSS computes insertion numbers based upon the number of statements to be inserted at a time, and the numbers of the bracketing statements. The user may override the automatic assignment of statement numbers by prefixing the first of a series of statements with the starting number he desires. Automatic assignment of following statement numbers depends upon the number he uses and the next statement number in the program.

The DELETE, MOVE, and COPY commands are complete in themselves; IPSS assumes that the user wishes to continue program composition when they have been executed. The INSERT and REPLACE commands assume that any number of statements may follow in one or more inputs; therefore, IPSS continues to assign statement numbers depending upon the numbers available where the statements are inserted. The user may indicate that he wishes to continue program composition by inputting COMPOSE. Block 1 of the display scope shows the user whether he is in the edit mode or the compose mode.

A COPY command that does not indicate where the statements are to go causes the statements to appear on block 2 of the display scope. The user may then modify them as desired and give a command to insert them anywhere in the program.

A SEQUENCE command causes the entire program to be resequenced. An option permits specifying the starting statement number and statement increment to be used in the resequencing.

All editing commands previously discussed apply to entire program statements. There is also a CHANGE command that permits any specified string of signs (except a null set) within a statement to be replaced with any other specified string of signs (including a null set). The range of statement numbers to be searched for the substitutions may be limited if desired. At the end of a statement in which one or more substitutions of signs have been made, the old statement is deleted and the new version of the statement is checked for syntactic correctness; it then replaces the old statement. This process continues until the end of the specified statement range is reached or until an error in syntax is detected.

IPSS tries to be permissive to the user and often provides optional ways of accomplishing the same result. For example, a REPLACE N command followed by a series of statements accomplishes the same result as a DELETE N command and an INSERT AT N command followed by the series of statements. User convenience is considered more important than optimum efficiency.

Program information storage and retrieval

At all stages of program composition and editing, a list (ordered by statement number) is maintained for each identifier in the program. Each list shows every current reference to the identifier and denotes whether the identifier is declared, set, or used in the statement. Deletions of statements cause deletions of references in the appropriate identifier lists; insertions of statements cause insertions of references in the appropriate identifier lists. Thus, at any time, the user may ask where a variable is set or used and the information is immediately available for display. A link deleted from the list for one identifier is available for reuse as needed for any identifier. In this way the size of the table needed for the set/used information is not prohibitive.

A dictionary is maintained with an entry for each identifier in the program. Information that helps in syntactic analysis and links to other tables (such as the set/used table and the table of procedures and tables) is kept in the dictionary entry. If a declaration of an identifier is deleted, it is assumed that this identifier will be redeclared and it will then be assigned to the same dictionary entry, so that the proper linkages are maintained. For this reason a dictionary entry is not reusable for another identifier even if the declaration and all references to the original identifier are deleted.

Other program information available to the user by command consists of a list of procedure names and their scope denoted by statement numbers, and a list of statement labels and corresponding statement numbers.
within the entire program or within any procedure.

Syntax analysis, error detection and correction

Most of the error detection done by IPSS is on an individual statement basis. All such errors within statements are detected regardless of whether the user is composing or editing his program. Some additional checks involving sequences of statements are made during program composition. During composition, a memory of preceding statements is maintained so that errors such as an incorrect table structure or a conditional clause immediately following another conditional clause are detected, whereas these same errors may go undetected when the user is editing his program. In the compose mode, an error message may tell the user what input is required; in this event only that particular input will be accepted next as a part of his program.

It is possible (though, in some cases, difficult) to detect these same errors in the edit mode but it is not considered desirable to do so, since this would force the user to construct his program in a particular order. It is not required that a program be correct at all stages of program composition. The final program is the one that counts; consequently, some kinds of errors involving sequences of statements are left for the compiler to catch. Most of the errors are detected by IPSS and the user is given a chance to correct these before compilation is attempted. When consecutive statements are input in the edit mode, some memory can be built up; the error checking of statement sequences approaches that of the compose mode.

In order that program names occurring in edited statements may be correctly identified as to class and scope, IPSS keeps track of the beginning and ending statement numbers associated with each table and each procedure in the program. This identification is necessary for error detection within individual statements. It certainly would be possible for the user to confuse IPSS, for example, by deleting the END of one procedure and the heading of the next in order to combine the two procedures into one. This may accomplish what he wishes, but some error checking is lost. He can accomplish the same result without the confusion by moving the desired statements from one procedure to another and then deleting the unneeded statements. More complete error detection is obtained through this second method.

When the end of the program is signaled by TERM$, or when the user requests that his program be compiled or saved, each procedure and the main program is checked to see that the BEGIN/END count is correct; if not, the user is told, for each such program area, that there are n too many BEGIN's or n too many END's. In order to determine more exactly just where the problem occurs, he can request that a BEGIN/END count be made for any program segment. When the error is narrowed down to a reasonably small program segment, he can request that segment to be displayed for visual checking. Unfortunately, because of the editing freedom permitted, a correct BEGIN/END count guarantees only that the number of BEGIN's and END's is correct and not that their order is logical. Here again, the final judge is the compiler.

The syntax analyzer was implemented using syntax-oriented techniques. These techniques have been used to construct compilers and have been well documented. Most of the error detection done by IPSS is on an individual statement basis. The errors that are detected are then analyzed by a syntax analyzer which determines the class of the error detected. The syntax analyzer contains subroutines that parse the statement symbol string and then analyze it using an encoded version of JOVIAL syntax equations. The same technique is used to analyze the IPSS commands. It would be relatively straightforward to change the syntax analyzer to handle another language that could be described in syntax equations. This technique also simplifies the task of changing the IPSS command structure.

Display hardware interface consideration

As stated before, one of the primary objectives of the IPSS project was to include the use of a display scope as an interactive device between a time-shared computer operating system and a user. As usual, there were many display devices available, and their characteristics (particularly their computer interfaces) were quite varied. These characteristics, which are discussed later, exerted considerable influence on the proposed use of a display in IPSS. In an attempt to reduce any delays of the project, an intermediate language was developed that isolated device-dependent computer characteristics through the proposed IPSS design. The language made it possible to plan and implement information manipulations on an available display before the operational display device was chosen.

Development of an intermediate display language that would be easy to use but not restricted to any particular display device proceeded naturally. After looking at descriptions of several displays that were being considered, one common characteristic (which was so dominant that it was almost overlooked) was noted. This characteristic was also an essential part of the human-to-computer interface. The characteristic was simply the ability to read and write in a language already known to the human. The only adequate and completely versatile means of communicating between a human and a computing machine is with words and numbers. If a display can accept and show a written language that is intelligible to the user, there is no limit to the type or amount of information that can be ex.
changed. Each of the displays being consider for use had this capability. With this one characteristic in mind, a language was developed assuming only that a display device would be chosen that would have the ability to display alphanumeric information in a gridded form (a number of lines of characters); that the device would also accept external inputs from a typewriter keyboard; and that it would send and receive the displayed data to and from a controlling computer.

An intermediate display user's language (IDUL)

The Intermediate Display User's Language (IDUL) is a tool for defining the format and content of a display device as specified by the program designer. This is done by dividing the view screen of the display into fields of information with one or more characters of displayed information per field. Fields are located by line and starting column; their lengths are established by field sizes; associated fields are grouped together with block numbers, and information to be displayed is defined by its location or address. This language is not a communication device between the outside user and the machine. Such communications are a function of each particular application—like IPSS—and should be left to the computer program designer's discretion. IDUL is concerned with internal communication. It allows the program designer to establish a display format and to design information exchanges without the burden of considering display specifics.

A language is a communication tool that is used to exchange information. In the case of IDUL, the language does not have the complex form of a natural language. Instead, it is composed of tabular data that have accepted meanings. Figure 7 shows a typical initial display view that can be used to explain the language forms. Notice that this display has 20 lines of 40 columns or positions which can be described as 22 fields, shown in Table I.

It is convenient to group the 22 fields into three separate blocks: (1) communications or control block, (2) input block, and (3) heading and data display block.

Remember, the display specification language is a programming convenience. The program designer uses it to effect selected information displays in predetermined formats. Therefore, device-specific routines to generate a display or to read indicated display fields as defined by internal tabulations must be available. The programmer sets up the tables and calls the proper routine to perform either the generation or reading of a display. This fact places an importance on the order and structure of the display definition tables.

Table II is a Format Definition Table that shows the fields that were defined in Table I in the IDUL tabular form for format definition. The fields are entered in the table in block, line, column order. The display input fields have zero entries under the "Location of Data" tabular column to indicate that the fields should initially contain blanks. The entries for heading fields in the same column ("Location of Data") show the actual information to be displayed, preceded by the code, LOC, which is an abbreviation for "location." In the internal format definition table, the variable data locations for the heading data, entries 7 through 11, would be inserted by the controlling computer program. The displayed contents of the last ten lines will change as the data file is enlarged; the data descriptions for these fields must be continually updated by the using program. The data fields most frequently read are put in block 1 of the table to make it more accessible. The next most frequently read fields are put in block 2, and all data to be displayed but not altered by the viewer are put in block 3. The block assignments are completely arbitrary with the exception of block 3 fields. Any field with a block number of 3 is protected from external change if this is possible on the display equipment. Block 3 was used as the protection indicator to allow the user more freedom in his block assignments while maintaining an ordered sequence to the field definitions. It is unlikely that any program designer will need more than two blocks to identify input fields, and there is no need to have more than one protection flag. Besides, if the designer wants more field blocking, he can always put the protected field definitions first and start his data blocks with block 4.

The last entry in the format definition table (Table II) is a cursor location. Column 18 of line 3 in Figure 7 is underlined. The underline mark is commonly used by display manufacturers to indicate for the display user's benefit where the next input character will go when he hits a keyboard character key. This mark (universally known as a cursor) is not always an underscore mark. On some displays it is a vertical line to the left of the character position. However, its function is always the

FIGURE 7—Typical initial display
same. It points to the next input position. The cursor is a valuable aid, but it is not essential; information exchange could take place without it. (The operator might have a little trouble getting his data in the right place, but he could do it.) IDUL does provide for cursor insertion, as the last entry indicates. It can be recognized as a cursor definition by its position in the table (last) and its zero field size.

The Intermediate Display User's Language (IDUL) is simply a table containing a field-by-field definition of a display view, where each field description contains a block number, line number, starting column, field size, and data location. Flexibility is obtained by establishing some rules for the use of block numbers and by allowing modifications of the master or original format definition with an overlay specification table. This table

<table>
<thead>
<tr>
<th>FIELD</th>
<th>LINE</th>
<th>COLUMN</th>
<th>SIZE</th>
<th>DESCRIPTION OF FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>Communications or control data</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>16</td>
<td>Heading</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>18</td>
<td>8</td>
<td>File name input field</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>38</td>
<td>Heading</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>37</td>
<td>1</td>
<td>Code input</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>29</td>
<td>7</td>
<td>Heading</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>34</td>
<td>1</td>
<td>Code input</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1</td>
<td>20</td>
<td>Heading</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>1</td>
<td>36</td>
<td>Data input</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>1</td>
<td>36</td>
<td>Data input</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>1</td>
<td>36</td>
<td>Data input</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>5</td>
<td>17</td>
<td>Heading</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>1</td>
<td>40</td>
<td>File data display</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>1</td>
<td>40</td>
<td>File data display</td>
</tr>
</tbody>
</table>

TABLE I—A field by field description of typical display in Figure 7
which is identical in content to the format definition
table) is used to temporarily change one or more entries
in a format definition table and to identify fields to be
read from the display. Fields can be deleted by simply
zeroing the size, or—if this is confusing—by replacing
the block, line, and column numbers with zero, which is
an undefined use. Also, block expansion can be pro-
vided by leaving an appropriate number of undefined
field entries in the tables between blocks.

Device-specific routines to interpret the tabular lan-
guage can be built and inserted at any time. This allows
display device changing with minimal delays, and it
makes it possible to support more than one display type
through internal selection of conversion routines.

### Display characteristics

Required insight into the problems of implementing
routines to transcribe such a display definition language
was gained through an examination of some display

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>BLOCK</th>
<th>LINE</th>
<th>COLUMN</th>
<th>SIZE</th>
<th>LOCATION OF DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>37</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>34</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>16</td>
<td>LOC(ENTER FILE NAME:)</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>38</td>
<td>LOC(INDICATE DESIRED OPERATION: COMPOSE )</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>29</td>
<td>7</td>
<td>LOC(EDIT )</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>20</td>
<td>LOC(ENTER NEW DATA HERE:)</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>17</td>
<td>LOC(DATA NOW ON FILE:)</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>20</td>
<td>1</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>0</td>
<td>0 (Cursor Location)</td>
</tr>
</tbody>
</table>

**TABLE II—Format definition table for typical display**
characteristics. The three displays discussed here are hypothetical, rather than real. This was done to incorporate as many characteristics of real displays into as few examples as possible.

The hypothetical displays have several common characteristics. Each one has a buffer storage where the display image is retained for automatic periodic regeneration of the picture; each one has a cursor that points to the next input position; each one has an input/output path to a computer; and (as pointed out before) each one displays character information in rows. The actual data transfer technique is not important in this discussion. Data transfer is a function of the sending and receiving equipment, and is—therefore—different in every combination. Yet, the capability must be there. Likewise, the buffer storage medium is not as important as the fact that the buffer is there. Some input/output restrictions are caused by the buffer hardware; these are important. (These restrictions are indicated in the individual display discussions.)

The three typical displays do not cover all of the possible display characteristics. They do exhibit a representative set of visual display features.

Table III lists six characteristics of three hypothetical display devices and an ideal display. The first device has a maximum of 40 lines of up to 50 characters per line. Its data storage buffer has space for 1000 characters of data that includes the format control characters. Thus, the amount of information that can be displayed is limited by size of the storage buffer, not by the row and column limits. Also, note that the buffer storage units are not addressable. The entire buffer must be reloaded to change the picture, or all preceding data must be read to read a field of data. Since the device has no computer interrupt signal, the computer program must poll the display to determine when data are ready for transmission from the display storage to the computer storage. This means that the supporting program must periodically send a message to the display asking for control data. As indicated above, format controls are required to position the information on the screen in the required positions. These characters (such as carriage return, horizontal and vertical spaces, insert cursor, enter format mode, return to start, etc.) occupy space in the display buffer storage, but at the same time they eliminate the need for storing blanks to fill fields and lines, and they protect data from keyboard change. The cursor controls are limited in the sense that the cursor position cannot be determined by the computer. It can be positioned, but it cannot be located if it has been moved.

The second display device is more versatile. It can

### Table III—Characteristics of hypothetical display devices

<table>
<thead>
<tr>
<th></th>
<th>MAX. NO. LINES</th>
<th>MAX. NO. COLMS.</th>
<th>BUFFER SIZE AND ACCESS METHOD</th>
<th>I/O CONTROLS</th>
<th>FORMAT CONTROLS</th>
<th>CURSOR CONTROLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>50</td>
<td>1000 characters of non-addressable storage</td>
<td>Computer-initiated full buffer with polling required</td>
<td>All displayed fields are relative to a starting position controlled by stored formatting characters; protection provided</td>
<td>Cursor can be inserted but cannot be found by computer</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>70</td>
<td>4000 or 8000 characters of addressable storage</td>
<td>Console interrupt signal available</td>
<td>Display positions are controlled by light beam position and display orders; protection provided</td>
<td>Cursor can be inserted into any buffer position and its position can be determined by the computer program</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>40</td>
<td>800 characters of addressable storage plus space for format controls</td>
<td>Console interrupt available</td>
<td>Each storage position is displayed in a fixed display position; protection provided</td>
<td>Cursor controlled by Cursor Address Register that can be set or read by the computer program</td>
</tr>
<tr>
<td>IDEAL</td>
<td>40</td>
<td>80</td>
<td>3200 characters of addressable storage</td>
<td>Console interrupt; partial read/writes from any location</td>
<td>Fixed storage-to-display positions relationship; data protection</td>
<td>Computer-accessible Cursor Address Register</td>
</tr>
</tbody>
</table>
display a full 50 lines of 70 characters per line; it has 4000 or 8000 characters of addressable buffer storage; it can be signaled to read through a computer interrupt key; display positions are set by positioning the light beam; fields can be protected; the cursor can be inserted at any displayed position; and the computer can determine the cursor location. This more flexible device can also draw lines or vectors between grid points, but this capability is not very common, so it is not handled by the basic IDUL. It can be utilized with simple additional code, but the resultant program becomes restricted to the device.

The last hypothetical display device is more direct and easier to use than the other two. It has 800 fixed storage positions that correspond to the 20 lines of 40 characters per line of displayed data. The first buffer storage character is displayed at line 1, position 1, the forty-first character is displayed at line 2, position 1, etc. This display can signal for computer action with an operator key, and the cursor is controlled by a cursor address register that can be set or sampled by the computer program. The displayed data can be protected from external change.

Of course, these typical display devices do not contain all of the features that are now available. They do point out some of the major differences that IDUL routines or any display support package must consider. The language translator that transcribes the display definitions into computer code must be able to recognize signals from the individual displays; it must convert the tabular definitions into forms acceptable to the display; it must convert data from the display (and in some cases reformat the data) into a form usable by the computer program; it must control the input/output operations; it must be able to effect data protection, and the Intermediate Display User’s Language Translator must provide a mechanism for setting cursor locations.

The characteristics for the ideal display were selected to simplify the translation of the Intermediate Display User’s Language into computer code. The fixed relationship between the storage position and display positions makes it possible to change or read all or part of the display. This eliminates the transmission of redundant information between the computer and the display. The absolute cursor controls and keyboard interrupt provide for immediate location of communication areas with minimal computer testing. And the character-by-character protection of displayed information permits maximum flexibility in the selection of characters for protection. An IDUL-to-machine-code translator could be quickly and easily written for these ideal characteristics. The described Intermediate Display User’s Language has been implemented for the Interactive Programming Support System using an IBM 2250 display operating under the ADEPT-50 Time-Sharing System. The 2250 display is not the one that will ultimately be used. Therefore, new routines will be required to interpret the basic IDUL, but the many display views set up by the IPSS routines will (we hope) remain intact, with each view defined in terms of its field length and display view position by line and column as stated in the Intermediate Display User’s Language tables.

CONCLUSION

We have found that a display scope for program composition and editing is preferred over a typewriter-like terminal by most users. This preference has also been noted in THOR. This perhaps reflects the fact that typing is a skill not necessarily possessed by all programmers.

None of the displays we have examined are completely suitable for the requirements of IPSS. In addition to the hardware characteristics noted in Table III, an ideal display should be large enough (say 40 lines with 80 characters each) to contain a reasonable sample of data. We have found that the programmer needs a minimum of 20 lines in block 3 alone, in order to correlate displays of several kinds of information. Two final requirements are that the display device must be compatible with computer input/output hardware and the device must be inexpensive.

Syntactic analysis combined with editing gives the programmer two powerful programming aids. He is assured that his edited program is syntactically correct by being immediately advised of errors. He has access to dynamically current program information. IPSS has just begun to develop techniques to process this information and present it in a form useful to the user and thus offer him a significantly improved alternative to personally extracting similar information from program listings.

REFERENCES

1 L P DEUTSCH  B W LAMPSON  
Reference manual QED time-sharing editor
Document No 30 60 30 University of California Berkeley  
November 1965
2 The compatible time-sharing system-A programmer’s guide
The MIT Press Massachusetts Institute of Technology  
Cambridge Massachusetts
3 J McCARTHY et al
THOR-A display based time-sharing system
Proceedings Spring Joint Computer Conference 1967  
pp 623-633
4 R W FLOYD
The syntax of programming languages—A survey
pp 346-353