MSPM SECTION BZ.10.01

Identification

APL Parser

Purpose

The Parser is the main control program of the Multics APL interpreter. The parser reads the input lines typed by the user, calls for them to be lexically analyzed, parses them, interprets them, prints results, and returns to read again. Special input lines; i.e., system requests beginning with ")" and definition mode entry beginning with nabla; are detected and properly routed.

Operation--Initialization

Operation of APL begins with initialization coding performed once per invocation of APL at the entrance to the parser. This consists of:

- setting the current and normal i/o streams to "user_input" and "user_output"; setting the diverted i/o streams to null.
- turning off the interrupt handling flags:
  - inh: set when interrupts are to be inhibited (critical variables are being manipulated).
  - intpend: set when interrupt occurs when inhibited; interrupt will be signalled again when inhibit is turned off.
  - alm: set when interrupt has occurred, the i/o buffers have been reset, further output is suppressed, and the alarm clock is set for 0.5-sec real-time signal; operation continues, hoping for a beginning-of-line to be reached by the interpreter (clean stop), otherwise the alarm will produce a dirty stop (and wipe the state indicator back to the previous suspension).
  - almpend: set when alarm went off when inhibited; it will be re-signalled when inhibit is turned off.
- a unique identifier is created to suffix all working segments required by this invocation of APL.

- a clear work space is created; this consists of creating in the user's process directory the segments "apl.stack.?", "apl.symbol.?", "apl.function.?", and "apl.value.?"; the symbol, function, and value areas will contain no entries, the stack will start with one console suspension frame (S-frame, see discussion of state-stack below).
- a loop is entered to process each command argument to the APL invocation; arguments are APL system requests, and they are processed the same as such requests read from the input stream, with the following exceptions:
  - any errors detected before all arguments have been processed are reported on "user_i/o", and the APL command terminates.
  - the i/o stream switching requests which wold normally establish diverted i/o streams instead affect the normal i/o streams when given as command arguments (see the discussion of i/o stream switching later on).
- a handler is established for the "program interrupt" condition; the handler is responsible for manipulating the interrupt switches as discussed above.
- "APL" is typed on the current input stream.
- the parser enters its main loop, to listen for user lines.

I/O Stream Switching

Three pairs of i/o streams are meaningful to the APL interpreter. They are known as the normal i/o streams, the diverted i/o streams, and the current i/o streams. The current i/o streams are the ones actually used by the interpreter to do its input/output. At any moment, the current i/o streams will be synonymous with either the normal streams, the diverted streams, or "user_i/o".

At the time APL is invoked, normal i/o streams are established, and the current streams are synonymous with them. The normal i/o streams will be attached to "user_input" and "user_output", unless the stream-diverting requests (")FI", ")FO", etc.) are read as command arguments, in which case they will be attached to the indicated files. The attachment of the normal streams will then remain unchanged for the duration of the APL invocation.

While APL is running, detection of a stream-diverting request will establish a diverted stream, the stream will be attached to the indicated file, and the current stream will be made synonmous with it. The normal stream will remain attached, but will not be accessed as long as the current stream remains switched to the diverted stream. Detection of a restore request (")RI", "RO", etc.) while the current stream is switched to the diverted stream, or end-of-information on the diverted stream, will cause the diverted stream to be detached, and the current stream will be identified with the normal stream again.

Detection of any error will cause the current streams to be switched to "user_i/o" so that the error comment and reply will always go to the user's terminal. The user may restore the former stream-switching when he desires with the restore requests. The restore request, when the current stream is switched to "user_i/o", will switch it back to the diverted stream if one has been established, or else back to the normal stream. Of course, the user can cancel his diverted stream before issuing the restore if he desires to return directly to the normal stream.

When APL invokes itself via the "uAPL" built-in function, it creates input and output files and supplies the necessary command arguments to itself.

Operation--Termination

Execution of an invocation of APL can be terminated by an error in the command arguments, the ")OFF" or ")CONTINUE" requests, or the detection of end-of-information on the normal input stream. The termination sequence is:

- the "program interrupt" condition handler is destroyed.
- the working segments in the process directory are deleted.
- all i/o streams attached are detached.
- return is made to caller of APL.

State Indicator Stack

One constituent of the workspace is the stack which mechanizes the state indicator. The stack for the current workspace is maintained in segment "apl.stack.?" in the process directory.

There are three different kinds of frames possible in the state-stack: S-frames (suspension), F-frames (function), and E-frames (evaluated input). When APL is running in the immediate execution mode, an S-frame will be at the top of the stack. If a console line makes a function reference, the evaluation of the console line will be held in the S-frame and an F-frame will be spawned to handle evaluation of the function. If that function calls another, again another F-frame will be produced. When the function returns, its F-frame is popped and the interpreter resumes processing the new stack top. In the event of an error during function execution, a new S-frame will be created, and previous F-frames will remain in the stack. A call for evaluated input causes an E-frame to handle the input line (character input need not be parsed, hence requires no stack frame).

The constituents of a stack frame are:

- its designation as S, F, or E.
- if F, a pointer to the usage bead of the associated function.
- if F, the line location counter (the number of the line currently being processed).
- if S or E, the lexical tokens of the typed line.
- the token location counter (token within the line currently being processed).
- if F, the local variable list (the list of variables which were inserted into the symbol table when this function was called and which must be removed when it returns).
- the parse stack for use by the reductions analyzer. This stack records the state of the parse and interpretation of the current line, including pointers to all temporary values. Sufficient information is remembered here to permit deallocation of all temporary space in the event that execution is interrupted at any arbitrary time and the frame discarded.

Operation--Main Loop

The main path of control in APL is the line processing loop in the parser. Fundamentally, the parser reads a line, lexically analyzes it, parses and interprets it, and returns to read the next line.

In somewhat more detail, the line loop involves the following:

- the "almset" switch is interrogated to see if the user has requested an interrupt since the last pass through the line loop. If so, see the discussion on interrupt processing below.
- if this frame is a S- or E-frame then:
  - output the S- or E-go-ahead characters.
  - read an input line. If end-of-information, restore normal input stream or terminate APL. If character error, type message and return to beginning of main loop.
  - if the line begins with nabla or right parenthesis, give up control to the editor or request processor respectively.
  - switch input streams if the ")FIA", ")RIA", ")FIOA", or ")RIOA" requests are pending.
  - lexically analyze the line, placing the result in the current stack frame.
- on the other hand, for an F-frame:
  - verify that the line location counter is within the function; else, function return to the calling frame.
  - check for stop-control on the current line, type message and suspend if so (spawn an S-frame on top of the stack).
- reset parse state to null, set token location counter to the first token, set parse rule pointer to the first rule.
- enter the reductions analysis loop:
  - if the current parse rule is the error rule, type a syntax error message identifying the first unprocessed token,

spawn an S-frame if current frame is an F-frame, and return to the top of the main loop.

- if the current input and parse state do not match the current rule, go to the next rule.
- if they do match, perform the actions associated with the current rule, then jump to the rule which handles the category currently at the top of the reduction stack. The actions associated with rules take care of: detection of end-of-line, branches, function calls, evaluated and character input requests, and most errors.

At this point,the reader is advised to read MSPM Section BZ.10.02, which describes in detail the formal syntax of Multics APL source lines, and contains the exact reduction rules used in the reductions analysis.

Implementation of Reductions Analyzer

The reductions analyzer is a small interpreter which is driven by the (fixed) rule table. The formal of the rule table is:

```
1 rules(1:146),
  2 action label,
  2 pattern,
    3 basic bit(36),
    3 stack(1:7) bit(36);
```

where "action" is initialized to a label on the particular actions associated with this rule, and "basic" and "stack" are 36-bit masks which look for a corresponding bit in the type-field of the token or stack entry to be matched. An all zero mask signifies end-of-list. Each syntactic type is assigned a unique bit of the mask.

The action coding takes care of both the syntactic actions required (executing the PULL, DONE, ERROR, or promotion to a higher category) and also the semantic actions (the actual interpretation on-the-fly of the meaning of the parsed construct).

F-frame; relative to "apl.stack.?" if this is an

The format of the reductions stack is:

```
1 rstack(1:infinity),
   2 type bit(36),
   2 source offset,
   2 usage offset;
```

where "type" is as discussed above, "source" is the relative offset of the lexical token which originally generated this stack entry (relative to "apl.function.?" if this is an S- or E-frame) (used to access the spellings of names, and also to access the character position within the input line in case of error diagnostics), and "usage" is the relative offset of the usage bead for this function or variable in the symbol table (relative to "apl.symbol.?") (used for accessing its value), or for VALs, CONs, EXPs and LISTs, the offset of the value itself (relative to "apl.value.?"), or for operator classes, which operator exactly. The reductions stack grows from the current top of the state-indicator stack (directly above the local variable list of an F-frame, or the token list of an S- or E-frame).

<u>Interrupt Processing</u>

This section explains the processing necessary to support the "program interrupt" feature of APL. The user may signal "program interrupt" at any arbitrary time and expects APL to abort any possibly lengthy calculation.

In order to recover from an interrupt in the cleanest possible way the four interrupt handling switches already described above (see "Operation--Initialization", this section) are provided.

The first requirement of the interrupt processing is to maintain the integrity of all pointers and storage allocations in the work-space. To do this, aborts cannot be allowed to happen in the middle of critical pointer manipulations, while the data-base is temporarily malformed. Since programmer-defined interrupts cannot be inhibited in PL/I, an inhibit switch is set up in the APL interpreter to serve the same purpose: if the actual interrupt occurs while the inhibit switch is set, no processing of the interrupt will take place at the time, but instead it will occur when the inhibit switch is turned off.

The next requirement of the mechanism is that it stop at the beginning-of-line point in interpretation whenever possible. This is because of the APL definition requirement that suspension always occur between lines; if a line must be aborted in the middle, then the state-indicator must be popped all the way back to the previous S-frame for the stop, with consequent loss of the user's calculations. To allow the current calculation to run to beginning-of-line whenever possible, the interrupt processor does not abort immediately; it sets an alarm clock to go off in 0.5-sec of real time. If no beginning-of-line has been reached before the alarm goes off, the necessary clearing of the state-stack will take place.

The actual sequence of events which occur upon the "program interrupt" condition are:

- If the "inh" switch is set, the "intpend" switch is set and immediate exit is made to the point of interruption.
- If "almset" is set, immediate exit (the alarm is already set from a previous program interrupt).
- Inhibit.
- Reset all console output buffers.
- Enable the alarm clock for 0.5-sec real-time signal.
- Set "almpend". When "almpend" is set, all further output will be inhibited by the console output routine, and also the interpreter main-loop will abort cleanly on beginning-of-line.
- Reset "intpend".
- Reset "inh" (this may signal the alarm condition, see below).
- Exit to point of interrupt.

Resetting the inhibit switch is always done by a subroutine which checks if any signals occurred while interrupts were inhibited. If so, they are signalled again:

- The inhibit switch is cleared.
- If "almpend" is set, signal alarm.

- If "intpend" is set, signal program interrupt.
- Exit to caller.

The sequence when the alarm clock goes off is:

- If "inh" is set, set "almpend" and exit.
- Inhibit.
- If currently in function-definition mode, exit to the editor to listen for the next editing request.
- Wipe the current line being interpreted out of the current state-frame, deallocating any temporary values from the value area in the process (pointers to these values are in the VAL, EXP, and LIST beads in the reductions stack).
- Do a non-local go-to to the interrupt coding at the parser's main-loop top (see next paragraph).

The events which occur when "almset" is detected at the top of the parser's main-loop are:

- Inhibit.
- Turn off the alarm clock.
- Reset "almset".
- If the present frame is an F-frame, spawn an S-frame on top of it. If it is an E-frame, pop successive frames off the stack (deallocating any temporary values associated with them) until an S-frame is encountered.
- Switch i/o streams to "user_i/o".
- Reset "intpend" and "almpend".
- Reset "inh".
- Type "APL".
- Go back to the top of the main-loop.