Published: 08/30/68

Identification

Syntactic Analyzer
James D. Mills

(Note that the following are Abstracts, which should be replaced by a full description at a later time.)

parse

Function of Entry:

Parse is the first phase of the compiler. It does syntactic analysis of all the source program.

Calling Sequence for Entry:

```
call parse (root);
```

Declaration of Arguments:

```
dcl root ptr;
```

Description of Arguments:

root is a pointer which, when parse finishes, points to a computation tree representing the entire source program.

condition_process

Function of Entry:

This procedure processes the condition prefix lists which may begin PL/I source statements.

Calling Sequence for Entry:

```
call  condition_process(i, conditions, check_ptr, cur_block);
```

Declaration of Arguments:

```
dcl  i  fixed bin(15),
     conditions bit(12),
     (check_ptr, cur_block) ptr;
```

Description of Arguments:

i is the index into token_list. Token_list is the array of pointers to token nodes for the current statement.

conditions is set by condition_process. Each bit represents a condition name and is 0 if off and 1 if on.

check ptr points to a list of nodes representing the identifiers specified in the "CHECK" or "NOCHECK" lists.

cur block points to a node representing the block containing the statement being processed.

convert_if_operator

Function of Entry:

This procedure converts the bit string expression obtained by parsing the expression in an if statement into an expression containing various kinds of jump operators. The purpose is to produce more optimal code.

Calling Sequence for Entry:

```
p1 = convert_if_operator (q, p, l1);
```

Declaration of Arguments:

```
dcl (q, p, l1) ptr;

dcl convert_if_operator external entry (ptr, ptr, ptr)
    returns (ptr);
```

Description of Arguments:

q points to the computation tree which is input to the procedure.

p points to the if_statement_node which will contain a pointer to the computation tree produced by convert_if_operator.

l1 is a pointer to a label node. The jump operator inserted into the computation tree represents a conditional transfer to that label.

The value of convert_if_operator points to the newly created computation tree.

free_tree

Function of Entry:

Free_tree is a recursive procedure which frees all nodes in a computation tree except token nodes.

Calling Sequence for Entry:

```
call free_tree(p);
```

Declaration of Arguments:

```
dcl p pointer;
```

Description of Arguments:

p points to a computation tree containing these nodes:

operator
operand
reference
string reference
tokens

get_block_node

Function of Entry:

This procedure allocates and initializes a block node.

Calling Sequence for Entry:

```
p = get_block_node (block_type, father_block);
```

Declaration of Arguments:

```
dcl  block_type     fixed bin(15),
     father_block   ptr;

dcl get_block_node external entry (fixed bin(15), ptr)
    returns (ptr);
```

Description of Arguments:

block_type — is an integer code indicating the type of block being represented, e.g., internal procedure, on unit, begin block, etc.

father_block — is a pointer to the block node containing the new block node.

The value of get_block_node points to the generated block node.

get_operator_node

Function of Entry:

This procedure allocates and initializes an operator node.

Calling Sequence for Entry:

```
p = get_operator_node (operator_type, number of operands,
                       father_node);
```

Declaration of Arguments:

```
dcl (operator_type, number_of_operands) fixed bin(15),
    father_node ptr;

dcl get_operator_node external entry(fixed bin(15),
    fixed bin(15), ptr) returns (ptr);
```

Description of Arguments:

operator_type is an integer code indicating the type of the operator.

number_of_operands is the number of operands for this operator.

father_node is a pointer to the node which contains a pointer to the operator node being generated.

The value of get_operator_node points to the generated operator node.

get_statement_node

Function of Entry:

Allocates a statement node and fills in the fields of the node.

Calling Sequence for Entry:

```
p = get_statement_node(statement_type, father_block,
    label_ptr, conditions);
```

Declaration of Arguments:

```
dcl  statement_type    fixed bin(15),
     father_block ptr,
     label_ptr ptr,
     conditions bit(12),
     get_statement_node external entry (fixed bin(15),
          ptr, ptr, bit(12)) returns (ptr);
```

Description of Arguments:

statement_type	is an integer value identifying the type of PL/I statement being represented by the node.
father_block	is a ptr to the block node containing the statement.
label_ptr	is a ptr to a chain of label nodes.
conditions	is a bit string coded to indicate which prefix conditions are enabled for this statement.

The value returned is a pointer to the statement node created.

record_context

Function of Entry:

This procedure traces down a chain of label or entry nodes and records label or entry context for each node.

Calling Sequence for Entry:

```
call record_contest$ {labels}  (label_ptr, father_block,
                     {entries}
     {statement_ptr}
     {    null    };
```

Declaration of Arguments:

```
dcl (label_ptr, father_blocks, statement_ptr) ptr;
```

Description of Arguments:

label_ptr is a pointer to a chain of label nodes or entry nodes.

father_block is a pointer to the node representing the block for which the context is to be recorded.

statement_ptr is a pointer to the node representing the statement on which the label occurred. For entries statement_ptr is null.

statement_type

Function of Entry:

This procedure is the key_stone to the parse. It determines the kind of statement currently being processed. All ambiguity is resolved by this procedure.

Calling Sequence for Entry:

```
type = statement_type(index, label_ptr, conditions,
       check_ptr, cur_block);
```

Declaration of Arguments:

```
dcl  index  fixed bin(15),
     (label_ptr, check_ptr, cur_block ptr,
     conditions bit(12);

dcl  statement_type external entry(fixed bin(15),
     ptr, ptr, ptr) returns (fixed bin(15));
```

Description of Arguments:

index — is an index into the token list which contains the statement to be identified. It is set by the caller and reset by statement_type.

label_ptr — is a ptr to a list of nodes representing the labels (or entry names) on the statement. It is set by statement_type.

conditions — represents the condition prefix names on the statement. It is set by statement_type through a call to condition_process.

check_ptr — is a ptr to a list of nodes representing the identifiers specified in a "CHECK" or "NOCHECK" list. It is set by statement_type through a call to condition-process.

cur_block — is a ptr to the node representing the block containing the statement being identified. It is set by the caller.