SYMBOL TABLE: A symbol table is a data structure containing a record for each identifiers. Symbol table allow us to find the record for each record and to store or retrieve data from that record quickly.

Following information will be stored about identifiers.

- → The name (as a string)
- → The data type
- → The block level
- → Its scope (global, local, parametis)
- → Its offset from the base pointer (for local variables and parameters only).

The information stored in symbol table can be used to answer the following questions.

- → storage address.
- → Data type of variables
- → scope of variable.

Symbol table functions:

The two most basic symbol table functions are

- → To insert a new symbol
- → To lookup an old symbol

How to store names in symbol table:

1. Fixed length
2. Variable length.

1. Fixed length: A fixed space for each name is allocated in symbol table.

→ The name can be refered by a pointer to the symbol entry

→ Wastage of space if name is too small.

Names							Attribute
c	a	l	c	u	l	a	
s	u	m					
a							
b							

2. Variable length: The amount of space required by the string is used to store information(names) The name can be stored with the help of starting index and length of each name.

Names		Attribute
starting index	length	
0	10	
10	4	
14	2	

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
c	a	l	c	u	l	a	t	e	$	s	u	m	$	a	$

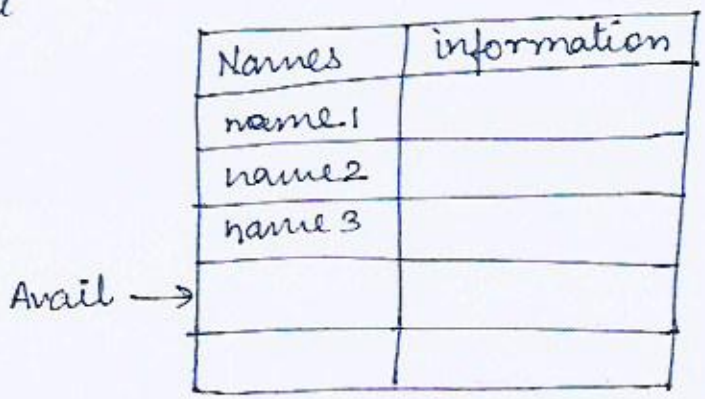
Data structures for symbol table:

- → Linear Linked list
- → Self Organizing list
- → Hash tables
- → Search tree

① List (Arrays)

→ Collection of arrays are used to store name and their associated information.

→ simple to implement



→ new element/name is stored at space pointed by the 'Avail.'

→ for inserting new name first it searches from beginning to avail. if the name is already present it gives "multiple defined names error" otherwise insert the name.

→ fast insertion

→ searching is slow.

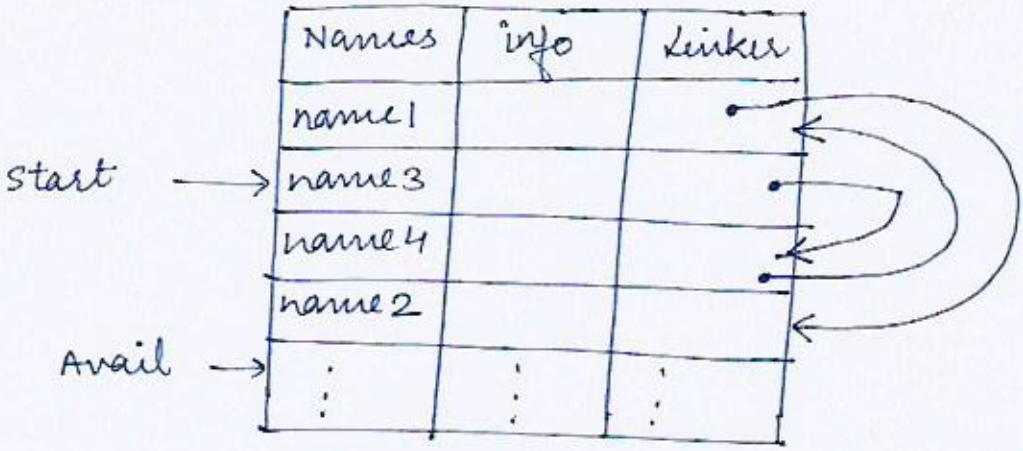
② Self organizing list:

→ Linker (next) field is added to each record

→ 'start' points to the first record of the list of names.

→ most frequently referred names tend to be front of

So the access time to most frequently accessed name will be least.



Hash table:

- A hash table is a data structure that associates keys with values.
- it maintains two tables
  - → hash table (index table)
  - → symbol table

→ Hash table contains k entries from 0 to (k-1). it simply contains pointers to the symbol table entries/names.

→ uses a hash function

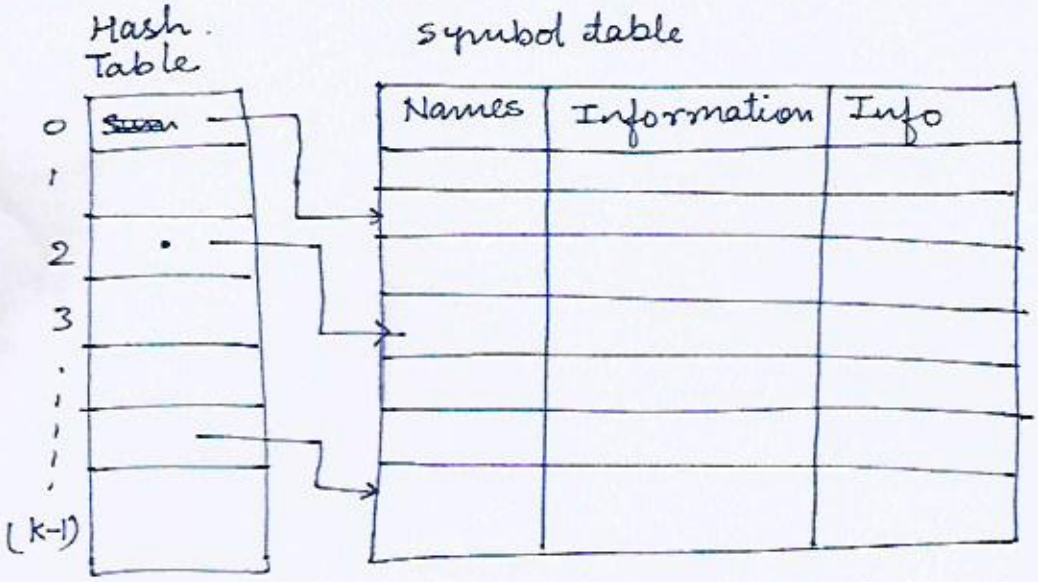
position = hash(names).

→ hash function results values/keys from 0 to (k-1)

→ hash function should be like → uniform distribution
→ minimum no. of collision

→ Searching is fast

→ complex to implement.



Search Trees:

In this method each record contains two link fields along with the information:

① Left
② Right

→ we use these fields to link the records into a binary search tree.

→ All names are created as child of root node that always follow the property of binary search tree.

i.e $name < name_i \Rightarrow$ all smaller names than $name_i$ must be left child of $name_i$.

$name > name_i \Rightarrow$ all larger names than $name_i$ must be right child of $name_i$

→ For inserting any name it always follow binary search tree inserting algorithm.

→ searching is fast

→ Complex to implement

Run time Environment

→ deals with the kind of support a program needs at runtime

→ deals with how the storage is managed at run time in context of compiler.

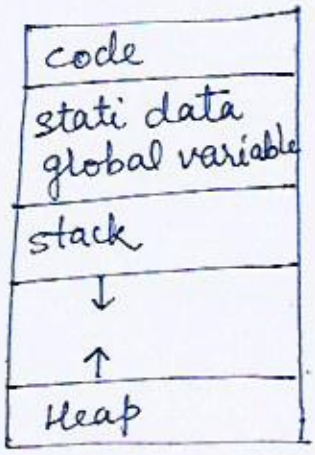
→ An operating system provides a block of memory to every program to be executed (execution of program so known as process). This memory space is called run time storage.

→ This run time storage is subdivided into parts to hold code and data

→ code :→ The generated target code will be stored here.

→ static :→ contains static or global data objects.

Being static, we have to give the size of variable or array (array is static) in the time of compilation, so that OS get to know how much space should be given to the static or global part.



memory layout.

→ when a variable is declared static, it will live throught the program.

→ Heap: Heap is used for dynamic memory allocation (the memory which is given at run time to the data objects (such as malloc or calloc)

→ stack :- stack is required for function calling. When ever a function call is made, then there is something called stack where activation record is pushed.

Each execution of a procedure or function is referred to as an activation.

→ Heap and stack grows in opposite direction. The size of stack and heap is not fixed it may grow or shrink interchangeably during the program execution.

Storage Allocation strategies :

→ There are 3 different storage allocation strategies based on this division of run time storage.
  → static allocation
  → stack allocation
  → Heap allocation

Static allocation :

- The size of data objects is known at compile time
- Names of these data objects are bound to storage at compile time
→ The binding of names with the storage do not change at run time.
→ In static allocation the compiler can determine the amount of storage required by each data object, therefore it becomes easy for a compiler to find the addresses of these data in activation record.
→ Simple to implement

Disadvantages of static allocation :

→ The static allocation can be done only if the size of data objects is known at compile time.

→ The data structures can not be created dynamically.

→ Static allocation can not manage allocation of memory at run time.

→ This type of allocation does not support recursive procedure call.

Stack allocation :

Stack allocation strategy is the type of allocation in which the storage is organized as stack.
This stack is called control stack.

→ Whenever a new activation begins the activation records are pushed onto the stack and whenever the activation ends, the activation record is popped off.

→ The local variables are stored in each activation record. Local variables are bound to fresh storage means if we a variable declared in a function like int a. Then this variable will be created as many times as function is called.

Disadvantages of stack allocation :

→ Memory addressing can be done using pointer and index register. Hence this allocation is slower than static allocation.

→ Local variables can not be retained once activation record ends.

Heap allocation :

→ In heap allocation, heap is used to manage dynamic memory allocation.

→ Data structures and data can be created dynamically.

→ Allocation and deallocation can be done in any order.

→ A linked list is maintained for free blocks.

→ The heap allocation allocates the continuous block of memory when required for storage of activation records or other data objects. The allocated memory can be deallocated when activation ends.
This deallocated (free) space can be reused by heap manager.

Activation Record :

→ Each execution of procedure is refered to as an activation of the procedure.

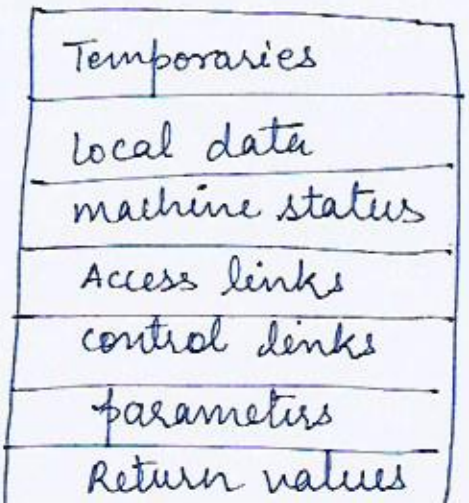
→ Activation record is a block of memory used for managing information needed by a single execution of a procedure.

→ Activation record consists of the following field.

Temporaries : used in expression evaluation

Local data : field for local data

Saved machine status : holds info about machine status before procedure call.



Access link: to access non local data

control link: points to activation record of caller

Actual parameters: field to hold actual parameters

Return value: field for holding value to be returned.

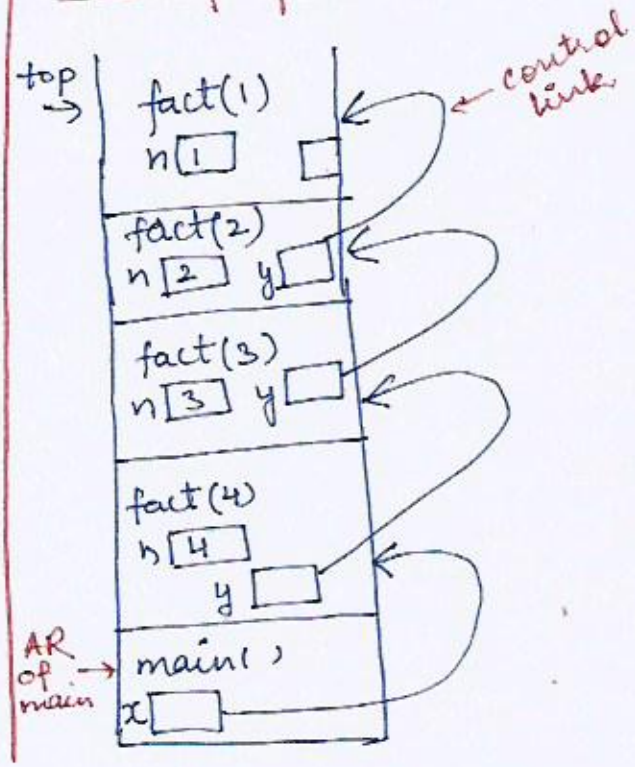
Example

```
void main()
{
1.  int x;
2.  x = fact(4);
3.  printf("%d", x);
}
int fact(int n)
{
4.  int y;
5.  if(n==1 || (n==0)) return 1;
6.  y = fact n* fact(n-1);
7.  return y;
}
```

→ local variables stored in stack.

→

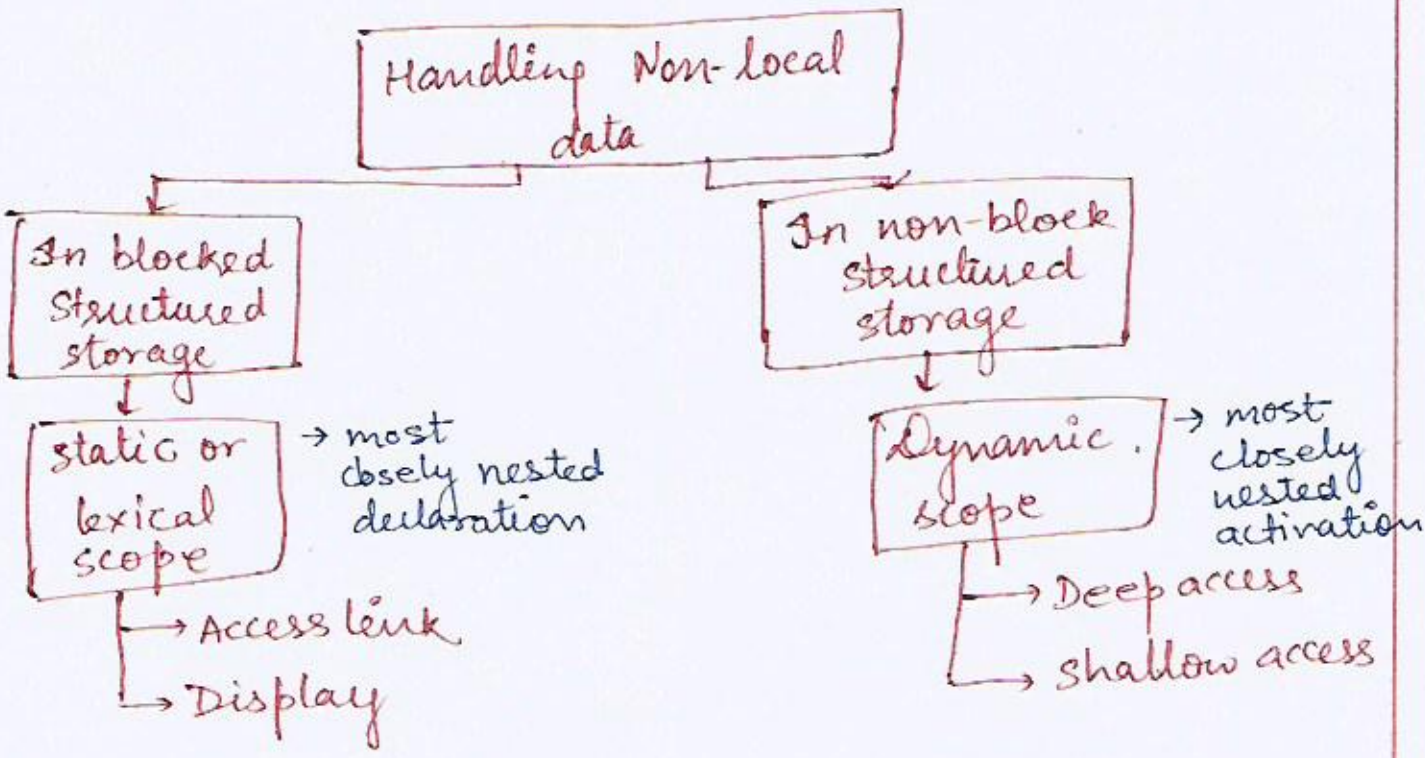
How stack will be implemented during procedure calls.



Block Structure and Non Block structure storage Allocation

→ The storage allocation can be done for two types of data variables.
  → local data
  → Non local data.

→ local data can be handled using activation record whereas non-local data can be handled using scope information



Local data:

→ local data can be accessed with the help of activation record.

Reference to any variable x in procedure = Base pointer to the start of procedure + offset of x from base pointer.

Access to non local variables:

using scope rule

→ static scope ⇒ most closely nested declaration

→ dynamic scope ⇒ most closely nested activation

Block: The block is a sequence of statements containing the local data declarations and enclosed within the delimiters

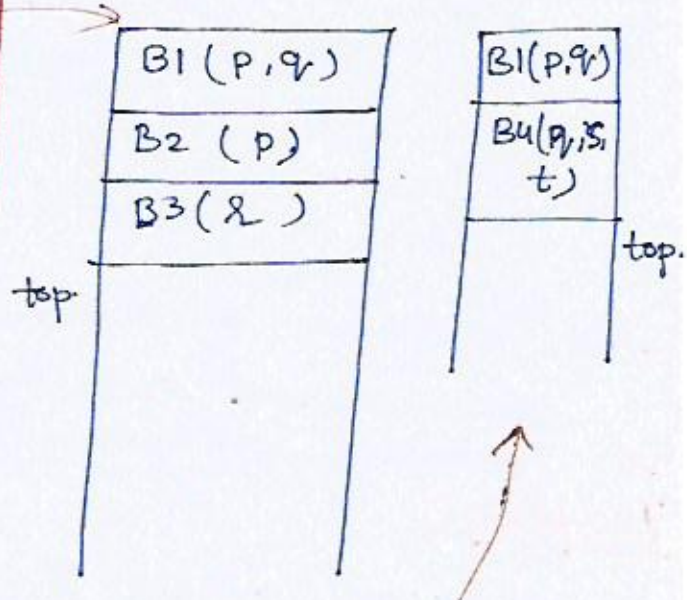
Example

```
main( )
{
  int p, q.
    {
      int p;
        {
          int r;
        }
    }
    ---------
    {
      int q, s, t;
    }
}
```

B1, B2, B3, B4

← The storage will be allocated to the complete procedure main.

→ uses stack for memory location.



Lexical scope can be implemented using

→ Access link

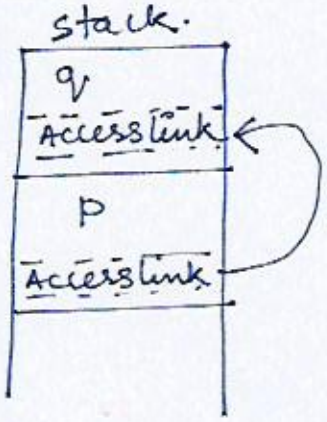
→ Display.

Access link:

→ Include a field 'access link' in activation record.

→ if procedure p is nested in procedure q then access link of p points to access link in most recent activation of q.

```
q()
{
  - - -
    p()
    {
    }
}
```



→ By traversing access link of activation records, non locals can be accessed correctly.

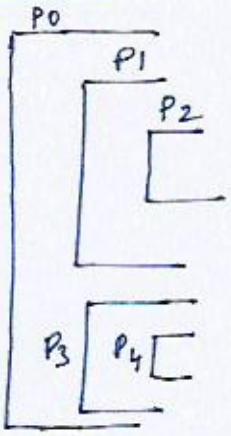
→ Traversal is expensive every time, slows down the speed.

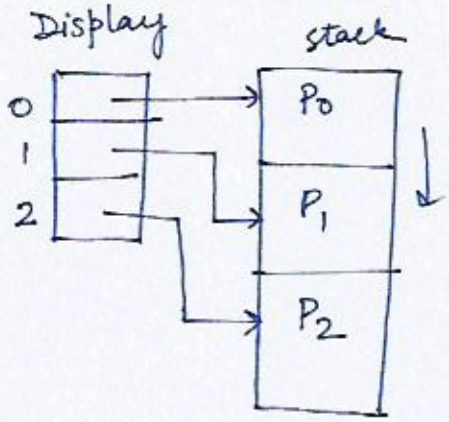
Display:

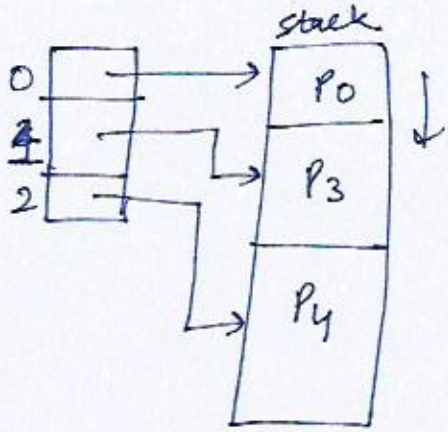
- An array of pointers to activation record is maintained
- Array is indexed by nesting level.

→ The pointers points only to accessible activation records.

Example







comparisons b/w access link and display:

Access link
→ takes more time to access non local variables
→ less space

Display
→ faster
→ takes more space.

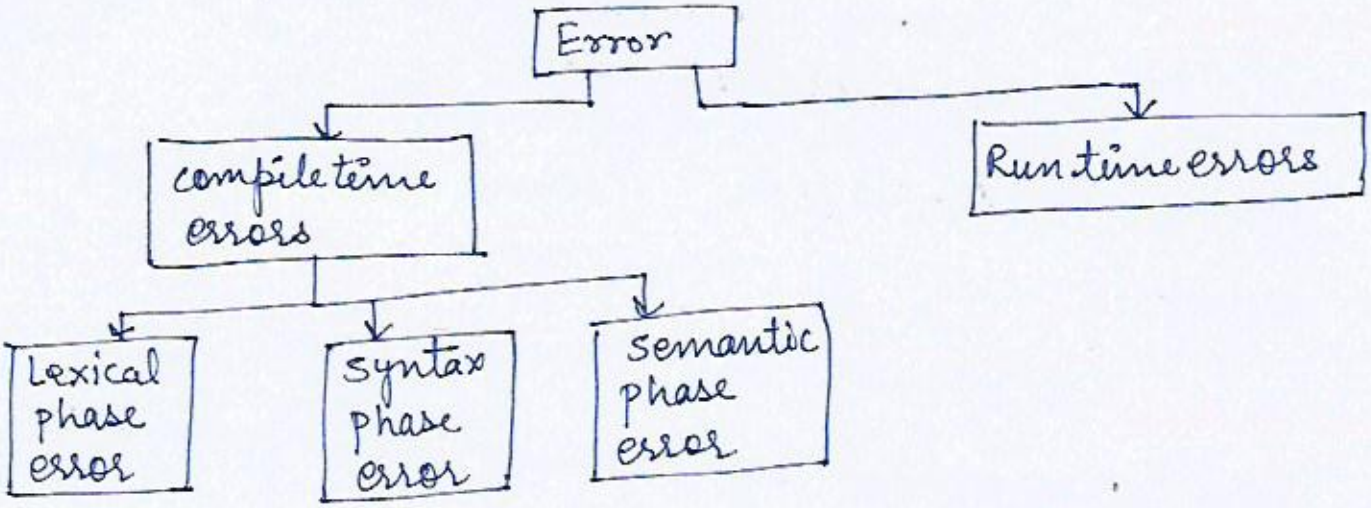
Error Handling:

The process of locating errors in source program and reporting to user is called error handling.

Function of Error Handler:

Error Detection → to identify all possible errors
Error Reporting → to report errors with appropriate message to user.
Error Recovery → to repair errors in order to continue processing of program.

classification of errors:



Lexical phase errors:

- errors occurred during lexical analysis.
- such as - misspelling an identifier.
  - Appearance of any illegal character

- Error Recovery: Panic mode:
  → In this recovery method, successive characters from the remaining input are deleted until the well-formed token is found.
  → to delete unwanted characters occurred.
  → to insert appropriate set of character in order to match string

Syntax Error:

- Errors occurred during syntax analysis.
  such as: errors in structure of the statement
  - arithmetic expressions with unbalanced parenthesis.

Error detection and reporting during syntax analysis:-

→ In parsing table driven passing methods:-
in passing table all blank entries are referred to as error.

FIRST Method: in passing table each error entry for state i is denoted by e_i.

if ParsingTable[i, t] ≠ e_i where $t \in T$ (all valid tokens)

Then report | Error at line ___ : expected tokens are t |

Second method:

→ To reduce error entries by replacing them from r_j entries.
r_j ⇒ reduction by j^{th} production entry in state i.

state	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				Accept			
2	r2	r2	s7	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4					

To fill error entries by reduce entries

shift entries will be same

Third method:
- To maintain a separate table with error messages.
- To fill parsing table with appropriate error entries.

Error	Error Message
e_1	missing operand
e_2	Right parenthesis not matching
e_3	missing operator.

Error Recovery methods:

- Panic mode
- Phrase level recovery
→ Error production
→ Global correction

Panic mode: - skip input symbol without checking additional errors.
- good for less errors.

Phrase level Recovery: - replace the portion of input string by some string
- perform local correction
→ local correction decided by compiler designer.
→ deleting or inserting comma, semi colon,

Error production: - include some error productions.
- if that production is used, report error
- difficult to maintain because change in ~~error~~ grammar.

Semantic errors:

→ errors occurred during semantic analysis.

→ such as incompatible data types; undeclared variables.

Recovery:-

for undefined identifier ⇒ make symbol table entry.

data type mismatch ⇒ do type conversion.

JOIN OUR WHATSAPP GROUP

https://chat.whatsapp.com/H4gUQPBdFet7oNfHaH0OGj