Compiler design

Q.2 role of symbol table

Q.1 Symbol table management with suitable example.

Q.3 organization of symbol table.

→ (1) Symbol table is an important data structure created & maintained by the compiler.

(2) It stores the information about the scope, name, instances of various entities such as variable & function names, classes, objects, etc.

(3) It is built in lexical & syntax analysis phases.

(4) It is used by compiles to achieve compiles-time efficiency.

(5) It is used by various phases of the compiler:-

(a) lexical Analysis :- create new table entry in the table like entries about token.

(b) syntax Analysis :- Add info regarding attribut type, scope, dimension, line of reference.

(c) semantic Analysis :- verify that expression & assignments are semantically correct (type checking) & update it accordingly.

(d) intermediate code generation :- Refers symbol table for knowing how much & what type of run time is allocated.

(e) code optimization :- uses info for machine optimization.

(f) code generation :- Generates code by using address info of identifies present in table.

Date
Page

Symbol table management :-

managing a symbol table involves inserting, searching & deleting entries.

(a) Inserting entries :-

When the compiler encounters a new identifier, it needs to be added to the symbol table. This process involves creating a new entry with the identifier's name, type, scope, & any other relevant information. The insertion process may also involve checking for duplicate identifiers within the same scope & throwing an error if one is found.

(b) Searching entries :-

When the compiler encounters an identifier in an expression or statement, it must look it up in the symbol table. Searching involves finding the relevant entry for the identifier based on its name & scope.

ex statement like $y = x + 1$; the compiler searches for x in the symbol table.

(c) Deleting entries :-

As the compiler processes the program, it may enter and exit various scopes. When exiting a scope, the entries associated with that scope should be removed from the symbol table.

Data structure for symbol tables or various approaches to symbol table organization

(a) Linked list :-

(1) A linear list of record is the easiest way to implement a symbol table. The new names are added to the table in the order that they arrive.

(2) whenever a new name is to be added to the table the table is first searched linearly or sequentially to check whether or not the name is already present in the table.

(3) If the name is not present, then the record for new name is created & added to the list at a position specified by the available pointer.

(4) Insertion is fast O(1), but lookup is slow for large tables - O(n) on average.

(b) Hash table :-

(1) A hash table is a table of k-pointers numbered from zero to k-1 that point to the symbol table of a record within the symbol table.

(2) To enter a name into symbol table, we find out the hash value of the name by applying a suitable hash fn.

(3) The hash fn maps the name into an integer between zero and k-1 and using this value as an index in the hash table, we search the list of the symbol table records that is built on that hash index.
(4) If the name is not present in that list we create a record for name & insert it at the head of the list.
(5) Insertion & lookup can be made very fast - O(1)
(6) The advantage is quick to search is possible & the disadvantage is that hashing is complicated to implement.

(3) Binary search tree :-

(1) Another approach to implement a symbol table is to use a binary search tree i.e. we add two links fields i.e left & right child.
(2) All names are created as child of the root node that always follows the property of the binary search tree.
(3) Insertion & lookup are $O(\log_2 n)$ on average.

Advantage :-
(1) The efficiency of a program increased
(2) better coding structure
(3) faster code execution
(4) Improved code reuse.

disadvantage :-
① Increased memory comsumption
② Increased processing time
③ complexity
④ limited scalability
⑤ upkeep
⑥ limited functionality

Q.4 What are error occur during lexical analysis & diff methods used for error recovery is such cases.

⇒ error occur during lexical analysis :-
① Exceeding the length of identifies or
② Appearance of illegal characters
③ unmatched strings or comments.

Error recovery method for lexical analysis include
① Panic mode Recovery :-
Panic mode recovery involves skipping input until a designed synchronized token is found, enabling compiles to recover from lexical errors & continue parsing.

example:- consider snippet with illegal character.

```
def hello ():
    Printf ("Hello, world! $");
```

The presence of the illegal character '$' at the end of the string triggers a lexical error. using panic mode recovery, the compiler would discard characters until it reaches a synchronizing token, such as a semicolon;

After recovery, the code might look like this:-

```
def hello():
    Printf("Hello, world");  // $ discarded
```

② Global corrections :-

Global correction involves analysing the entire input to identify of correct lexical errors, providing a more comprehensive approach to error handling.

ex

```
#include <stdio.h>
int main()
{ int num = 10;
  Printf("The numbers is : %d\n", num);
  return 0;
}
```

The misspelled fn printf instead of printf triggers a lexical error. with global corrections the compiler might analyze the entire code and suggest corrections.

```
#include <stdio.h>
{ int num = 10;
  Printf("The number is: %d\n", num);
  return 0;
}
```

③ Token Rejection :-

Token rejection involves discarding tokens containing errors while allowing the parsing to continue with the remaining input, minimizing disruption.

ex.

```
function greet()
{ console.log('Hello, world!);
}
```

The missing closing single quote "'" at the end of string triggers a lexical error.
with token rejection, the lexer would discard the incomplete string token & continue processing the rest of the code :-

```
function greet()
{ console.log('Hello, world!');
}
```

④ Interactive error handling :-
Interactive error handling involves interacting with the users or programmer to resolve lexical errors, providing high level of user involvement in error resolution.

ex
Enter a arithmetic expression : 2 * (3+4)) - 5

The extra ')' closing parenthis at the end of expression is a lexical error. In an interactive environment, the compiler might prompt the user to correct the expression :-

Enter arithmetic expression : 2 * (3+4) - 5

Q.4 Overview of YACC with diagram, its purpose & how it is utilized. Explain automatic error recovery in YACC.

→ overview
① YACC stands for Yet another compiler compiler
② YACC provides a tool to produce a parser for a given grammar.
③ YACC is a program designed to compile a LALR(1) grammar.
④ It is used to produce the source code of the syntactic analyzer
⑤ The input of YACC is the rule or grammar and output is a C program.

Date
Page

(a) Gener Grammar specification :-
The first step in using YACC is to define the grammar of the language to be parsed. This grammar is typically using BNF or EBNF

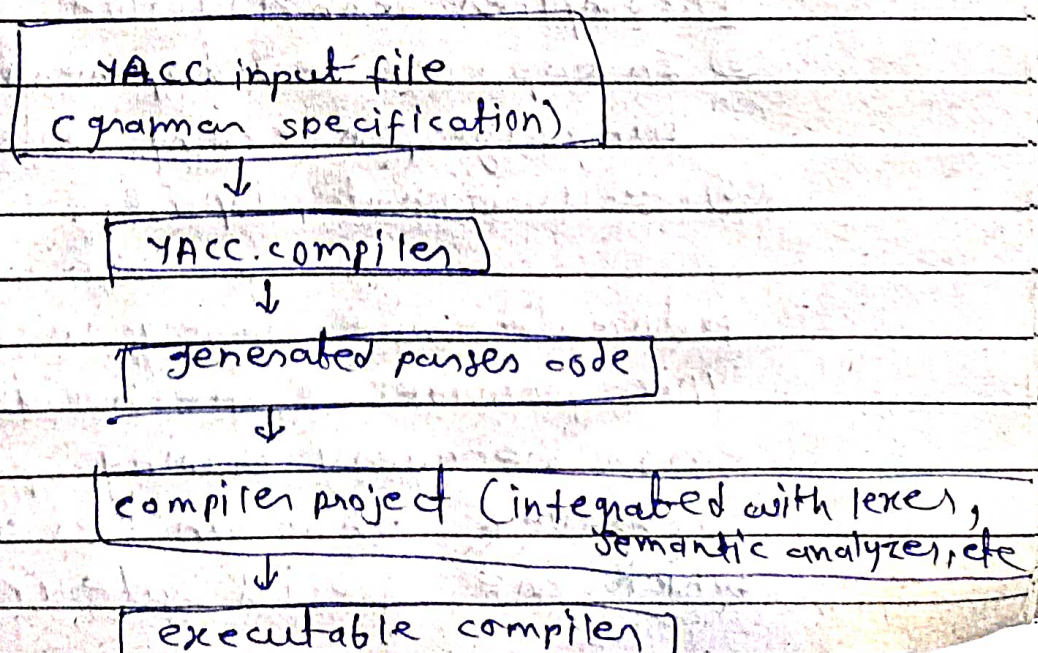
(b) YACC input file :-
The grammar specification is written in a file with '.y' extension, which serves as input to the YACC tool.

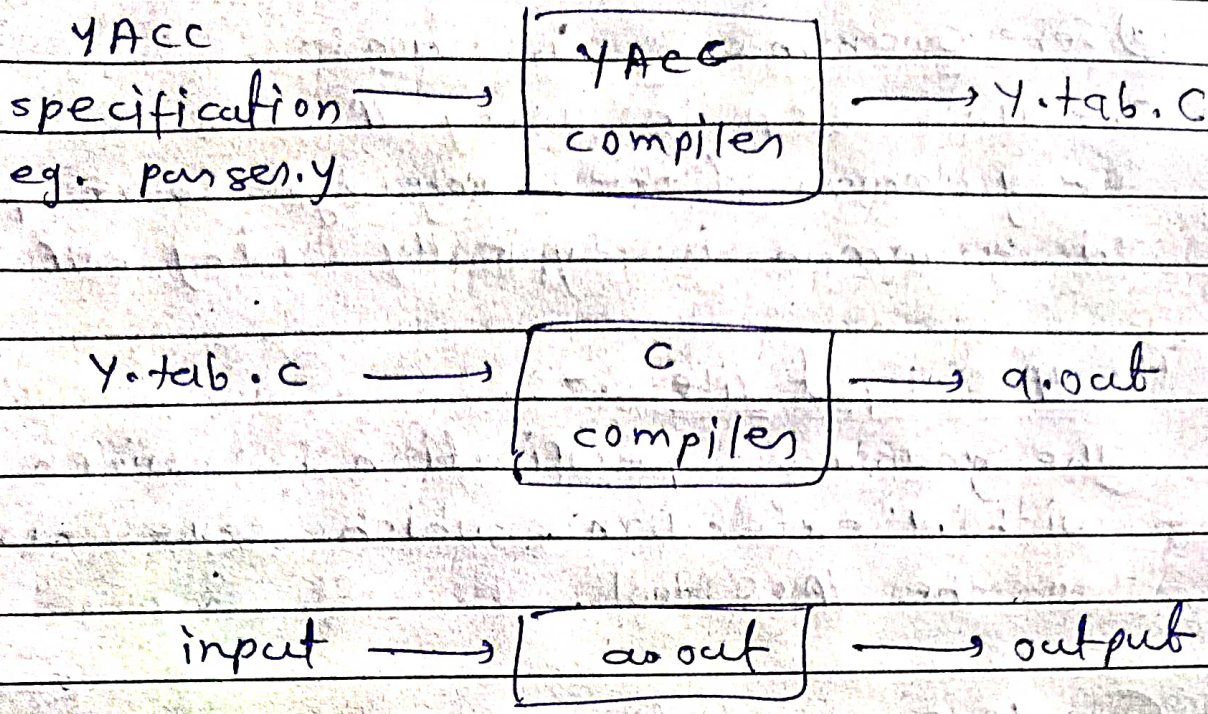
(c) YACC compilation :- The YACC tool processes the input grammar file & generates parser code in C or target programming language

(d) Parser Integration :- The generate parser code is then integrated into the compiler.

(e) compilation :- It includes The generated parser is compiler to produce an executable compiler.

Diagram :-





Purpose of YACC :-

1. Automated parser generation.
2. Language Independence. (Parser koi bhi program language me generate kr skta h)
3. Efficiency
4. errors handling

utilization of YACC :-

1. compiler construction
2. Interpreter development
3. Language processing tools
4. educational purpose

Automatic error recovery in YACC :-

YACC provides a mechanism for automatic error recovery during parsing. This is achieved through error production which are special rule in grammer.

Date
Page

(1) Error production :- Error productions are rules that specify how to recover from a syntax error encountered during parsing. This rule trigger when parser encounters an unexpected input.

(2) Error token :- Error token that represent unexpected input. when parser encounter such tokens it can apply error production to recover from the error.

(3) Error handling actions :- Error production include semantic actions that define how the parser should recover from the error. These actions may involve discarding input tokens, inserting missing token or other strategies to synchronize the parser.

(4) Error msg :- YACC can also be configured to generate meaningful message

Q.5 can impact of error recovery on the overall efficiency of performance of LR parsing.

→ Error recovery plays a singinificant role in the overall efficiency of performance of LR parsing, which is a parsing technique commonly used in the construction of parsers generated by tools like YACC.
LR parsers is powerfull because they can handle a wide range of grammer efficiently.

The impact of error recovery on LR parsing efficiency of performance :-

(1) minimizing disruption :- Error recovery mechanism help LR parsers to minimize disruption caused by syntax error in input code

(2) maintaining parsing speed :- Efficient error recovery technique ensure that LR parsers can maintain parsing speed even in the presence of error.

(3) Reducing memory usage :- LR parsing algo typically require maintaing a parse stack to track the parsing state of facilitate parsing decision.

(4) Ensuring robustness :- Robust error recovery technique enhance the LR parser to handle various types of errors efficient effectively.

(5) maintaining Error Reporting Accuracy :- while error recovery focuses on enabling the parser to continue despite errors, it is also essential to ensure accurate error reporting to aid developers in debugging their code.

Q. Short note :-

* Storage allocation :-

(1) One of the imp task that a compiler must perform is to allocate the resources of target machine to represent the data.

(2) A compiler decide the run-time representation of data object in the source program.

(3) storage allocation strategies
   (a) static storage allocation
   (b) stack —11—
   (c) heap —11—

* Activation Record :-