4
UNIT

Symbol Tables

CONTENTS

PART-1

Symbol Tables : Data Structure for Symbol Tables.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. Discuss symbol table with its capabilities ?

Answer

1. A symbol table is a data structure used by a compiler to keep track of scope, life and binding information about names.
2. These information are used in the source program to identify the various program elements, like variables, constants, procedures, and the labels of statements.
3. A symbol table must have the following capabilities :
   - **a. Lookup :** To determine whether a given name is in the table.
   - **b. Insert :** To add a new name (a new entry) to the table.
   - **c. Access :** To access the information related with the given name.
   - **d. Modify :** To add new information about a known name.
   - **e. Delete :** To delete a name or group of names from the table.

Que 4.2. What are the symbol table requirements ? What are the demerits in the uniform structure of symbol table ?

Answer

The basic requirements of a symbol table are as follows :

1. **Structural flexibility :** Based on the usage of identifier, the symbol table entries must contain all the necessary information.
2. **Fast lookup/search :** The table lookup/search depends on the implementation of the symbol table and the speed of the search should be as fast as possible.
3. **Efficient utilization of space :** The symbol table must be able to grow or shrink dynamically for an efficient usage of space.
4. **Ability to handle language characteristics :** The characteristic of a language such as scoping and implicit declaration needs to be handled.

Demerits in uniform structure of symbol table :

1. The uniform structure cannot handle a name whose length exceed upper bound or limit or name field.
2. If the length of a name is small, then the remaining space is wasted.

Que 4.3. **How names can be looked up in the symbol table ?**

Discuss. **AKTU 2016-17, Marks 10**

Answer

1. The symbol table is searched (looked up) every time a name is encountered in the source text.
2. When a new name or new information about an existing name is discovered, the content of the symbol table changes.
3. Therefore, a symbol table must have an efficient mechanism for accessing the information held in the table as well as for adding new entries to the symbol table.
4. In any case, the symbol table is a useful abstraction to aid the compiler to ascertain and verify the semantics, or meaning of a piece of code.
5. It makes the compiler more efficient, since the file does not need to be re-parsed to discover previously processed information.

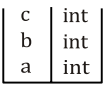
For example : Consider the following outline of a C function :

```
void scopes ( )
    {
    int a, b, c;                    /* level 1 */
    . . . . . .
                {
        int a, b;                   /* level 2 */
      . . . .
      }
      {
        float c,  d;                /* level 3 */
          {
          int m;                    /* level 4 */
          . . . . .
        }
       }
    }
```

The symbol table could be represented by an upwards growing stack as :

i. Initially the symbol table is empty.

ii. After the first three declarations, the symbol table will be



iii. After the second declaration of Level 2.

b	int
a	int
c	int
b	int
a	int

iv. As the control come out from Level 2.

c	int
b	int
a	int

v. When control will enter into Level 3.

d	float
c	float
c	int
b	int
a	int

vi. After entering into Level 4.

m	int
d	float
c	float
c	int
b	int
a	int

vii. On leaving the control from Level 4.

d	float
c	float
c	int
b	int
a	int

viii. On leaving the control from Level 3.

c	int
b	int
a	int

ix. On leaving the function entirely, the symbol table will be again empty.

Que 4.4. **What is the role of symbol table ? Discuss different data structures used for symbol table.**

OR

Discuss the various data structures used for symbol table with suitable example.

Answer

Role of symbol table :

1. It keeps the track of semantics of variables.
2. It stores information about scope.
3. It helps to achieve compile time efficiency.

Different data structures used in implementing symbol table are :

1. Unordered list :

a. Simple to implement symbol table.

b. It is implemented as an array or a linked list.

c. Linked list can grow dynamically that eliminate the problem of a fixed size array.

d. Insertion of variable take O(1) time , but lookup is slow for large tables *i.e.*, O(n) .

2. Ordered list :

a. If an array is sorted, it can be searched using binary search in O($\log_2 n$).

b. Insertion into a sorted array is expensive that it takes O(n) time on average.

c. Ordered list is useful when set of names is known *i.e.*, table of reserved words.

3. Search tree :

a. Search tree operation and lookup is done in logarithmic time.

b. Search tree is balanced by using algorithm of AVL and Red-black tree.

4. Hash tables and hash functions :

a. Hash table translate the elements in the fixed range of value called hash value and this value is used by hash function.

b. Hash table can be used to minimize the movement of elements in the symbol table.

c. The hash function helps in uniform distribution of names in symbol table.

For example : Consider a part of C program

int *x*, *y*;
msg ( );

1. Unordered list :

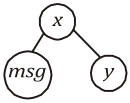
S. No.	Name	Type
1	*x*	int
2	*msg*	function
3	*y*	int

2. Ordered list :

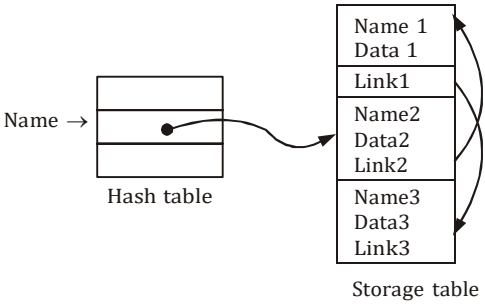
Id	Name	Type
Id1	*x*	int
Id2	*y*	int
Id3	*msg*	function

Id
Id1
Id2
Id3

3. Search tree :



4. Hash table :



Que 4.5. **Describe symbol table and its entries. Also, discuss various data structure used for symbol table.**

AKTU 2015-16, Marks 10

Answer

Symbol table : Refer Q. 4.1, Page 4–2C, Unit-4.

Entries in the symbol table are as follows :

1. **Variables :**
   a. Variables are identifiers whose value may change between executions and during a single execution of a program.
   b. They represent the contents of some memory location.
   c. The symbol table needs to record both the variable name as well as its allocated storage space at runtime.

2. **Constants :**
   a. Constants are identifiers that represent a fixed value that can never be changed.
   b. Unlike variables or procedures, no runtime location needs to be stored for constants.
   c. These are typically placed right into the code stream by the compiler at compilation time.
3. **Types (user defined) :**
   a. A user defined type is combination of one or more existing types.
   b. Types are accessed by name and reference a type definition structure.
4. **Classes :**
   a. Classes are abstract data types which restrict access to its members and provide convenient language level polymorphism.
   b. This includes the location of the default constructor and destructor, and the address of the virtual function table.
5. **Records :**
   a. Records represent a collection of possibly heterogeneous members which can be accessed by name.
   b. The symbol table probably needs to record each of the record's members.

Various data structure used for symbol table : Refer Q. 4.4, Page 4–4C, Unit-4.

PART-2

Representing Scope Information.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.6. Discuss how the scope information is represented in a symbol table.

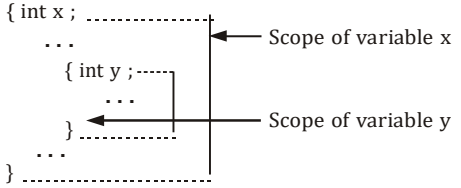
Answer

1. Scope information characterizes the declaration of identifiers and the portions of the program where it is allowed to use each identifier.
2. Different languages have different scopes for declarations. For example, in FORTRAN, the scope of a name is a single subroutine, whereas in

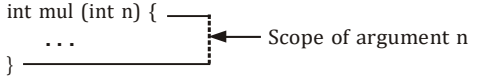
ALGOL, the scope of a name is the section or procedure in which it is declared.

3. Thus, the same identifier may be declared several times as distinct names, with different attributes, and with different intended storage locations.
4. The symbol table is thus responsible for keeping different declaration of the same identifier distinct.
5. To make distinction among the declarations, a unique number is assigned to each program element that in return may have its own local data.
6. Semantic rules associated with productions that can recognize the beginning and ending of a subprogram are used to compute the number of currently active subprograms.
7. There are mainly two semantic rules regarding the scope of an identifier :
   a. Each identifier can only be used within its scope.
   b. Two or more identifiers with same name and are of same kind cannot be declared within the same lexical scope.
8. The scope declaration of variables, functions, labels and objects within a program is shown below :

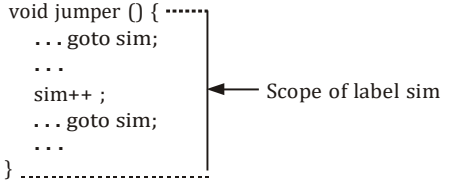
Scope of variables in statement blocks :



Scope of formal arguments of functions :



Scope of labels :



Que 4.7. **Write a short note on scoping.**

Answer

1. Scoping is method of keeping variables in different parts of program distinct from one another.
2. Scoping is generally divided into two classes :
   a. **Static scoping :** Static scoping is also called lexical scoping. In this scoping a variable always refers to its top level environment.
   b. **Dynamic scoping :** In dynamic scoping, a global identifier refers to the identifier associated with the most recent environment.

Que 4.8. Differentiate between lexical (or static) scope and dynamic scope.

Answer

S. No.	Lexical scope	Dynamic scope
1.	The binding of name occurrences to declarations is done statistically at compile time.	The binding of name occurrences to declarations is done dynamically at run-time.
2.	The structure of the program defines the binding of variables.	The binding of variables is defined by the flow of control at the run time.
3.	A free variable in a procedure gets its value from the environment in which the procedure is defined.	A free variable gets its value from where the procedure is called.

Que 4.9. Distinguish between static scope and dynamic scope. Briefly explain access to non-local names in static scope.

AKTU 2018-19, Marks 07

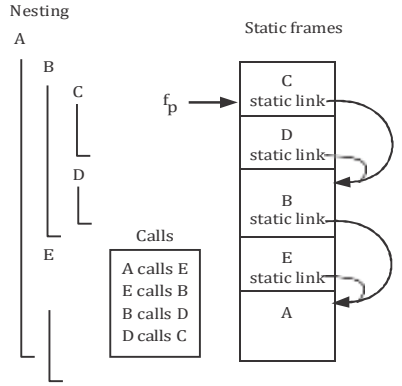
Answer

Difference : Refer Q. 4.8, Page 4–9C, Unit-4.

Access to non-local names in static scope :

1. Static chain is the mechanism to implement non-local names (variable) access in static scope.
2. A static chain is a chain of static links that connects certain activation record instances in the stack.
3. The static link, static scope pointer, in an activation record instance for subprogram *A* points to one of the activation record instances of *A*'s static parent.

4. When a subroutine at nesting level *j* has a reference to an object declared in a static parent at the surrounding scope nested at level *k*, then *j-k* static links forms a static chain that is traversed to get to the frame containing the object.
5. The compiler generates code to make these traversals over frames to reach non-local names.
   For example : Subroutine *A* is at nesting level 1 and *C* at nesting level 3. When *C* accesses an object of *A*, 2 static links are traversed to get to *A*'s frame that contains that object



PART-3

Run-Time Administration : Implementation of Simple Stack Allocation Scheme.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.10. Draw the format of activation record in stack allocation and explain each field in it. **AKTU 2018-19, Marks 07**

Answer

1. Activation record is used to manage the information needed by a single execution of a procedure.
2. An activation record is pushed into the stack when a procedure is called and it is popped when the control returns to the caller function.

Format of activation records in stack allocation :

Return value
Actual parameters
Control link
Access link
Saved machine status
Local data
Temporaries

Fields of activation record are :

1. **Return value :** It is used by calling procedure to return a value to calling procedure.
2. **Actual parameter :** It is used by calling procedures to supply parameters to the called procedures.
3. **Control link :** It points to activation record of the caller.
4. **Access link :** It is used to refer to non-local data held in other activation records.
5. **Saved machine status :** It holds the information about status of machine before the procedure is called.
6. **Local data :** It holds the data that is local to the execution of the procedure.
7. **Temporaries :** It stores the value that arises in the evaluation of an expression.

Que 4.11. How to sub-divide a run-time memory into code and data areas ? Explain. **AKTU 2016-17, Marks 10**

Answer

Sub-division of run-time memory into codes and data areas is shown in Fig. 4.11.1.

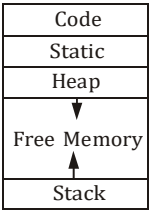


Fig. 4.11.1.

1. **Code :** It stores the executable target code which is of fixed size and do not change during compilation.
2. **Static allocation :**
   a. The static allocation is for all the data objects at compile time.
   b. The size of the data objects is known at compile time.
   c. The names of these objects are bound to storage at compile time only and such an allocation of data objects is done by static allocation.
   d. In static allocation, the compiler can determine amount of storage required by each data object. Therefore, it becomes easy for a compiler to find the address of these data in the activation record.
   e. At compile time, compiler can fill the addresses at which the target code can find the data on which it operates.
3. **Heap allocation :** There are two methods used for heap management :
   a. Garbage collection method :
      i. When all access path to a object are destroyed but data object continue to exist, such type of objects are said to be garbaged.
      ii. The garbage collection is a technique which is used to reuse that object space.
      iii. In garbage collection, all the elements whose garbage collection bit is 'on' are garbaged and returned to the free space list.
   b. Reference counter :
      i. Reference counter attempt to reclaim each element of heap storage immediately after it can no longer be accessed.
      ii. Each memory cell on the heap has a reference counter associated with it that contains a count of number of values that point to it.
      iii. The count is incremented each time a new value point to the cell and decremented each time a value ceases to point to it.
4. **Stack allocation :**
   a. Stack allocation is used to store data structure called activation record.
   b. The activation records are pushed and popped as activations begins and ends respectively.

c. Storage for the locals in each call of the procedure is contained in the activation record for that call. Thus, locals are bound to fresh storage in each activation, because a new activation record is pushed onto the stack when call is made.

d. These values of locals are deleted when the activation ends.

Que 4.12. Why run-time storage management is required ? How simple stack implementation is implemented ?

Answer

Run-time storage management is required because :

1. A program needs memory resources to execute instructions.
2. The storage management must connect to the data objects of programs.
3. It takes care of memory allocation and deallocation while the program is being executed.

Simple stack implementation is implemented as :

1. In stack allocation strategy, the storage is organized as stack. This stack is also called control stack.
2. As activation begins the activation records are pushed onto the stack and on completion of this activation the corresponding activation records can be popped.
3. The locals are stored in the each activation record. Hence, locals are bound to corresponding activation record on each fresh activation.
4. The data structures can be created dynamically for stack allocation.

Que 4.13. Discuss the following parameter passing techniques with suitable example.

i. Call by name

ii. Call by reference

OR

Explain the various parameter passing mechanisms of a high level language. **AKTU 2018-19, Marks 07**

Answer

i. Call by name :

1. In call by name, the actual parameters are substituted for formals in all the places where formals occur in the procedure.

2. It is also referred as lazy evaluation because evaluation is done on parameters only when needed.

For example :

```
main (){
int n1=10;n2=20;
printf("n1: %d, n2: %d\n", n1, n2);
swap(n1,n2);
printf("n1: %d, n2: %d\n", n1, n2); }
swap(int c ,int d){
int t;
t=c;
c=d;
d=t;
printf("n1: %d, n2: %d\n", n1, n2);
}
```

Output : 10 20
20 10
20 10

ii. Call by reference :

1. In call by reference, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within the called function.
2. In call by reference, alteration to actual arguments is possible within called function; therefore the code must handle arguments carefully else we get unexpected results.

For example :

```
#include <stdio.h>
void swapByReference(int*, int*); /* Prototype */
int main() /* Main function */
{
int n1 = 10; n2 = 20;
```

```
/* actual arguments will be altered */
swapByReference(&n1, &n2);
printf("n1: %d, n2: %d\n", n1, n2);
}
void swapByReference(int *a, int *b)
{
int t;
t = *a; *a = *b; *b = t;
}
```

Output : n1: 20, n2: 10

PART-4

Storage Allocation in Block Structured Language.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.14. Explain symbol table organization using hash tables. With an example show the symbol table organization for block structured language.

Answer

1. Hashing is an important technique used to search the records of symbol table. This method is superior to list organization.
2. In hashing scheme, a hash table and symbol table are maintained.
3. The hash table consists of k entries from 0, 1 to $k - 1$. These entries are basically pointers to symbol table pointing to the names of symbol table.
4. To determine whether the 'Name' is in symbol table, we used a hash function 'h' such that h(name) will result any integer between 0 to $k - 1$. We can search any name by position = h(name).
5. Using this position, we can obtain the exact locations of name in symbol table.

6. The hash table and symbol table are shown in Fig. 4.14.1.

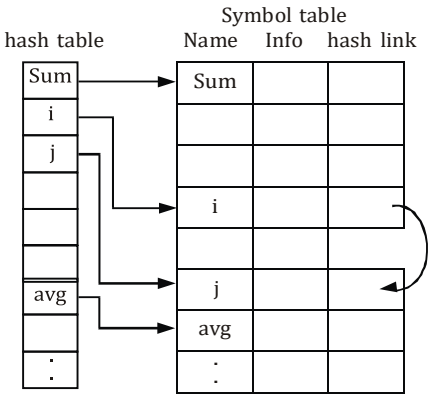


Fig. 4.14.1.

7. The hash function should result in uniform distribution of names in symbol table.
8. The hash function should have minimum number of collision.

PART-5

Error Detection and Recovery : Lexical Phase Errors, Syntactic Phase Errors, Semantic Errors.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.15. Define error recovery. What are the properties of error message ? Discuss the goals of error handling.

Answer

Error recovery : Error recovery is an important feature of any compiler, through which compiler can read and execute the complete program even it have some errors.

Properties of error message are as follows :

1. Message should report the errors in original source program rather than in terms of some internal representation of source program.
2. Error message should not be complicated.
3. Error message should be very specific and should fix the errors at correct positions.
4. There should be no duplicacy of error messages, i.e., same error should not be reported again and again.

Goals of error handling are as follows :

1. Detect the presence of errors and produce "meaningful" diagnostics.
2. To recover quickly enough to be able to detect subsequent errors.
3. Error handling components should not significantly slow down the compilation of syntactically correct programs.

Que 4.16. What are lexical phase errors, syntactic phase errors and semantic phase errors ? Explain with suitable example.

AKTU 2015-16, Marks 10

Answer

1. Lexical phase error :

a. A lexical phase error is a sequence of character that does not match the pattern of token *i.e.*, while scanning the source program, the compiler may not generate a valid token from the source program.

b. Reasons due to which errors are found in lexical phase are :

i. The addition of an extraneous character.

ii. The removal of character that should be presented.

iii. The replacement of a character with an incorrect character.

iv. The transposition of two characters.

For example :

i. In Fortran, an identifier with more than 7 characters long is a lexical error.

ii. In Pascal program, the character ~, & and @ if occurred is a lexical error.

2. Syntactic phase errors (syntax error) :

a. Syntactic errors are those errors which occur due to the mistake done by the programmer during coding process.

b. Reasons due to which errors are found in syntactic phase are :

i. Missing of semicolon

ii. Unbalanced parenthesis and punctuation

For example : Let us consider the following piece of code :

```
int x;

int y //Syntax error
```

In example, syntactic error occurred because of absence of semicolon.

3. Semantic phase errors :

a. Semantic phase errors are those errors which occur in declaration and scope in a program.

b. Reason due to which errors are found :

i. Undeclared names

ii. Type incompatibilities

iii. Mismatching of actual arguments with the formal arguments.

For example : Let us consider the following piece of code :

```
scanf("%f%f", a, b);
```

In example, *a* and *b* are semantic error because scanf uses address of the variables as &a and &b.

4. Logical errors :

a. Logical errors are the logical mistakes founded in the program which is not handled by the compiler.

b. In these types of errors, program is syntactically correct but does not operate as desired.

For example :

Let consider following piece of code :

```
x = 4;

y = 5;

average = x + y/2;
```

The given code do not give the average of *x* and *y* because BODMAS property is not used properly.

Que 4.17. What do you understand by lexical error and syntactic error ? Also, suggest methods for recovery of errors.

OR

Explain logical phase error and syntactic phase error. Also suggest methods for recovery of error. **AKTU 2017-18, Marks 10**

Answer

Lexical and syntactic error : Refer Q. 4.16, Page 4–17C, Unit-4.

Various error recovery methods are :

1. **Panic mode recovery :**
   a. This is the simplest method to implement and used by most of the parsing methods.
   b. When parser detect an error, the parser discards the input symbols one at a time until one of the designated set of synchronizing token is found.
   c. Panic mode correction often skips a considerable amount of input without checking it for additional errors. It gives guarantee not to go in infinite loop.

   For example :

   Let consider a piece of code :

   $a = b + c;$

   $d = e + f;$

   By using panic mode it skips $a = b + c$ without checking the error in the code.

2. **Phrase-level recovery :**
   a. When parser detects an error the parser may perform local correction on remaining input.
   b. It may replace a prefix of the remaining input by some string that allows parser to continue.
   c. A typical local correction would replace a comma by a semicolon, delete an extraneous semicolon or insert a missing semicolon.

   For example :

   Let consider a piece of code

   while $(x > 0)$ $y = a + b;$

   In this code local correction is done by phrase-level recovery by adding 'do' and parsing is continued.

3. **Error production :** If error production is used by the parser, we can generate appropriate error message and parsing is continued.

   For example :

   Let consider a grammar

   $E \rightarrow + E | - E | * A | / A$

   $A \rightarrow E$

   When error production encounters * A, it sends an error message to the user asking to use '*' as unary or not.

4. **Global correction :**
   a. Global correction is a theoretical concept.
   b. This method increases time and space requirement during parsing.

Que 4.18. **Explain in detail the error recovery process in operator precedence parsing method.** **AKTU 2018-19, Marks 07**

Answer

Error recovery in operator precedence parsing :

1. There are two points in the parsing process at which an operator-precedence parser can discover syntactic error :
   a. If no precedence relation holds between the terminal on top of the stack and the current input.
   b. If a handle has been found, but there is no production with this handle as a right side.
2. The error checker does the following errors :
   a. Missing operand
   b. Missing operator
   c. No expression between parentheses
   d. These error diagnostic issued at handling of errors during reduction.
3. During handling of shift/reduce errors, the diagnostic's issues are :

a. Missing operand

b. Unbalanced right parenthesis

c. Missing right parenthesis

d. Missing operators

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What are the symbol table requirements ? What are the demerits in the uniform structure of symbol table ?

Ans. Refer Q. 4.2.

Q. 2. What is the role of symbol table ? Discuss different data structures used for symbol table.

Ans. Refer Q. 4.4.

Q. 3. Describe symbol table and its entries. Also, discuss various data structure used for symbol table.

Ans. Refer Q. 4.5.

Q. 4. Distinguish between static scope and dynamic scope. Briefly explain access to non-local names in static scope.

Ans. Refer Q. 4.9.

Q. 5. Draw the format of activation record in stack allocation and explain each field in it.

Ans. Refer Q. 4.10.

Q. 6. Explain the various parameter passing mechanisms of a high level language.

Ans. Refer Q. 4.13.

Q. 7. Define error recovery. What are the properties of error message ? Discuss the goals of error handling.

Ans. Refer Q. 4.15.

Q. 8. What are lexical phase errors, syntactic phase errors and semantic phase errors ? Explain with suitable example.

Ans. Refer Q. 4.16.

Q. 9. Explain logical phase error and syntactic phase error. Alsosuggest methods for recovery of error.

Ans. Refer Q. 4.17.

Q. 10. Explain in detail the error recovery process in operatorprecedence parsing method.

Ans. Refer Q. 4.18.