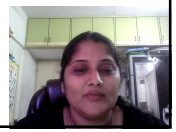
Unit- V Graph and Symbol Table

Deepali Londhe



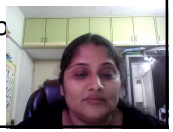
1

Unit-V Contents

Graph -Concept and terminologies, Graph as an ADT, Representation of graphs using adjacency matrix and adjacency list, Breadth First Search traversal, Depth First Search traversal, Prim's and Kruskal's algorithms for minimum spanning tree, Shortest path using Dijkstra's algorithm, topological sorting.

Symbol Table -Notion of Symbol Table, OBST, AVL Trees

Heap: Heap data structure, Min and Max Heap, Heap so applications of heap

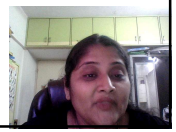


2

Content

- **Unit-V**
- Symbol Table
  - Notion of Symbol Table
  - OBST
  - HEAP
- Heap Data Structure
  - Concept
  - Applications
  - Main Heap
  - Max Heap
  - Heap sort Implementation

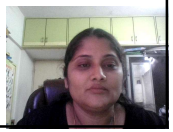
- **Unit VI**
- Hash Tables:
  - Basic concepts,
  - Hash function,
  - Hashing methods
  - Collision resolution,



3

Symbol table

- While compiler and assemblers scan a program, each identifier must be examined to determine if it is a keyword.
- This information is stored in a special kind of data structure known as symbol table.
- When identifiers are found, they will be entered into a **symbol table,** which will hold all relevant information about identifiers
- Keywords are also stored in a symbol table for the look up purpose



4

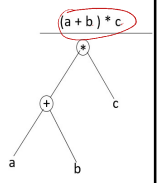
The Symbol Table

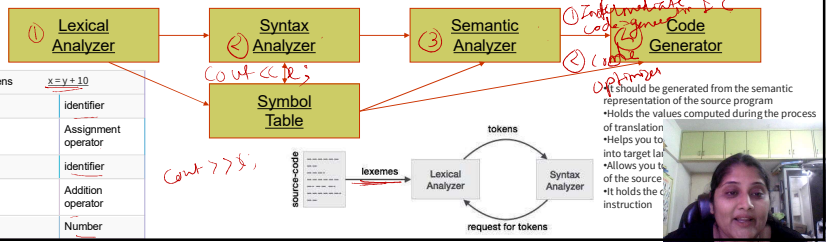
- This information will be used later by the semantic analyzer and the code generator.

•Identify the lexical units in a source code
•Classify lexical units into classes like constants, reserved words, and enter them in different tables. It will Ignore comments in the source program
•Identify token which is not a part of the language

•Obtain tokens from the lexical analyzer
•Checks if the expression is syntactically correct or not
•Report all syntax errors
•Construct a hierarchical structure which is known as a parse tree

•Helps you to store type information gathered and save it in symbol table or syntax tree
•Allows you to perform type checking
•In the case of type mismatch, where there are no exact type correction rules which satisfy the desired operation a semantic error is shown
•Collects type information and checks for type compatibility
•Checks if the source language permits the operands or not





5

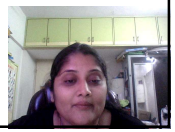
Need For symbol Table In lexical analysis

Token categorization :

- **What is a token ?**

  It is a keywords, identifiers, labels, constants etc in the program.

- Source program is scanned character by character the to identify token .
- After identification these tokens are stored in a special structure is called as **symbol table.**



6

Structure of symbol table

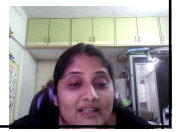
- Symbol Table is a standard Data Structure used in language processing.
- Symbol table store information pertaining to the tokens as

A **<name/symbol/ token ,atrribute >** pair.

- ***Thus Symbol Table is a set of name-value pair.***

Symbol	Attributes
A	----------
Total	----------
B	----------

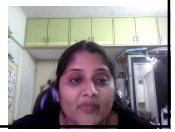
- The attribute can be Types of identifier, Its usage or its memory address.
- There are separate symbol tables for keywords, user identifiers, constants etc.



7

The structure of symbol table

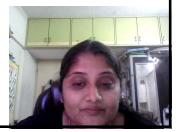
- **We will store the following information about identifiers**.
  - The name (as a string).
  - The data type.
  - The block level.
  - Its scope (global, local, or parameter).
  - Its offset from the base pointer (for local variables and parameters only).



8

Symbol table

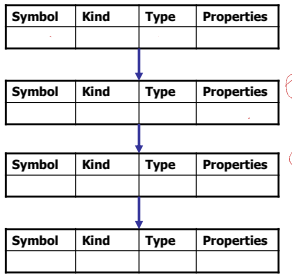
- An environment that stores information about identifiers
- A data structure that captures scope information
- Each entry in symbol table contains
  - The name of an identifier
  - Its kind (variable/method/field…)
  - Type
  - Additional properties, e.g, `final`, `public` (not needed for IC)
- One symbol table for each scope

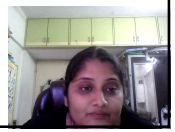


9

Scope nesting in IC

Scope nesting mirrored in hierarchy of symbol tables



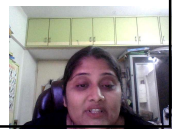


10

Symbol table example

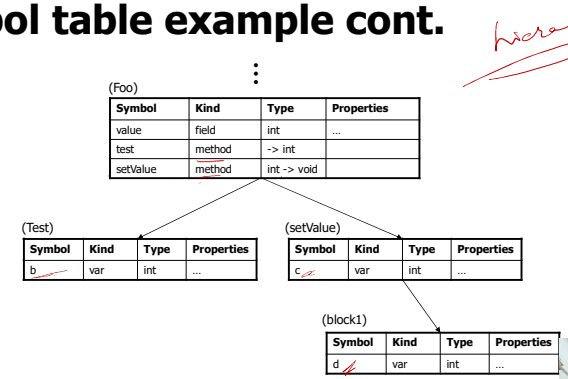
```
class Foo {
  int value;
  int test() {
    int b = 3;
    return value + b;
  }
  void setValue(int c) {
    value = c;
    { int d = c;
      c = c + d;
      value = c;
    }
  }
}

class Bar {
  int value;
  void setValue(int c) {
       value = c;
  }
}
```

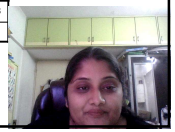
scope of b
scope of d (block1)
scope of c
scope of value
scope of c
scope of value



11

Symbol table example cont.





12

Checking scope rules

(Foo)

Symbol	Kind	Type	Properties
value	field	int	...
test	method	-> int	
setValue	method	int -> void	

(Test)

Symbol	Kind	Type	Properties
b	var	int	...

(setValue)

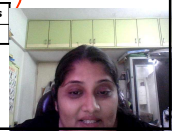
Symbol	Kind	Type	Properties
c	var	int	...

(block1)

Symbol	Kind	Type	Properties
d	var	int	...

lookup(value)

```
void setValue(int c) {
    value = c;
    { int d = c;
      c = c + d;
      value = c;
    }
}
```



13

Catching semantic errors

Error !

(Foo)

Symbol	Kind	Type	Properties
value	field	int	...
test	method	-> int	
setValue	method	int -> void	

(Test)

Symbol	Kind	Type	Properties
b	var	int	...

(setValue)

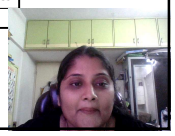
Symbol	Kind	Type	Properties
c	var	int	...

(block1)

Symbol	Kind	Type	Properties
d	var	int	...

lookup(myValue)

```
void setValue(int c) {
    value = c;
    { int d = c;
      c = c + d;
      myValue = c;
    }
}
```



14

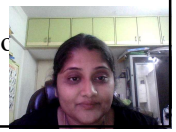
Operations on Symbol Table

Operations on Symbol Table

- Search for a particular name.
- Retrieve the attributes of a name.
- Insert a new name and its value.
- Delete a name and its value

Note:

For ADT of symbol Table refer Fundamental
Data structure by Sartaj sahani



15

Types of symbol table

Static symbol table

"constructed for fixed set of data already known in advance

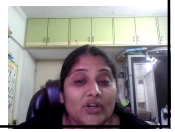
and no insert and delete after Construction , only searching is mainly needed operation "

Eg. Keywords table

Dynamic symbol table

" Insertion, deletion and searching can be done any time , dynamic in input set."

Eg. User identifier table



16

Ordered list

Possible implementation

Unordered one list : Array / Linked list

Easy to implement

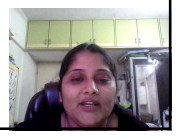
Search time will be very long if source has many symbols

Ordered list :

- Hash table

Tree like table :

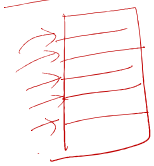
- **Binary Search tree its variations:**
  - Red black tree
  - AVL tree
  - OBST

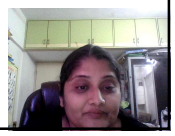


17

Static Tree Table :

- BST /OBST
- Huffman's tree
- Hash Table (keyword table).

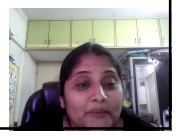




18

Dynamic Tree Table

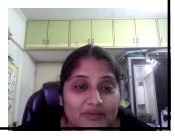
- BST/AVL Tree(Adelson Velskii and E.M. Landis)
- Hash Tables



19

Static Tree Table

- Optimal Binary Search Tree (OBST),Huffman Tree and Hashing can be used to implement Static Tree table When different symbols and its probability are known in advance.
- OBST comes under Dynamic programming Algorithm Technique.



20

Thank You !!!