JECRC Foundation

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Year & Sem – 3rd Year & 5th Sem
Subject – COMPILER DESIGN
Unit – 4
Presented by – (Abhishek Dixit, Assistant Prof., Dept of CSE)

VISION AND MISSION OF INSTITUTE

To become a renowned center of outcome based learning and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.

M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.

M3: Offer opportunities for interaction between academia and industry.

M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

VISION OF THE DEPARTMENT

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

MISION OF THE DEPARTMENT

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.
M2: To provide opportunities for interaction between academia and industry.
M3: To provide platform for lifelong learning by accepting the change in technologies
M4: To develop aptitude of fulfilling social responsibilities

PROGRAM OUTCOMES

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM EDUCATIONAL OBJECTIVES

1. To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science &Engineering by way of analyzing and exploiting engineering challenges.
2. To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
3. To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.
4. To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self motivated life-long learning needed for a successful professional career.
5. To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge

PROGRAM SPECIFIC OBJECTIVES

1. PSO1. Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.

2. PSO2. Ability to Design and Develop Mobile and Web-based applications under realistic constraints.

COURSE OUTCOME

CO1: Compare different phases of compiler and design lexical analyzer.
CO2: Examine syntax and semantic analyzer by understanding grammars.
CO3: Illustrate storage allocation and its organization & analyze symbol table organization.
CO4: Analyze code optimization, code generation & compare various compilers.

CO-PO MAPPING

Semester	Subject	Code	L/T/P	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
V	COMPILER DESIGN	5CS4-02	L	1. Compare different phases of compiler and design lexical analyzer..	3	3	3	3	1	1	1	1	1	2	1	3
			L	2. Examine syntax and semantic analyzer by understanding grammars.	3	3	3	2	1	1	1	0	1	2	1	3
			L	3. Illustrate storage allocation and its organization & analyze symbol table organization.	3	3	2	2	1	1	1	1	1	2	1	3
			L	4.Analyze code optimization, code generation & compare various compilers.	3	3	3	3	2	1	1	1	1	2	1	3

SYLLABUS

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

III Year-V Semester: B.Tech. Computer Science and Engineering

5CS4-02: Compiler Design

Credit: 3 **Max. Marks: 150(IA:30, ETE:120)**

3L+0T+0P **End Term Exam: 3 Hours**

SN	Contents	Hours
1	**Introduction:**Objective, scope and outcome of the course.	01
2	**Introduction:** Objective, scope and outcome of the course. Compiler, Translator, Interpreter definition, Phase of compiler, Bootstrapping, Review of Finite automata lexical analyzer, Input, Recognition of tokens, Idea about LEX: A lexical analyzer generator, Error handling.	06
3	**Review of CFG Ambiguity of grammars:** Introduction to parsing. Top down parsing, LL grammars & passers error handling of LL parser, Recursive descent parsing predictive parsers, Bottom up parsing, Shift reduce parsing, LR parsers, Construction of SLR, Conical LR & LALR parsing tables, parsing with ambiguous grammar. Operator precedence parsing, Introduction of automatic parser generator: YACC error handling in LR parsers.	10
4	**Syntax directed definitions;** Construction of syntax trees, S-Attributed Definition, L-attributed definitions, Top down translation. Intermediate code forms using postfix notation, DAG, Three address code, TAC for various control structures, Representing TAC using triples and quadruples, Boolean expression and control structures.	10
5	**Storage organization;** Storage allocation, Strategies, Activation records, Accessing local and non-local names in a block structured language, Parameters passing, Symbol table organization, Data structures used in symbol tables.	08
6	**Definition of basic block control flow graphs;** DAG representation of basic block, Advantages of DAG, Sources of optimization, Loop optimization, Idea about global data flow analysis, Loop invariant computation, Peephole optimization, Issues in design of code generator, A simple code generator, Code generation from DAG.	07

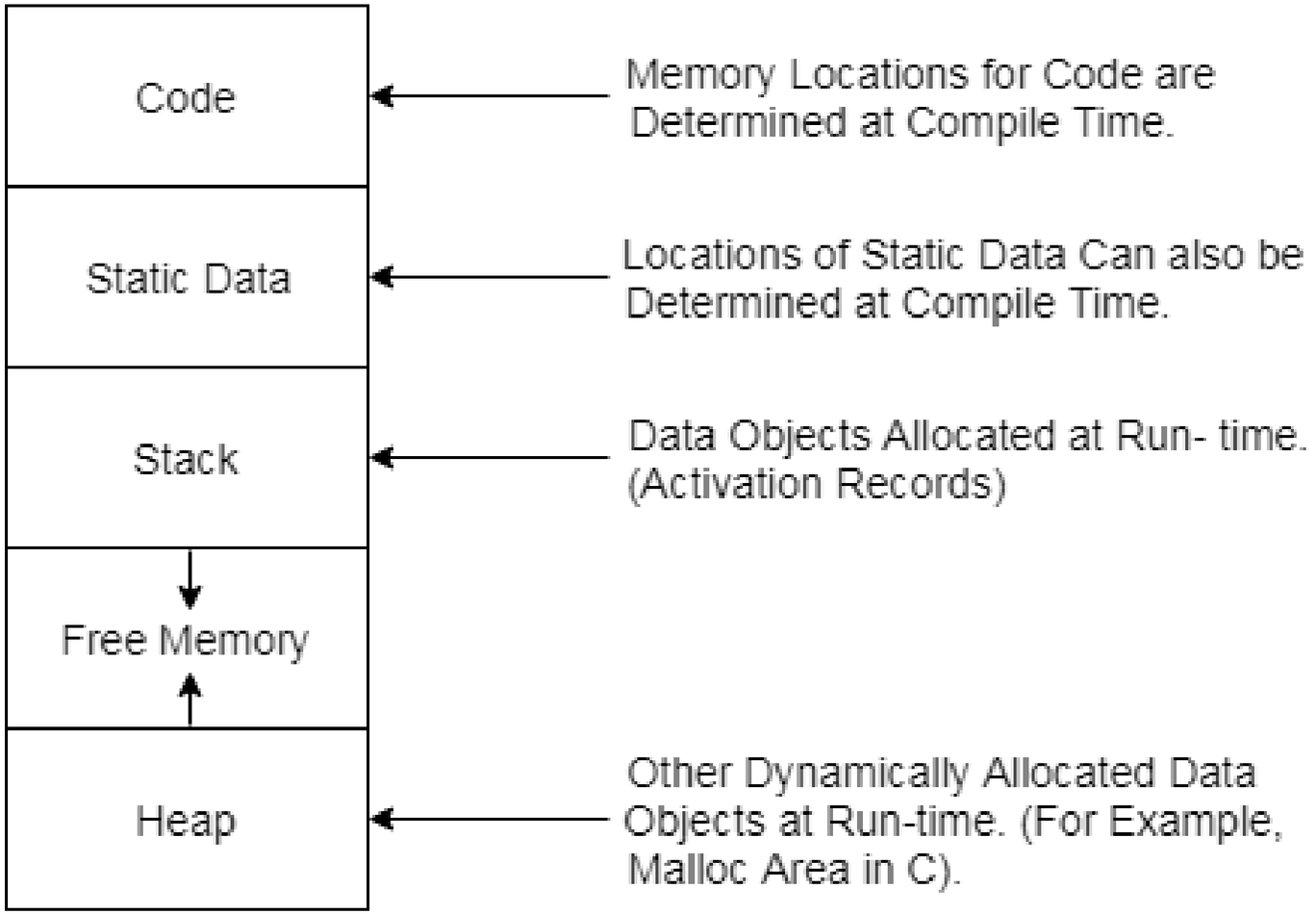
Storage Organization:

When the target program executes then it runs in its own logical address space in which the value of each program has a location.

The logical address space is shared among the compiler, operating system and target machine for management and organization.

The operating system is used to map the logical address into physical address which is usually spread throughout the memory.

Subdivision of Run-time Memory:



STORAGE ALLOCATION

- Storage allocation strategies are the strategies by which it is decided that which type of object is provided to particular data object.
- It is based on programming language implementation.

STORAGE ALLOCATION STRATEGIES

The different storage allocation strategies are :

1. Static allocation - lays out storage for all data objects at compile time
2. Stack allocation - manages the run-time storage as a stack.
3. Heap allocation - allocates and deallocates storage as needed at run time from a data area known as heap.

STATIC ALLOCATION

In static allocation, names are bound to storage as the program is compiled, so there is no need for a run-time support package. Since the bindings do not change at run-time, every time a procedure is activated, its names are bound to the same storage locations. Therefore values of local names are retained across activations of a procedure.

That is, when control returns to a procedure the values of the locals are the same as they were when control left the last time. From the type of a name, the compiler decides the amount of storage for the name and decides where the activation records go. At compile time, we can fill in the addresses at which the target code can find the data it operates on.

STACK ALLOCATION OF SPACE

All compilers for languages that use procedures, functions or methods as units of user-defined actions manage at least part of their run-time memory as a stack. Each time a procedure is called , space for its local variables is pushed onto a stack, and when the procedure terminates, that space is popped off the stack.

Calling sequences:

Procedures called are implemented in what is called as calling sequence, which consists of code that allocates an activation record on the stack and enters information into its fields. A return sequence is similar to code to restore the state of machine so the calling procedure can continue its execution after the call. The code in calling sequence is often divided between the calling procedure (caller) and the procedure it calls (callee).

HEAP ALLOCATION

Stack allocation strategy cannot be used if either of the following is possible :

1. The values of local names must be retained when an activation ends.
2. A called activation outlives the caller.

Heap allocation parcels out pieces of contiguous storage, as needed for activation records or other objects. Pieces may be deallocated in any order, so over the time the heap will consist of alternate areas that are free and in use.

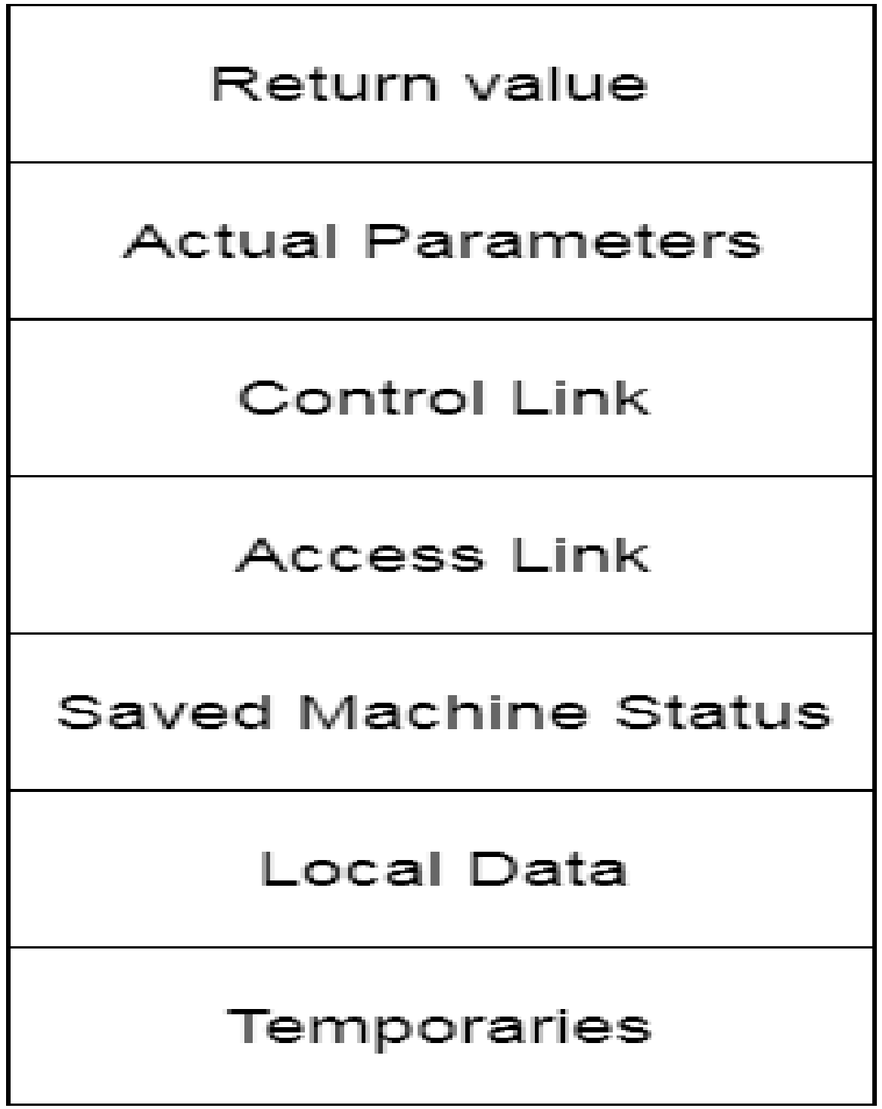
Activation Record:

Control stack is a run time stack which is used to keep track of the live procedure activations i.e. it is used to find out the procedures whose execution have not been completed.

When it is called (activation begins) then the procedure name will push on to the stack and when it returns (activation ends) then it will popped.

Activation record is used to manage the information needed by a single execution of a procedure.

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.


Return value
Actual Parameters
Control Link
Access Link
Saved Machine Status
Local Data
Temporaries

Return Value: It is used by calling procedure to return a value to calling procedure.
Actual Parameter: It is used by calling procedures to supply parameters to the called procedures.
Control Link: It points to activation record of the caller.
Access Link: It is used to refer to non-local data held in other activation records.
Saved Machine Status: It holds the information about status of machine before the procedure is called.
Local Data: It holds the data that is local to the execution of the procedure.
Temporaries: It stores the value that arises in the evaluation of an expression.

Accessing local and non-local names in a block structured Language:

What is Block?

- A block is a section of software code or an algorithm in software programming.
- A block can consist of one or more statements or declarations. It is possible for a block to contain one or more blocks nested within it.
- If a programming language comprises blocks and nested blocks, it is called a block-structured programming language.
- Blocks are a basic feature of structured programming and help form control structures.
- However, it is not necessary to add blocks in software code; blocks should be driven by necessity.
- At the same time, blocks can improve code efficiency.
- A block is also known as a code block.

What is LOCAL & NON-LOCAL Variable

• A local variable is a variable that is given *local scope*. Local variable references in the function or block in which it is declared override the same variable name in the larger scope. In programming languages with only two levels of visibility, local variables are contrasted with global variables.

• In programming language theory, a non-local variable is a variable that is not defined in the local scope. While the term can refer to global variables, it is primarily used in the context of nested and anonymous functions where some variables can be neither in the local nor the global scope.

There are two type of scope rules

1. **Static Scope Rule:**

   Scope of name can be determined by examining the text of the program

2. **Dynamic Scope Rule:**

   The scope of the name can be determined by considering the current activation of the procedure

There are two type of scope rules

1. **Static Scope Rule:**

   Scope of name can be determined by examining the text of the program

2. **Dynamic Scope Rule:**

   The scope of the name can be determined by considering the current activation of the procedure

Parameters passing:

The communication medium among procedures is known as parameter passing. The values of the variables from a calling procedure are transferred to the called procedure by some mechanism.

Before moving ahead, first go through some basic terminologies pertaining to the values in a program.

r-value:
The value of an expression is called its r-value. The value contained in a single variable also becomes an r-value if it appears on the right-hand side of the assignment operator. r-values can always be assigned to some other variable.

l-value:
The location of memory (address) where an expression is stored is known as the l-value of that expression. It always appears at the left hand side of an assignment operator.

For example:

```
day = 1;
week = day * 7;
 month = 1;
year = month * 12;
```

From this example, we understand that constant values like 1, 7, 12, and variables like day, week, month and year, all have r-values. Only variables have l-values as they also represent the memory location assigned to them.

For example:

```
7 = x + y;
```

is an l-value error, as the constant 7 does not represent any memory location.

Pass by Value:

In pass by value mechanism, the calling procedure passes the r-value of actual parameters and the compiler puts that into the called procedure's activation record. Formal parameters then hold the values passed by the calling procedure. If the values held by the formal parameters are changed, it should have no impact on the actual parameters.

Pass by Reference:

In pass by reference mechanism, the l-value of the actual parameter is copied to the activation record of the called procedure. This way, the called procedure now has the address (memory location) of the actual parameter and the formal parameter refers to the same memory location. Therefore, if the value pointed by the formal parameter is changed, the impact should be seen on the actual parameter as they should also point to the same value

Parameter passing methods are the ways in which parameters are transferred between functions when one function calls another.

There are four different ways of passing parameters which are as follows:

- 1. Call by Value
- 2. Call by Reference (address)
- 3. Call by Value Result
- 4. Call by Name

❖ Passing parameter by value

- By default, parameters are passed by value. In this method a duplicate copy is made and sent to the called function. There are two copies of the variables. So if you change the value in the called method it won't be changed in the calling method.

Example:

```
void swap(int x,int y){
int t;
t=x;
x=y;
y=t;  }
void main(){
 int a=1,b=2;
 swap(a,b);
 printf("a=%d,b=%d",a,b); }
```

❖ Passing parameter by reference

- Passing parameters by ref uses the address of the actual parameters to the formal parameters. It requires ref keyword in front of variables to identify in both actual and formal parameters.

- We use this process when we want to use or change the values of the parameters passed.

Example:

```
void swap(int *x, int*y){
  int t;
  t=*x;
  *x=*y;
  *y=t;
 }
void main()
 {
  int a=1,b=2;
 swap(&a , &b);
 printf("a=%d , b=%d" ,a ,b)
 }
```

Passing parameter by Value Result

- Like reference parameters, output parameters don't create a new storage location and are passed by reference. During the execution of called method the actual parameter value is not affected. After execution value of formal parameter is passed to actual parameter.

- We use this process when we want some parameters to bring back some processed values form the called method.

Example:

```
    int y: //global variable
copyValue(int x){
      x=2;
      y=0;  }
  main(){
   y=10;
   copyValue(y);
   print(y);  }
```

Passing parameter by Name

- In this technique of parameter passing the actual parameter are substituted for the formal parameter i.e formal parameter are replaced by actual parameter inside function.

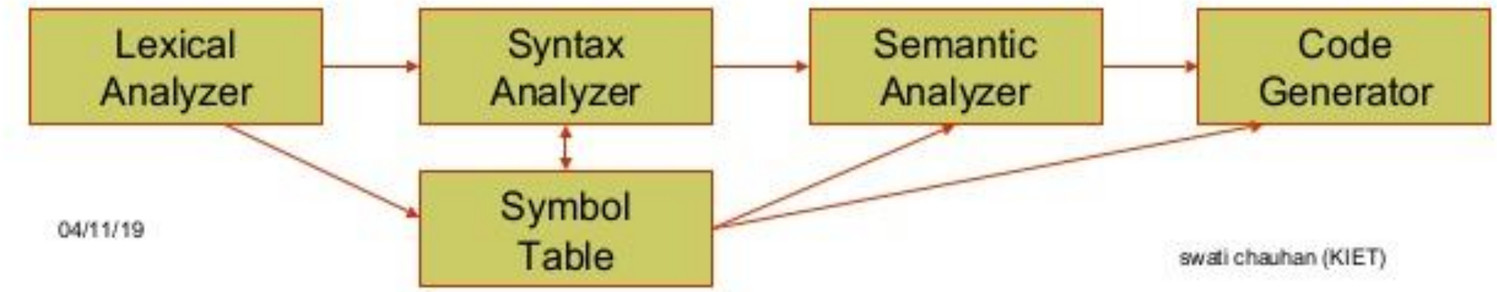
Example:

```
void Init(int x,int y){

 for(int k=0; k<=5; k++){
 y=0;    // here A[j] is replaced with y
 x=x++; // here j is replaced with x
 }
}

main(){
    int i,A[10];
   j=0;
  Init(j,A[j]);
  }
```

SYMBOL TABLE

Symbol tables are data structures that are used by compilers to hold information about source-program constructs

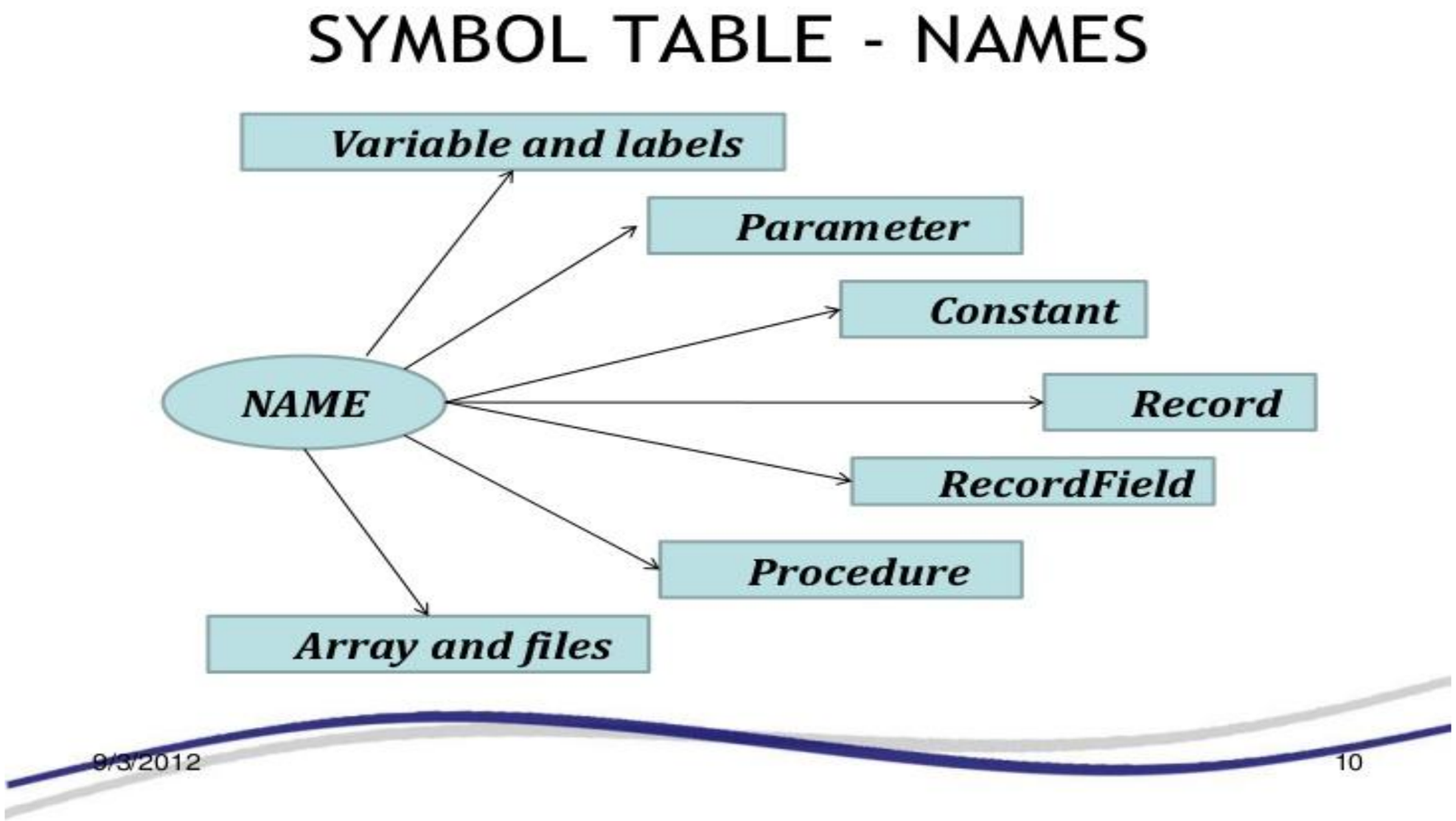
.



APPLICATIONS

- It is used to store the name of all entities in a structured form at one place.
- It is used to verify if a variable has been declared.
- It is used to determine the scope of a name.
- It is used to implement type checking by verifying assignments and expressions in the source code are semantically correct.

COMPONENTS OF SYMBOL TABLE



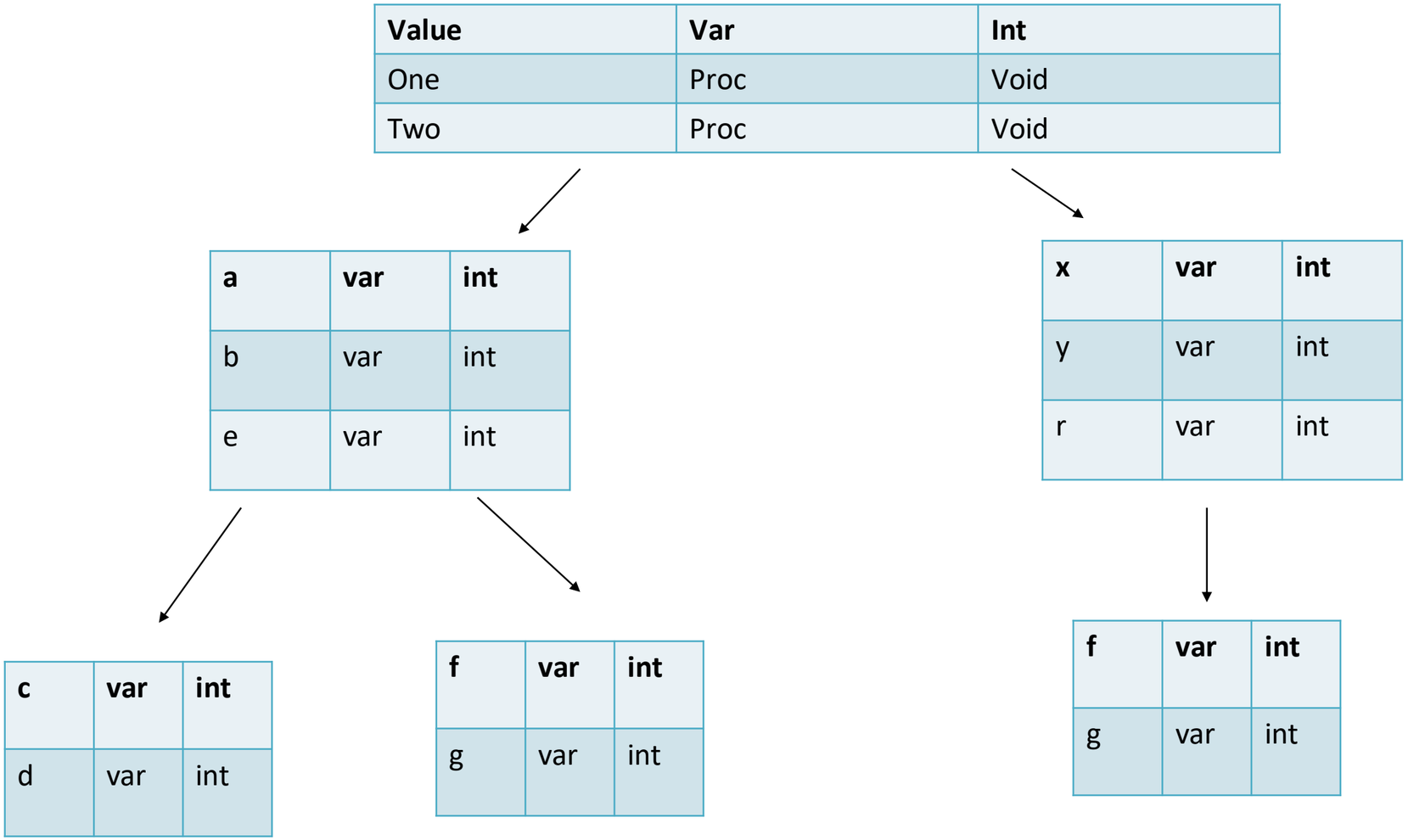
OPERATIONS ON SYMBOL TABLE

- Insert
- Lookup
- Delete
- Scope Management
    1. Local
    2. Global

Example

```
int value;
void one( )
{ int a;
   int b;
    { int c;
       int d;
     }
int e;
{ int f;
   int g;
}
```

```
void two( )
{ int x;
   int y;
    { int p;
       int q;
     }
int r;
}
```


Value
Var
Int
One
Proc
Void
Two
Proc
Void
a
var
int
b
var
int
e
var
int
x
var
int
y
var
int
r
var
int
c
var
int
d
var
int
f
var
int
g
var
int
f
var
int
g
var
int

Implementation

A symbol table can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table

Among all, symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol

Linear Search

- Simplest way
- Array is used to store information
- If there are n records then
    1. n/2- successful search
    2. n- unsuccessful search

Binary Search

- Store symbol in alphabetical order
- Table is divided into 2 tables
- Achieved by comparing coming elements with middle element

Hash Table

- Most powerful implementation table
- 2 tables are maintained
    1. Hash table
    2. Symbol Table
- Tables are link through pointer.

REFERENCES/BIBLOGRAPHY

1. slideshare.net
2. Javapoint.com
3. Cse.iitm.ac.in

JECRC Foundation

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Thank you!