



SUBJECT- COMPILER DESIGN



SEMESTER- 5TH SEM

VISSION 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.

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 symboltable 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	2	1	1	1	1	2	1	3		
			L	2. Examine syntax and semantic analyzer and illustrate storage allocation and its organization	3	3	3	3	1	1	1	1	1	1	2	2	3	
			L	3. Analyze symbol table organization, code optimization and code generator	3	3	3	3	2	1	1	1	1	1	1	2	2	3
			L	4. Compare and evaluate various compilers and analyzers	3	3	3	3	2	1	1	1	1	1	1	2	1	3

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

PSO

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

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

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

LECTURE PLAN:**Subject: Compiler Design (5CS4 – 02)****Year/Sem: III/V**

Unit No./ Total lec. Req.	Topics	Lect. Req.
Unit-1 (6)	Compiler, Translator, Interpreter definition, Phase of compiler	1
	Introduction to one pass & Multipass compilers, Bootstrapping	1
	Review of Finite automata lexical analyzer, Input, buffering,	2
	Recognition of tokens, Idea about LEX:, GATE Questions	1
	A lexical analyzer generator, Error Handling, Unit Test	1
Unit-2 (17)	Review of CFG Ambiguity of grammars, Introduction to parsing	2
	Bottom up parsing Top down Parsing Technique	5
	Shift reduce parsing, Operator Precedence Parsing	2
	Recursive descent parsing predictive parsers	1
	LL grammars & passers error handling of LL parser	1
	Conical LR & LALR parsing tables	3
	parsing with ambiguous grammar, GATE Questions	2
Introduction of automatic parser generator: YACC error handling in LR parsers, Unit Test	1	
Unit 3- (7)	Syntax directed definitions; Construction of syntax trees	1
	L-attributed definitions, Top down translation	1
	Specification of a type checker, GATE Questions	1
	Intermediate code forms using postfix notation and three address code,	2
	Representing TAC using triples and quadruples, Translation of assignment statement.	1
	Boolean expression and control structures, Unit Test	1
Unit 4- (4)	Storage organization, Storage allocation, Strategies, Activation records,	1
	Accessing local and non local names in a block structured language	1
	Parameters passing, Symbol table organization, GATE Questions	1
	Data structures used in symbol tables, Unit Test	1
Unit 5- (6)	Definition of basic block control flow graphs,	1
	DAG representation of basic block, Advantages of DAG,	1
	Sources of optimization, Loop optimization Idea about global data flow analysis, Loop invariant computation, Loop invariant computation, Tutorial	2
	Peephole optimization, GATE Questions, Tutorial	1
	Issues in design of code generator, A simple code generator, Code generation from DAG., UNIT TEST, Revision	1



JECRC Foundation



**JAIPUR ENGINEERING COLLEGE
AND RESEARCH CENTRE**

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Year & Sem – 3rd/ 5th sem
Subject – Compiler Design
Unit – 4

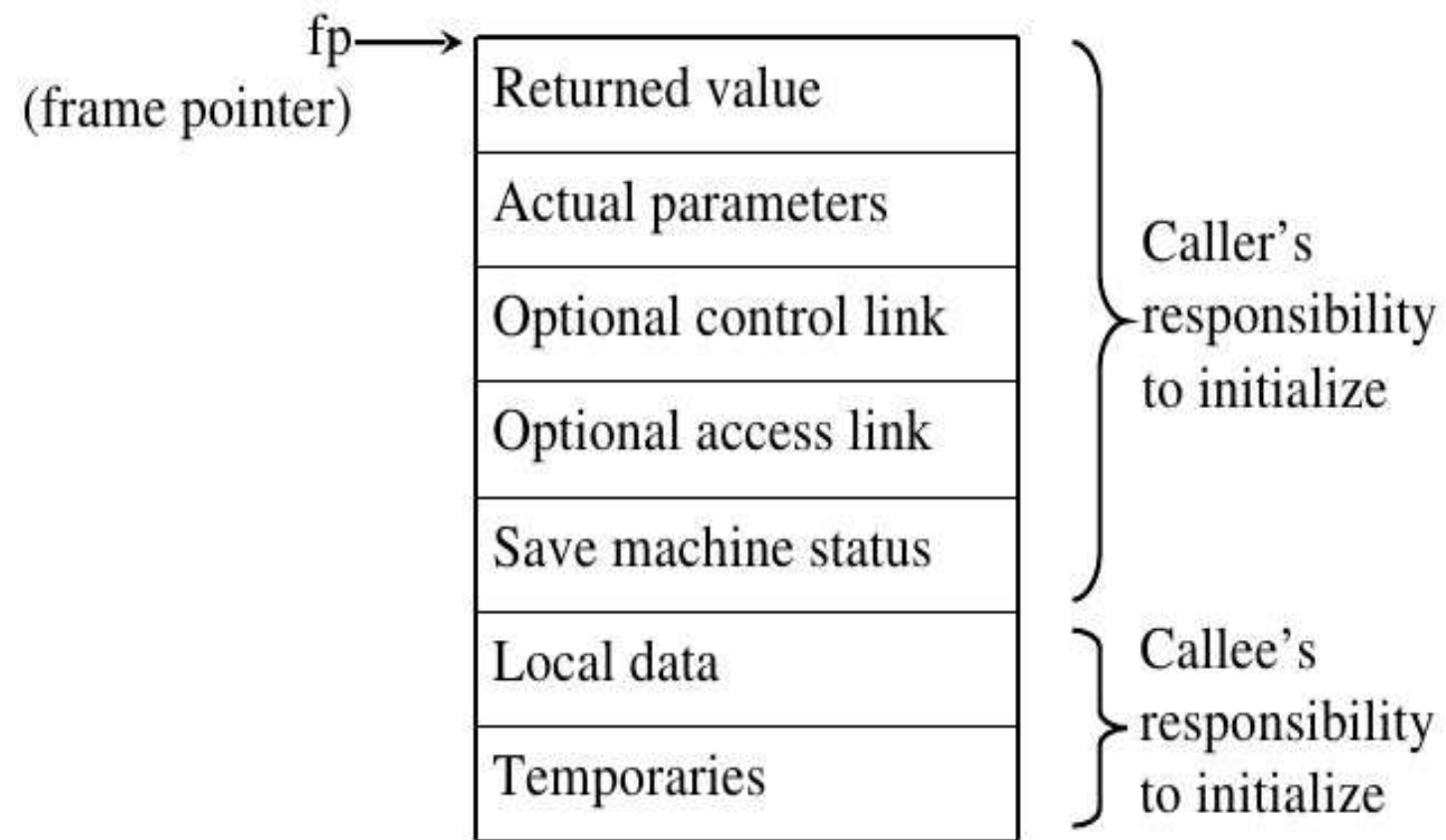
Activation Record

- 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.

The diagram below shows the contents of activation records:

11

Activation Records (Subroutine Frames)



- **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.

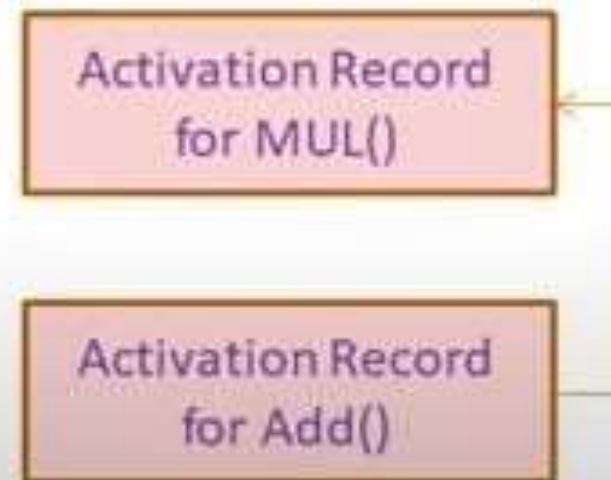
□ **Control Link:** It points to activation record of the caller.

- **Control link**

- It points to the Activation Record of the calling function

- Eg:

```
Mul()
{
    ...
    Add();
    ...
}
```



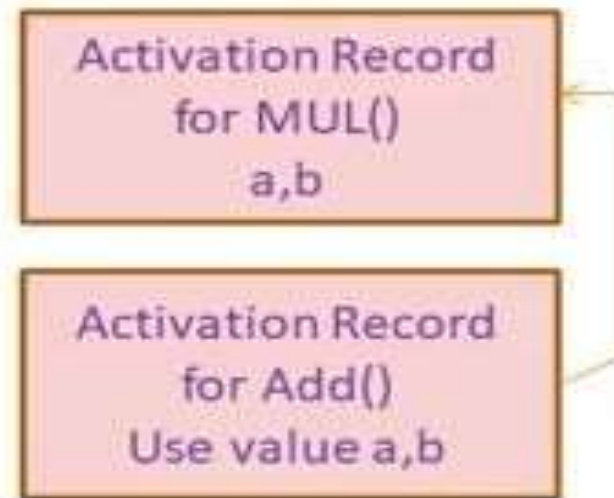
□ **Access Link:** It is used to refer to non-local data held in other activation records.

- **Access Link**

- It refers to the local data of the called function but found in different activation record

Eg:

```
Mul()
{
    Int a,b;
    Add();
    ...
}
Add()
{
    ...
    X=a+b;
    ...
}
```



- **Saved Machine Status:** It holds the information about status of machine before the procedure is called. It also saves the address of the next instruction to be executed
- **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.

BASIC BLOCKS

Introduction

- The basic block is a sequence of consecutive statements which are always executed in sequence without halt or possibility of branching.
- The basic blocks does not have any jump statements among them.
- When the first instruction is executed, all the instructions in the same basic block will be executed in their sequence of appearance without losing the flow control of the program.

Examples



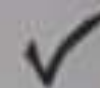
$$a = b + c + d$$

Three address code -

$$t_1 = b + c$$

$$t_2 = t_1 + d$$

$$a = t_2$$



IF A < B then 1 else 0

(1) IF (A < B) goto (4)

(2) T1 = 0

(3) goto (5)

(4) T1 = 1

(5)



Learn
Vid
Fun!

Parameter Passing Mechanisms

Call-by-value

C uses call-by-value everywhere (except macros...)
Default mechanism in Pascal and in Ada

```
callByValue(int y)
{
    y = y + 1;
    print(y);
}

main()
{
    int x = 42;
    print(x);
    callByValue(x);
    print(x);
}
```

output:

x = 42

y = 43

x = 42

x's value does *not*
change when y's
value is changed

Function - Call by address

. In this method the addresses of the actual arguments are copied to the formal arguments. Since the formal arguments point to the actual arguments as they contain addresses of the actual arguments, if we do any changes in the formal arguments they will affect the actual arguments. The formal arguments will be pointers because we have to store addresses in them.

For example (C++):

```
void swap(int *,int *);

void main ( )
{
    int a,b;
    clrscr ( );
    a=10;
    b=20;
    cout<<"Before calling swap function: a="<<a<<" b="<<b;
    swap (&a,&b);
    cout<<"\nAfter calling swap function: a="<<a<<" b="<<b;
    getch ();
}

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
    cout<<"\nInside swap function: x="<<*x<<" y="<<*y;
}
```

Call-by-reference

Available in C++ with the '&' type constructor
(and in Pascal with the `var` keyword)


```
callByRef(int &y)
{
    y = y + 1;
    print(y);
}

main()
{
    int x = 42;
    print(x);
    callByRef(x);
    print(x);
}
```

output:

x = 42
y = 43
x = 43

x's value changes
when y's value
is changed



1. CALL BY NAME

In this method of parameter passing, a subprogram call is the replacement of formal arguments. *formal replace by actual.*
subprogram call by the actual body of the function. Each formal parameter is actually parameters are passed as it without evaluating it. The actual parameter is substituted everywhere for the formal parameter in the body of the called program before execution of the subprogram begins. This type of methods are mostly used when function body contains small number of operations like finding sum, maximum, multiplication, cube of numbers or if-then-else conditional statements. Another name for call by name is call by macro. A macro is a very useful feature in C and C++ programming language. Another example is inline functions in C++, which are similar to macro but type checking is performed for the actual arguments that are passed. As an example of macro consider the C code

```
#define PRINT "hello world"  
#define MAX(a,b) a>b?a:b  
#define SUM(a,b) a+b
```

The first macro is **PRINT** without arguments; second **MAX** and **SUM** both take two arguments. They can be used as

```
dprintf(PRINT);  
x=SUM(10,20);  
y=MAX(10,20);
```

Here before the compilation a special program known as preprocessor checks for the macro in the programs and if there is any, then macro name is replaced by their expansion or body. Here after the preprocessing the code will look like

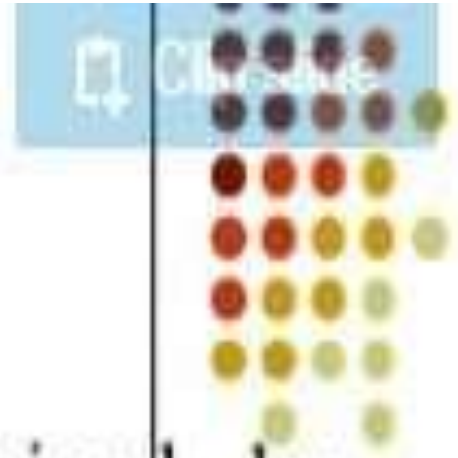
```
printf("hello world");  
x=10+20;  
y=10>20?10:20;
```

Note the actual parameters are substituted for the formal parameters **a** and **b** for **MAX** and **SUM**. The function/macro is not called but the code is expanded at the place of function call.

Symbol Table Management



Symbol Table



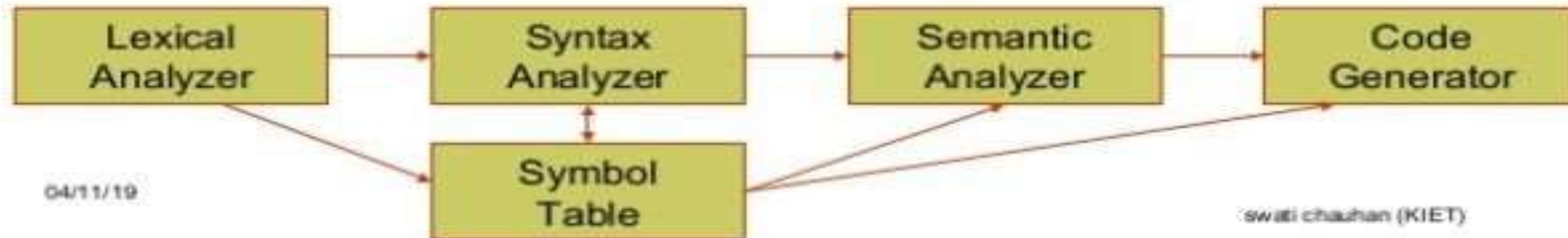
- Symbol table is data structure created and maintained by compilers to store information about the occurrence of various entities such as **variable names, function names, objects, classes, interfaces, etc.**
- Symbol table is used by both the analysis and the synthesis parts of a compiler.

Symbol Table



- When identifiers are found, they will be entered into a symbol table, which will hold all relevant information about identifiers and other symbols, variables, constants, procedures statements e.t.c,
- This information about the name:-
 - Type
 - Its Form , Its Location

It will be used later by the semantic analyzer and the code generator.



04/11/19

swati chauhan (KIET)



JECRC Foundation



**JAIPUR ENGINEERING COLLEGE
AND RESEARCH CENTRE**

*Thank
you!*