K.S.R. COLLEGE OF ENGINEERING (Autonomous)

Vision of the Institution

- We envision to achieve status as an excellent educational institution in the global knowledge hub, making self-learners, experts, ethical and responsible engineers, technologists, scientists, managers, administrators and entrepreneurs who will significantly contribute to research and environment friendly sustainable growth of the nation and the world.

Mission of the Institution

- To inculcate in the students self-learning abilities that enable them to become competitive and considerate engineers, technologists, scientists, managers, administrators and entrepreneurs by diligently imparting the best of education, nurturing environmental and social needs.
- To foster and maintain a mutually beneficial partnership with global industries and Institutions through knowledge sharing, collaborative research and innovation.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

- To create ever green professionals for software industry, academicians for knowledge cultivation and researchers for contemporary society modernization.

Mission of the Department

- To produce proficient design, code and system engineers for software development.
- To keep updated contemporary technology and fore coming challenges for welfare of the society.

Programme Educational Objectives (PEOs)

PEO1 : Figure out, formulate, analyze typical problems and develop effective solutions by imparting the idea and principles of science, mathematics, engineering fundamentals and computing.

PEO2 : Competent professionally and successful in their chosen career through life-long learning.

PEO3 : Excel individually or as member of a team in carrying out projects and exhibit social needs and follow professional ethics.

K.S.R. COLLEGE OF ENGINEERING (Autonomous)

Department of Computer Science and Engineering

Subject Name: Principles of Compiler Design
Subject Code: 16CS511 **Year/Semester:III / V**

Course Outcomes: On completion of this course, the student will be able to

CO1 Know the knowledge about how to implement system software like assembler, loader and linker.
CO2 *Gain knowledge on phases involved in design of compilers and Build NFA and DFA using the formalisms and techniques of context free grammar.*
CO3 Design top down and bottom up parser using algorithms.
CO4 Develop intermediate code and memory space allocation.
CO5 *Validate various techniques of code optimization and generation.*

Program Outcomes (POs) and Program Specific Outcomes (PSOs)

A. Program Outcomes (POs)

Engineering Graduates will be able to :

PO1 **Engineering knowledge:** Ability to exhibit the knowledge of mathematics, science, engineering fundamentals and programming skills to solve problems in computer science.

PO2 **Problem analysis:** Talent to identify, formulate, analyze and solve complex engineering problems with the knowledge of computer science. .

PO3 **Design/development of solutions:** Capability to design, implement, and evaluate a computer based system, process, component or program to meet desired needs.

PO4 **Conduct investigations of complex problems:** Potential to conduct investigation of complex problems by methods that include appropriate experiments, analysis and synthesis of information in order to reach valid conclusions.

PO5 **Modern tool Usage:** Ability to create, select, and apply appropriate techniques, resources and modern engineering tools to solve complex engineering problems.

PO6 **The engineer and society:** Skill to acquire the broad education necessary to understand the impact of engineering solutions on a global economic, environmental, social, political, ethical, health and safety.

PO7 **Environmental and sustainability:** Ability to understand the impact of the professional engineering solutions in societal and Environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO8 **Ethics:** Apply ethical principles and commit to professional ethics and responsibility and norms of the engineering practices.

PO9 **Individual and team work:** Ability to function individually as well as on multi-disciplinary teams.

PO10 **Communication:** Ability to communicate effectively in both verbal and written mode to excel in the career.

PO11 **Project management and finance:** Ability to integrate the knowledge of engineering and management principles to work as a member and leader in a team on diverse projects.

PO12 **Life-long learning:** Ability to recognize the need of technological change by independent and life-long learning.

B. Program Specific Outcomes (PSOs)

PSO1 Develop and Implement computer solutions that accomplish goals to the industry, government or research by exploring new technologies.

PSO2 Grow intellectually and professionally in the chosen field.

UNIT-I

ASSEMBLER, LOADER AND LINKER

1. Define system software.

It consists of variety of programs that supports the operation of the computer. This software makes it possible for the user to focus on the other problems to be solved without needing to know how the machine works internally.

2. Define the basic functions of assembler.

Convert mnemonic operation codes to their machine language equivalents

Convert symbolic operands to their equivalent machine addresses

Build the machine instructions in the proper format

Convert the data constants to internal machine representations

Write the object program and the assembly listing

3. What is meant by assembler directives? Give example.

3. What is meant by assembler directives? Give Example.

These are the statements that are not translated into machine instructions, but they provide instructions to assembler itself.

Example: START, END, BYTE, WORD, RESW and RESB.

4. What are forward references?

It is a reference to a label that is defined later in a program.

Consider the statement

```
10    1000                STL       RETADR

.  .       .                .

.  .       .

80    1036      RETADR    RESW      1
```

The first instruction contains a forward reference RETADR. If we attempt to translate the program line by line, we will unable to process the statement in line10 because we do not know the address that will be assigned to RETADR .The address is assigned later(in line 80) in the program.

5. What are the three different records used in object program?

The header record, text record and the end record are the three different records used in object program.

The header record contains the program name, starting address and length of the program.

Text record contains the translated instructions and data of the program.

End record marks the end of the object program and specifies the address in the program where execution is to begin.

6. What is the need of SYMTAB (symbol table) in assembler?

The symbol table includes the name and value for each symbol in the source program, together with flags to indicate error conditions. Sometimes it may contain details about the data area. SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.

7. Write the steps required to translate the source program to object program.

- Convert mnemonic operation codes to their machine language equivalents.
- Convert symbolic operands to their equivalent machine
- Build the machine instruction in the proper format.
- Write the object program and assembly listing.

8. Define control section.

A control section is a part of the program that maintains its identity after assembly; each control section can be loaded and relocated independently of the others.

Control sections are most often used for subroutines. The major benefit of using control sections is to increase flexibility.

9. What is meant by machine independent assembler features?

The assembler features that do not depend upon the machine architecture are known as machine independent assembler features.

Eg: program blocks, Literals.

10. What are the basic functions of loaders?

- Loading – brings the object program into memory for execution
- Relocation – modifies the object program so that it can be loaded at an address different from the location originally specified.
- Linking – combines two or more separate object programs and also supplies the information needed to reference them.

11. What is meant by bootstrap loader?

This is a special type of absolute loader which loads the first program to be run by the computer. (usually an operating system)

12. What are relative (relocative) loaders?

Loaders that allow for program relocation are called relocating (relocative) loaders.

13. What is the use of modification record?

Modification record is used for program relocation. Each modification record specifies the starting address and the length of the field whose value is to be altered and also describes the modification to be performed.

14. Define Relocation bit method.

If the relocation bit corresponding to a word of object code is set to 1, the program's starting address is to be added to this word when the program is relocated. Bit value 0 indicates no modification is required.

15. What is the use of the variable PROGADDR?

- It gives the beginning address in memory where the linked program is to be loaded.
- The starting address is obtained from the operating system.

16. Write the two passes of a linking loader.

Pass1: assigns address to all external symbols

Pass 2: it performs actual loading, relocation and linking.

17. Give the functions of the linking loader.

The linking loader performs the process of linking and relocation. It includes the operation of automatic library search and the linked programs are directly loaded into the memory.

18. Give the difference between linking loader and linkage editors.

Linking loader	Linkage editor
The relocation and linking is performed each time the program is loaded	It produces a linked version of a program and which is written in a file for later Execution
Here the loading can be accomplished in a single pass	Two passes are required

19. **What is meant by static executable and dynamic executable?**

- In static executable, all external symbols are bound and ready to run.
- In dynamic executables some symbols are bound at run time.

16 Marks

1. Explain in detail about basic assembler functions.

A simple SIC assembler

Assembler Algorithm

Data structures

2. Explain about the machine-Dependent Assembler features.

Instruction formats

Addressing modes

Program Relocation

3. Discuss in detail about the machine-Independent Assembler features.

Literals

Symbol-Defining
Statements Expressions

Program blocks

Control sections and Program Linking

4. Explain in detail about the assembler Design options.

One-pass Assembler

Multi-pass Assembler

5. Explain in detail about basic loader functions.

Design of an Absolute Loader
A simple Bootstrap loader.

6. Explain about Machine -Dependent Loader Features.

Relocation

Program Linking

Algorithm

Data structures

7.Discuss in detail about Machine-independent Loader features

Automatic Library

Search Loader Options

8.Explain about the Loader Design Options.

Linkage Editor

Dynamic linking

Bootstrap loaders

UNIT-II

COMPILER AND LEXICAL ANALYSIS

1. What is a Complier?

A Complier is a program that reads a program written in one language-the source language-and translates it in to an equivalent program in another language-the target language . As an important part of this translation process, the compiler reports to its user the presence of errors in the source program

2. State some software tools that manipulate source program?

i. Structure editors
ii. Pretty printers
iii. Static
iv. Checkers
v. Interpreters.

3.What are the cousins of compiler? April/May 2004, April/May 2005

The following are the cousins of

i. Pre-processors
ii. Assemblers
iii. Loaders
iv. Link editors.

4. What are the main two parts of compilation? What are they performing?

The two main parts are

- **Analysis** part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.

- **Synthesis** part constructs the desired target program from the intermediate representation.

5.What is a Structure editor?

A structure editor takes as input a sequence of commands to build a source program .The structure editor not only performs the text creation and modification functions of an ordinary text editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.

6.What are a Pretty Printer and Static Checker?

- A Pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visible.

- A static checker reads a program, analyses it and attempts to discover potential bugs with out running the program.

7. How many phases does analysis consists?

Analysis consists of three phases

i .Linear analysis

ii. Hierarchical analysis
iii. Semantic analysis

8. What happens in linear analysis?

This is the phase in which the stream of characters making up the source program is read from left to right and grouped in to tokens that are sequences of characters having collective meaning.

9. What happens in Hierarchical analysis?

This is the phase in which characters or tokens are grouped hierarchically in to nested collections with collective meaning.

10. What happens in Semantic analysis?

This is the phase in which certain checks are performed to ensure that the components of a program fit together meaningfully.

11. What is a Loader? What does the loading process do?

A Loader is a program that performs the two functions

i. Loading

ii .Link editing

The process of loading consists of taking relocatable machine code, altering the relocatable address and placing the altered instructions and data in memory at the proper locations.

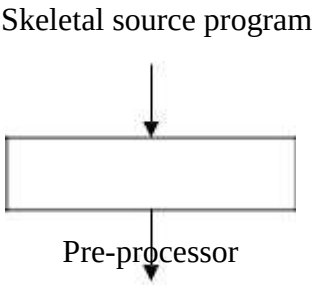
13. What does the Link Editing does?

Link editing: This allows us to make a single program from several files of relocatable machine code. These files may have been the result of several compilations, and one or more may be library files of routines provided by the system and available to any program that needs them.

14. What is a preprocessor?

A pre-processor is one, which produces input to compilers. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a distinct program called a pre-processor.

The pre-processor may also expand macros into source language statements.



15. State some functions of Pre-processors

i) Macro processing
ii) File inclusion
iii) Relational Pre-processors
iv) Language extensions

16. What is a Symbol table?

A Symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

17. State the general phases of a compiler

i. Lexical analysis
ii. Syntax analysis
iii. Semantic analysis
iv. Intermediate code generation
v. Code optimization
vi. Code generation

18. What is an assembler?

Assembler is a program, which converts the source language in to assembly language.

19.What is the need for separating the analysis phase into lexical analysis and parsing?

(Or) What are the issues of lexical analyzer?

vii. Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases.

18. Compiler efficiency is improved.

19. Compiler portability is enhanced.

20.What is Lexical Analysis?

The first phase of compiler is Lexical Analysis. This is also known as linear analysis in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.

21. What is a lexeme? Define a regular set.

- A Lexeme is a sequence of characters in the source program that is matched by the pattern for a token.
- A language denoted by a regular expression is said to be a regular set

22. What is a sentinel? What is its usage?

A Sentinel is a special character that cannot be part of the source program. Normally we use 'eof' as the sentinel. This is used for speeding-up the lexical analyzer.

23. What is a regular expression? State the rules, which define regular expression? Regular expression is a method to describe regular language

Rules:

1) ε-is a regular expression that denotes {ε} that is the set containing the empty string
2) If a is a symbol in ∑,then a is a regular expression that denotes {a}

3) Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then,
   a) (r)/(s) is a regular expression denoting L(r) U L(s).
   b) (r)(s) is a regular expression denoting L(r)L(s)
   c) (r)* is a regular expression denoting L(r)*.
   d) (r) is a regular expression denoting L(r).

24. What are the Error-recovery actions in a lexical analyzer?

1. Deleting an extraneous character
2. Inserting a missing character
3. Replacing an incorrect character by a correct character

4. Transposing two adjacent characters

25. Construct Regular expression for the language

L= {w ε{a,b}/w ends in abb}

Ans:{a/b}*abb.

26. What is recognizer?

Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine, otherwise it is rejected.

27. Write short notes on buffer pair.

It is a specialized buffering technique used to reduce the overhead required to process an input character. Buffer is divided into two N-character halves. Use two pointers. Used at times when the lexical analyzer needs to look ahead several characters beyond the lexeme for a pattern before a match is announced.

28. Differentiate tokens, patterns, lexeme.

- Tokens- Sequence of characters that have a collective meaning.
- Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token
- Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.

29 List the operations on languages.

- **Union -** L U M ={s | s is in L or s is in M}
- **Concatenation** – LM ={st | s is in L and t is in M}
- **Kleene Closure –** L* (zero or more concatenations of L)
- **Positive Closure –** L+ (one or more concatenations of L)

30. Write a regular expression for an identifier.

An identifier is defined as a letter followed by zero or more letters or digits.The regular expression for an identifier is given as **letter (letter | digit)***

31. Mention the various notational shorthand's for representing regular expressions.

- One or more instances (+)
- Zero or one instance (?)
- Character classes ([abc] where a,b,c are alphabet symbols denotes the regular expressions a | b | c.)
- Non regular sets

32. State some compiler construction tools?

i. Parse generator
ii. Scanner generators
iii. Syntax-directed translation engines
iv. Automatic code generator
v. Data flow engines.

16 MARKS

1. Phases of Compiler.

A Compiler operates in phases, each of which transforms the source program from one representation into another. The following are the phases of the compiler:

Main phases:

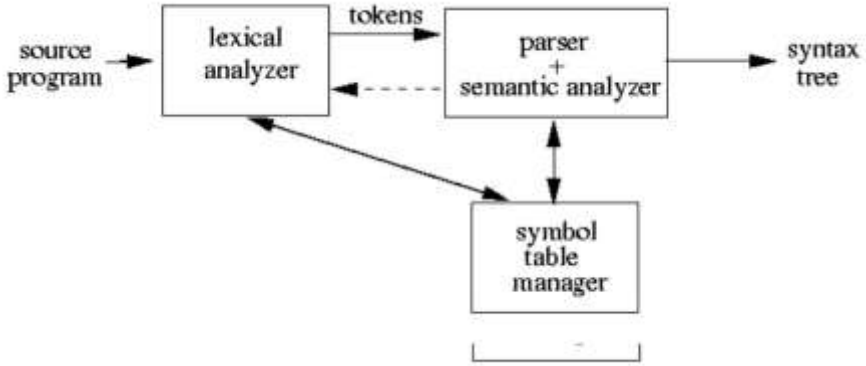
1) Lexical analysis
2) Syntax analysis
3) Semantic analysis
4) Intermediate code generation
5) Code optimization
6) Code generation

2. Compiler Construction tools.

The following are the compiler construction tools:

1) **Parser Generators**
2) **Scanner Generator**
3) **Syntax-Directed Translation**
4) **Automatic Code Generators**
5) **Data-Flow Engines**

3.R**oles and tasks of a lexical analyzer.**



4. Converting a Regular Expression into a DFA.

5. Specification and recognition of tokens.

6. **Minimization of DFA.**

UNIT III SYNTAX ANALYSIS

1. **Define parser.**

A **parser** takes input in the form of a sequence of tokens or program instructions and usually builds a data structure in the form of a parse tree or an abstract syntax tree.

2. **Mention the basic issues in parsing.**

There are two important issues in parsing.

- Specification of syntax
- Representation of input after parsing.

3. **Why lexical and syntax analyzers are separated out?**

Reasons for separating the analysis phase into lexical and syntax analyzers:

- Simpler design.
- Compiler efficiency is improved.
- Compiler portability is enhanced.

4. **Define a context free grammar.**

A context free grammar G is a collection of the following

- V is a set of non terminals
- T is a set of terminals
- S is a start symbol
- P is a set of production rules

G can be represented as G = (V,T,S,P)

Production rules are given in the following

form

Non terminal → (V U T)*

5. **Briefly explain the concept of derivation.**

Derivation from S means generation of string w from S. For constructing derivation two things are important.

i) Choice of non terminal from several others.

ii) Choice of rule from production rules for corresponding non

terminal. Instead of choosing the arbitrary non terminal one can choose

i) either leftmost derivation – leftmost non terminal in a sentinel form

ii) or rightmost derivation – rightmost non terminal in a sentinel form

6. **Define bottom up parsing?**

It attempts to construct a parse tree for an input string is beginning at leaves and working up towards the root (i.e.) reducing a string „w‟ to the start symbol of a grammar. At each reduction step, a particular substring matching the right side of a production is replaced by the symbol on the left of that production. It is a rightmost derivation and it‟ s also known as shifts reduce parsing.

7. **Define ambiguous grammar.**

A grammar G is said to be ambiguous if it generates more than one parse tree for some sentence of language L(G).

i.e. both leftmost and rightmost derivations are same for the given sentence.

8. **What is a operator precedence parser?**

A grammar is said to be operator precedence if it possess the following properties:

No production on the right side is ε.

There should not be any production rule possessing two adjacent non terminals at the right hand side.

9. **List the properties of LR parser.**

LR parsers can be constructed to recognize most of the programming languages for which the context free grammar can be written.

The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.

LR parsers work using non backtracking shift reduce technique yet it is efficient one.

10. **Mention the types of LR parser.**

- SLR parser- simple LR parser
- LALR parser- lookahead LR parser
- Canonical LR parser

11. **What are the problems with top down parsing?**

The following are the problems associated with top down parsing:

- Backtracking
- Left recursion

- Left factoring
- Ambiguity

12. **Write the algorithm for FIRST and FOLLOW.**

FIRST

If X is terminal, then FIRST(X) IS {X}.

If X → ε is a production, then add ε to FIRST(X).

If X is non terminal and X → Y1,Y2..Yk is a production, then place a in FIRST(X) if for some i , a is in FIRST(Yi) , and ε is in all of FIRST(Y1),…FIRST(Yi-1);

FOLLOW

1. Place $ in FOLLOW(S), where S is the start symbol and $ is the input right end marker.

2. If there is a production A → αBβ, then everything in FIRST (β) except for ε is placed in FOLLOW (B).

3. If there is a production A → αB, or a production A→ αBβ where FIRST (β) contains ε , then everything in FOLLOW (A) is in FOLLOW (B).

13. **List the advantages and disadvantages of operator precedence parsing. Advantages**

This typeof parsing is simple to implement.

Disadvantages

The operator like minus has two different precedence(unary and binary).Hence it is hard to handle tokens like minus sign.

This kind of parsing is applicable to only small class of grammars.

14. **What is dangling else problem?**

Ambiguity can be eliminated by means of dangling-else grammar which is show below: stmt → if expr then stmt

| if expr then stmt else
stmt | other

15. **Write short notes on YACC.**

YACC is an automatic tool for generating the parser program.

YACC stands for Yet Another Compiler Compiler which is basically the utility available from UNIX.

Basically YACC is LALR parser generator.

It can report conflict or ambiguities in the form of error messages.

16. **What is meant by handle pruning?**

A rightmost derivation in reverse can be obtained by handle pruning.

If w is a sentence of the grammar at hand, then w = γn, where γn is the nth right-sentential form of some as yet unknown rightmost derivation

S = γ0 => γ1…=> γn-1 => γn = w

16. **Define LR (0) items.**

An LR(0) item of a grammar G is a production of G with a dot at some position of the right side. Thus, production A → XYZ yields the four items

A→**.**XYZ

A→X**.**YZ A→XY.Z A→XYZ**.**

17. **What is meant by viable prefixes?**

The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. An equivalent definition of a viable prefix is that it is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

18. **Define handle.**

A handle of a string is a substring that matches the right side of a production, and whose reduction to the non terminal on the left side of the production represents one step along the reverse of a rightmost derivation.

A handle of a right – sentential form γ is a production A→β and a position of γ where the string β may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of γ. That is , if S =>αAw =>αβw,then A→β in the position following α is a handle of αβw.

19. **What are kernel & non-kernel items?**

Kernel items, which include the initial item, S'→ **.**S, and all items whose dots are not at the left end.

Non-kernel items, which have their dots at the left end.

20. **What is phrase level error recovery?**

Phrase level error recovery is implemented by filling in the blank entries in the predictive parsing table with pointers to error routines. These routines may change, insert, or delete symbols on the input and issue appropriate error messages. They may also pop from the stack.

16 Marks

1) Construct a predictive parsing.table for the Grammar
E → E + T / F

T→ T * F / F

F → (E) / id

2) Give the LALR parsing table for the grammar.

S → L = R / R

L→ * R / id

R→ L

3) Consider the Grammar

E →TE'

E'→ + TE' / E

T → FT'

T' → *FT' / E

F→ (E) / id

Construct a predictive parsing table for the grammar shown above. Verify whether the input string id + id * id is accepted by the grammar or not.

4) Consider the grammar.

E→ E + T

E → T

T → T *

F

T→ F

F→ (E) / id

Construct an LR parsing table for the above grammar. Give the moves of the LR parser on id * id + id

5) For the grammar given below, calculate the operator precedence relation and the precedence functions

E -> E + E | E – E |

| E * E | E | E |

| E ^ E | (E) |

| -E | id

6) Compare top down parsing and bottom up parsing methods.

7) What are LR parsers? Explain with a diagram the LR parsing algorithm.

8) Explain recursive descent parser with appropriate examples.

9) Compare SLR, LALR and LR parses.

UNIT IV – INTERMEDIATE CODE AND RUN TIME ENVIRONMENT

1. List the different storage allocation strategies.

The strategies are:

- Static allocation
- Stack allocation
- Heap allocation

2. What are the contents of activation record?

The activation record is a block of memory used for managing the information needed by a single execution of a procedure. Various fields for activation record are:

- Temporary variables
- Local variables
- Saved machine registers
- Control link
- Access link
- Actual parameters
- Return values

3. What is dynamic scoping?

In dynamic scoping a use of non-local variable refers to the non-local data declared in most recently called and still active procedure. Therefore each time new findings are set up for local names called procedure. In dynamic scoping symbol tables can be required at run time.

4. Define symbol table.

Symbol table is a data structure used by the compiler to keep track of semantics of the variables. It stores information about scope and binding information about names.

7. What are the various ways to pass a parameter in a function?

- Call by value
- Call by reference
- Copy-restore
- Call by name

8. Suggest a suitable approach for computing hash function.

Using hash function we should obtain exact locations of name in symbol table.

The hash function should result in uniform distribution of names in symbol table.

The hash function should be such that there will be minimum number of collisions. Collision is such a situation where hash function results in same location for storing the names.

10. What are the functions used to create the nodes of syntax trees?

Mknode (op, left, right)

Mkleaf (id,entry)

Mkleaf (num, val)

11. What are the functions for constructing syntax trees for expressions?

i) The construction of a syntax tree for an expression is similar to the translation of the expression into postfix form.

ii) Each node in a syntax tree can be implemented as a record with several fields.

12. Give short note about call-by-name?

Call by name, at every reference to a formal parameter in a procedure body the

name of the corresponding actual parameter is evaluated. Access is then made to the effective parameter.

13. How parameters are passed to procedures in call-by-value method?

This mechanism transmits values of the parameters of call to the called program. The transfer is one way only and therefore the only way to returned can be the value of a function.

```
Main ( )
{ print (5);
  } Int
  Void print (int n)
  { printf (“%d”, n); }
```

14. Define static allocations and stack allocations

Static allocation is defined as lays out for all data objects at compile time.

Names are bound to storage as a program is compiled, so there is no need for a run time support package.

Stack allocation is defined as process in which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, and activation records are pushed and popped as activations begin and end.

17. Define a syntax-directed translation?

Syntax-directed translation specifies the translation of a construct in terms of Attributes associated with its syntactic components. Syntax-directed translation uses a context free grammar to specify the syntactic structure of the input. It is an input- output mapping.

18. Define an attribute. Give the types of an attribute?

An attribute may represent any quantity, with each grammar symbol, it

associates a set of attributes and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production.

Example: a type, a value, a memory location etc.,

i) Synthesized attributes.

ii) Inherited attributes.

19. Give the 2 attributes of syntax directed translation into 3-address code?

i) E.place, the name that will hold the value of E and

ii) E.code , the sequence of 3-addr statements evaluating E.

20. Write the grammar for flow-of-control statements?

The following grammar generates the flow-of-control statements, if-then, if-

then-else, and while-do statements.

S -> if E
then S1

| If E then S1

else S2

| While E do S1.

16 Marks

1) What is a three address code? Mention its types. How would you implement these address statements? Explain with suitable examples.
2) Discuss in detail about the run time storage arrangement.
3) Explain how declarations are done in a procedure using syntax directed translations.
4) .
4) Write syntax directed translation for arrays.
5) Explain about back patching with an example.

UNIT V

CODE OPTIMISATION AND CODE GENERATION

1. Define code generation.

It is the final phase in compiler model and it takes as an intermediate representation of the source program as a input and produces as equivalent target program as a output then intermediate instructions are translated into a sequence of machine instructions that perform the same task.

2. What are the issues in the design of code generator?

Input to the generator

- Target program

Memory management

- Instruction selection

Register allocation

- Choice of evaluation order

Approaches to code generation.

3. Give the variety of forms in target program.

- Absolute machine language.
- Relocatable machine language.

Assembly language.

4. Give the factors of instruction selections.

Uniformity and completeness of the instruction sets

Instruction speed and machine idioms

Size of the instruction sets.

5. What is code motion?

Code motion is an optimization technique in which amount of code in a loop is decreased. This transformation is applicable to the expression that yields the same result independent of the number of times the loop is executed. Such an expression is placed before the loop.

6. What are the properties of optimizing compiler?

The source code should be such that it should produce minimum amount of target code.

There should not be any unreachable code.

Dead code should be completely removed from source language.

The optimizing compilers should apply following code improving transformations on source language.

i) common sub expression elimination

ii) dead code elimination

iii) code movement

iv) strength reduction

5. What are the sub problems in register allocation strategies?

During register allocation, we select the set of variables that will reside in register at a point in the program.

During a subsequent register assignment phase, we pick the specific register that a variable reside in.

6. Give the standard storage allocation strategies.

Static allocation

Stack allocation.

7. Define static allocations and stack allocations

Static allocation is defined as lays out for all data objects at compile time. Names are bound to storage as a program is compiled, so there is no need for a Run time support package.

Stack allocation is defined as process in which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, and activation records are pushed and popped as activations begin and end.

8. Define basic block and flow graph.

A basic block is a sequence of consecutive statements in which flow of Control enters at the beginning and leaves at the end without halt or possibility Of branching except at the end.

A flow graph is defined as the adding of flow of control information to the Set of basic blocks making up a program by constructing a directed graph.

9. Write the step to partition a sequence of 3 address statements into basic blocks.

1. First determine the set of leaders, the first statement of basic blocks.

- The rules we can use are the following.
- The first statement is a leader.
- Any statement that is the target of a conditional or unconditional goto is a leader.
- Any statement that immediately follows a goto or conditional goto statement is a leader.

2. For each leader, its basic blocks consists of the leader and all statements Up to but not including the next leader or the end of the program.

10. Give the important classes of local transformations on basic blocks

- Structure preservation transformations
- Algebraic transformations.

11. Describe algebraic transformations.

It can be used to change the set of expressions computed by a basic blocks into a algebraically equivalent sets. The useful ones are those that simplify the

Expressions place expensive operations by cheaper ones.

X = X+ 0

X = X * 1

12. What is meant by register descriptors and address descriptors?

A register descriptor keeps track of what is currently in each register. It is consulted whenever a new register is needed.

An address descriptor keeps track of the location where ever the current value of the name can be found at run time. The location might be a register, a stack location, a memory address.

13. What are the actions to perform the code generation algorithms?

- Invoke a function get reg to determine the location L.
- Consult the address descriptor for y to determine y‟, the current location of y.
- If the current values of y and/or z have no next uses, are not live on exit from the block, and are in register, alter the register descriptor.

14. Write the code sequence for the d:=(a-b)+(a-c)+(a-c).

Statement	Code generation	Register descriptor	Address descriptor
t:=a-b	MOV a,R0 SUB b,R0	R0 contains t	t in R0
u:=a-c	MOV a,R1 SUB c,R1	R0 contains t R1 contains u	t in R0 u in R1
v:=t+u	ADD R1,R0	R0 contains v R1 contains u	u in R1 v in R0
d:=v+u	ADD R1,R0 MOV R0,d	R0 contains d	d in R0 d in R0 and memory

15. Write the labels on nodes in DAG.

A DAG for a basic block is a directed acyclic graph with the following Labels on nodes:

- Leaves are labeled by unique identifiers, either variable names or constants.

  Interior nodes are labeled by an operator symbol.

- Nodes are also optionally given a sequence of identifiers for labels.

16. Give the applications of DAG.

- Automatically detect the common sub expressions

  Determine which identifiers have their values used in the block.

- Determine which statements compute values that could be used outside the blocks.

17. Define Peephole optimization.

A Statement by statement code generation strategy often produces target code that contains redundant instructions and suboptimal constructs. “Optimizing” is misleading because there is no guarantee that the resulting code is optimal. It is a method for trying to improve the performance of the target program by examining the short sequence of target instructions and replacing this instructions by shorter or faster sequence.

18. Write the characteristics of peephole optimization?

- Redundant-instruction elimination
- Flow-of-control optimizations.
- Algebraic simplifications
- Use of machine idioms

19. What are the structure preserving transformations on basic blocks?

- Common sub-expression elimination
- Dead-code elimination
- Renaming of temporary variables
- Interchange of two independent adjacent statement

20. Define Common sub-expression elimination with example.

It is defined as the process in which eliminate the statements which has the Same expressions. Hence this basic block may be transformed into the equivalent Block.

Ex:

a : =b + c b

:=a - d

c :=b + c

After elimination:

a : =b + c b

:=a - d

c :=a

21. Define Dead-code elimination with example.

It is defined as the process in which the statement x=y+z appear in a basic block, where x is a dead that is never subsequently used. Then this statement maybe safely removed without changing the value of basic blocks.

22. Define Renaming of temporary variables with example..

We have the statement u:=b + c ,where u is a new temporary variable, and change all uses of this instance of t to u, then the value of the basic block is not changed.

23. Define reduction in strength with example.

Reduction in strength replaces expensive operations by equivalent cheaper ones on the target machines. Certain machine instructions are cheaper than others and can often be used as special cases of more expensive operators.

Ex:

X^2 is invariably cheaper to implement as x*x than as a call to an exponentiation routine.

24. Define use of machine idioms.

The target machine may have harder instructions to implement certain specific operations efficiently. Detecting situations that permit the use of these instructions can reduce execution time significantly.

25. Define code optimization and optimizing compiler

The **term code-optimization** refers to techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program. Compilers that apply code-improving transformations are called optimizing-compilers.

16 Marks

1. What are the issues in the design of code generator? Explain in detail.
2. Explain basic blocks and flow graphs.
3. Explain about transformation on a basic block.
4. Write a code generation algorithm. Explain about the descriptor and function getreg().Give an example.
5. Explain peephole optimization
6. Explain DAG representation of basic blocks.
7. Explain principle sources of code optimization in details.
8. Explain the Source language issues with details.

9. Explain about Optimization of basic blocks.

10. Explain simple code generator with suitable example.

11. Discuss about the following:

    a. Copy Propagation b) Dead-code Elimination and c) Code motion