SYED AMMAL ENGINEERING COLLEGE
(An ISO 9001: 2008 Certified Institution)
Dr. E.M.Abdullah Campus, Ramanathapuram – 623 502
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS6660-Compiler Design

Part A

UNIT I-INTRODUCTION TO COMPILERS

1. What is a language processing system?

In language processing system the source program is first preprocessed preprocessors. The modified source program is processed by compiler to form target assembly code which is then translated by assembler to generate relocatable object codes that are processed by linker and loader to generate target program

2. Define compiler.

A compiler is a program that reads a program written in one language (source language) and translates it into an equivalent program in another language (target language).This translation process, the compiler reports to its users the presence of errors in the source program.

3. Differentiate compiler and interpreter.

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Debug line by line Ex: Perl ,Ruby, Python, etc	Generates error after scanning entire program. Ex: C++ Compiler, GCC

4. **Define concrete and abstract syntax tree with example.**

   Abstract syntax tree- It is a syntax tree in which each node represents an operator and the children of the node represent the operand.

   Concrete syntax tree-A parse tree is called concrete syntax tree. A parse tree pictorially shows how the start symbol of a grammar derives a string in the language.If a production is of the form A→XYZ, then a parse tree may have an interior node labeled A with children labeled X,Y and Z.

5. **List four software tools that analyses the source program.**

   →Structure editors
   →Pretty printers
   →Static checkers
   →Interpreters

6. **What are the cousins of compiler?**

   - Preprocessor
     - Macro processing
     - File inclusion
     - Rational preprocessors
     - Language extension
   - Compiler
   - Assembler
   - Loader and Link editors
   -

7. **Define assembler and write its types**

   It is defined by the low level language is assembly language and high level language is machine language is called assembler.

   - One pass assembler
   - Two pass assembler

8. **What are the classifications of compiler?**

   i) Single pass compiler
   ii) Multi pass compiler
   iii) Load-and-go compiler
   iv) Debugging or optimizing compiler

9. Define preprocessor

What are the functions of preprocessors?

A preprocessor produces input to compilers.They may perform the following functions:

- Macro processing
- File inclusion
- Rational preprocessors
- Language extension

10. What is the impact of more number of phases in compilation?

It takes more time and space

11. List the compiler construction tools.

→Parser generators

→Scanner generators

→Syntax directed translation engine

→Automatic code generators

→Data flow engines

12. What do you meant by passes?

A pass reads the source program or the output of the previous pass, makes the transformations specified by its phases and writes output into an intermediate file, which may then be read by a subsequent pass. In an implementation of a compiler, portions of one or more phases are combined into a module called pass.

13. What is front end and back end?

The phases are collected into a front end and a back end. The front end consists of those phases or parts of phases, that depends primarily on the source language and is largely independent of the target machine. The back ends that depend on the target machine and generally these portions do not depend on the source language.

14. What do you meant by phases?

Each of which transforms the source program one representation to another. A phase is a logically cohesive operation that takes as input one

representation of the source program and produces as output another representation

15.List the phases of compiler

- Lexical analysis
- Syntax analysis
- Semantic analysis
- Intermediate code generation
- Code Optimization
- Code generation

16.What is meant by lexical analysis?

It reads the characters in the program and groups them into tokens that are sequences of characters having a collective meaning Such as an identifier, a keyword, a punctuation, character or a multi-character operator like ++.

17.What is meant by syntax analysis?

It processes the string of descriptors, synthesized by the lexical analyzer, to determine the syntactic structure of an input statement. This process is known as parsing. Output of the parsing step is a representation of the syntactic structure of a statement. It is represented in the form of syntax tree.

18.What is meant by semantic analysis?

This phase checks the source program for semantic errors and gathers type of information for the subsequent phase.

19.What is meant by intermediate code generation?

After syntax and semantic analysis, some compilers generate an explicit intermediate representation of the source program. It can have a variety of forms. This form called three-address code. It consists of sequence of instructions, each of which has at most three operands.

20.What are the formats of Intermediate code?

- Three address code
- Postfix Notation
- Syntax tree notation

21.What are the properties of Intermediate Code

- It must be easy to produce.
- It must be easy to translate

22.Define three address code.

It is a sequence of instruction in which each instruction must have atmost three operands and two operators.It is the output of Intermediate code generation.

Ex: t1:=id1+10
Id2:=t1

23.What are the properties of three address code?

- Each instruction must have at most one operator in addition to assignment operator.
- Compiler must generate a temporary name to hold the value computed by each instruction.
- Some three address instructions have fewer than three operands.

24.Define Symbol Table.

A symbol table is a data structure containing a record for each identifier, with fields for the attributes like name, location, type and size of the identifier. A compiler uses a symbol table to keep track of scope and binding information about variable names of variables. Insertion, search, deletion and updation can be done in the table.

25.Write short notes on symbol table manager

The table management or bookkeeping portion of the compiler keeps track of the names used by program and records essential information about each, such as its type (int, real etc.,) the data structure used to record this information is called a symbol table manger.

26.Write short notes on error handler

The error handler is invoked when a flaw in the source program is detected. It must warn the programmer by issuing a diagnostic, and adjust the information being passed from phase to phase so that each phase can

proceed. So that as many errors as possible can be detected in one compilation.

27.What are the formats of target code?

- Assembly language format
- Relocatable binary format
- Memory image format

28.List the types of compiler

Incremental Compiler-Recompiles only modified portion of source program

Cross compiler-Compiler which runs in one machine and generates target code for another machine.

UNIT II

LEXICAL ANALYSIS

1. What are the issues in lexical analysis?

i) Simpler design
ii) Compiler efficiency is improved by specialized buffering techniques for reading input characters and processing tokens and significantly speeds up the performance of a compiler
iii) Compiler portability is enhanced.

2. Differentiate lexeme, tokens and patterns.

A set of strings in the input described by a rule called **pattern** associated with the token.

A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Token is a logical cohesive sequence of characters that can be treated as a single unit

3. Give the algebraic properties of regular expression

i) r/s = s/r / is commutative
ii) r/(s/t)=(r/s)/t / is associative

iii) (rs)t=r(st) concatenation is associative
iv) r(s/t)=rs/rt concatenation distributes over /
v) r**=r* * is idempotent

4. **What are the operations on language?**
   - Union
   - Concatenation
   - Kleene closure or star closure and
   - Star closure.

5. **Give the error recovery actions in lexical errors?**
   - Deleting an extraneous character
   - Inserting a missing character
   - Replacing an incorrect character by a correct character.
   - Transposing two adjacent characters

6. **Define regular expression?**

   It is built up out of simpler regular expression using a set of defining rules. Each regular expression r denotes a language L(r). The defining rules specify how L(r) is formed by combining in various ways the languages denoted by the sub expressions of r.

7. **Give the precedence of regular expression operator?**
   - The unary operator * has the highest precedence and is left associative.
   - Concatenation has the second highest precedence and is left associative.
   - / has the lowest precedence and is left associative.

8. **Give the rules in regular expression?**

   1) € is a regular expression that denotes {€}, that is the set containing the empty string.

   2) If „a‟ is a symbol in Σ , then a is a regular expression that denoted {a} , i.e., the set containing the string a.

   3) Suppose r and s are regular expression denoting the languages L(r) and L(s).

9. **Give the types of notational shorthand's of RE?**
   - Zero or more instance -*

- One or more instance -+
- Zero or one instance-?
- Character classes -[]

10.Define kleene closure or star closure and positive closure.

Star closure of L, denoted L^*, is the set of those strings that can be formed by taking any number of strings from L and concatenating all of them.

$L^* = Li$

Positive closure of L ,denoted L^+, is the set of those strings that can be Formed by one or more concatenations of L

$L^+ = Li$

11.Define character class with example.

The notation [abc] where a, b, c are alphabet symbols denotes the regular expression a/b/c.

Example:

[A-z] = a | b | c | -------| z

Regular expression for identifiers using character classes

[a – z A – Z] [A – Z a – z 0 – 9] *

12.Write the R.E. for the following language.

i. Set of statements over {a,b,c} that contain no two consecutive b‟s

(b/c) (A/c/ab/cb) *

ii. Set of statements over {a,b,c} that contain an even number of a‟s

((b/c)* a (b/c) * a)* (b/c)*

13. Describe the language denoted by the following R.E.

i. (0/1)*0(0/1)(0/1)

The set of all strings of 0's and 1's with the third symbol from the right end is 0.

ii. 0(0/1)*0

The set of all strings of 0's and 1's starting and ending with 0.

iii. (00/11)*((01/10)(00/11)*(01/10)(00/11)*)

The set of all strings of 0's and 1's with even number of 0's and 1's

14. Define regular set?

A language denoted by a regular expression is said to be a regular set.

15. What are the tasks in lexical analyzer?

- Primary task is stripping out from the source program comments and white space in the form of blank, tab, new line characters.
- Secondary task is correlating error messages from the compiler with the source program.

16.Define finite automata and write its types.

A recognizer for a language is a program that takes as input a string x and answers "yes" if x is a sentence of the language and "no" otherwise.

A better way to convert a regular expression to a recognizer is to construct a generalized transition diagram from the expression. This diagram is called a finite automation. The types are:

1. Deterministic (DFA)
2. Non-deterministic (NFA)

17. What are the three parts of lexical program?

Declarations
%%
Translation rules
%%
Auxiliary procedures

18. What are the four functions of converting regular expression to DFA directly?

- Nullable (n)
- Firstpos (n)
- Last (n)
- Followpos (n)

UNIT III

SYNTAX ANALYSIS

1. What do you meant by parser and its types?

A parser for grammar G is a program that takes as input a string w and produces as output either a parse tree forw, if w is a sentence of G, or an error message indicating that w is not a sentence of G. It obtains a string of tokens from the lexical analyzer, verifies that the string is generated by the grammar for the source language.

a) Top down parsing
b) Bottom up parsing

2. What are the goals of error handler in a parser?

i) It should report the presence of errors clearly and accurately
ii) It should recover from each error quickly enough to be able to detect subsequent errors
iii) It should not significantly slow down the processing of correct programs.

3. What are error recovery strategies in parser?

a) Panic mode
b) Phrase level
c) Error productions
d) Global corrections

4. Define CFG.

Many programming language has rules that prescribe the syntactic structure of well-formed programs. The syntax of programming language constructs can be described by CFG.

CFG is denoted as G=(V,T,P,S)
Where V=Variable or Non Terminal
T=Terminals
P=Productions of the form $A \rightarrow \alpha$, where α=string of terminals and non terminals
S=Start Symbol of grammar G

5. Define derivations. Give an example and its types.

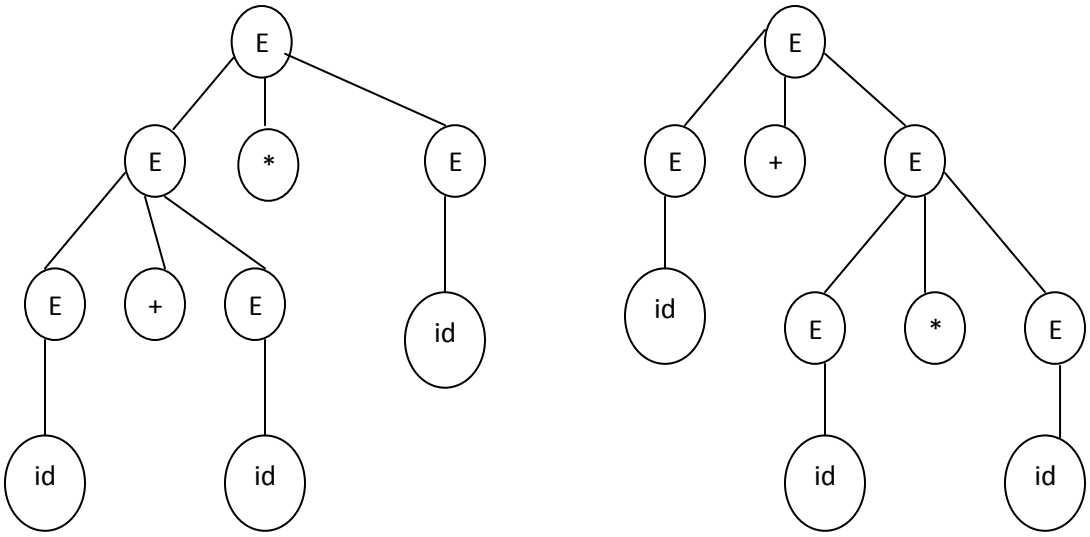
We apply the productions of a CFG to infer that certain strings are in the language of a certain variable. There are two approaches (a) derivations (b)

recursive inference or reduction. Then derivation uses the production from head to body. A replacement according to a production is known as derivation.
i) Left most derivation
ii) Right most derivation or canonical derivations
E→ E+E→id+E→id+(E)
→ id+(E+E)→id+(id*E)
→ id+(id*id)

6. Define parse tree.
The graphical representation of the derivation is called parse tree.

- The root of the parse tree is labeled with starting symbol
- Each interior node will be the non terminal of grammar G
- Each leaf is labeled with terminal or
- The leaves of the parse tree form sentence of G called yield of parse tree

7.Define ambiguity or ambiguous grammar.
A grammar that produces more than one leftmost most derivation or more than one rightmost most derivation for some sentences is said to be ambiguous.
Eg: A→ E+E | E*E | (E) |id
For the Input string id + id * id, more than one parse tree is



8. Define sentential form.
If G = (V, T, P, S) is a CFG, then any string α in (VUT)* such that S→* α is a sentential form.

9. Define yield of the string.
A string that is derived from the root variable is called the yield of the tree.

10.Give the several reasons for writing a grammar.
a) The lexical rules of a language are frequently quite simple and to describe them we do not need a notation as powerful as grammars.
b) R.E generally provides a more concise and easier to understand notation for token than grammars.
c) More efficient lexical analyzers can be constructed automatically from R.E than from arbitrary grammars.
d) Separating the syntactic structure of a language into lexical and nonlexical parts provides a convenient way of modularizing the front end of a compiler into two manageable-sized components.

11.Define left factoring.
The process of factoring out of common prefixes of alternates is called as left factoring. It is a grammar production transformation that is useful for producing grammar suitable for predictive parsing.
If the production is of the form $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
Then after eliminating left factoring, productions are
$A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2$

12.What are the difficulties with top down parsing?
a) Left recursion
b) Backtracking
c) The order in which alternates are tried can affect the language accepted
d) When failure is reported we have very little idea where the error actually occurred.

14.Define top down parsing.
It can be viewed as an attempt to find the left most derivation for an input string. It can be viewed as attempting to construct a parse tree for the input starting from the root and creating the nodes of the parse tree in preorder.

13.What is meant by recursive-descent parser?
A parser that uses a set of recursive procedures to recognize its input with no backtracking is called a recursive-descent parser. To avoid the necessity of a recursive language, we shall also consider a tabular implementation of recursive descent called predictive parsing.

14. What is a predictive parser?

A predictive parser is an efficient way of implementing recursive-descent parsing by handing the stack of activation records explicitly. For LL (1) – the first L means the input is scanned from left-to-right. The second L means it uses leftmost derivation for input string. The number 1 in the input symbol means it uses only one input symbol(look ahead) to predict the parsing process.

15.Define left recursion. Give an example.

A grammar is left recursive if it has a nonterminal A such that there is a derivation A→Aα for some strings α.

Example: Pair of productions A→Aα | β is

Replaced by productionrules

A→βA' and A'→ αA' |€

16.Eliminate left recursion from the grammar.

S→ Aa |b, A→Ac |Sd |€

Replaced as A→bdA' | A'

A' →cA' |adA' |€

Finally we obtain

S→Aa | b

A→bdA' | A'

A' →cA' |adA' | €

17.Define LL (1) grammar.

A grammar G is LL (1) if and only if, whenever A→ α | β are two distinct productions of G of the following conditions

a) For no terminal **a** do both α and β derive strings beginning with α.

b) At most one of α and β can derive the empty string.

c) If β*→ € then α does not derive any string beginning with a terminal in FOLLOW (A).

18.What are the possibilities of non-recursive predictive parsing?

a) If X = a = $ the parser halts and announces successful completion of parsing

b) If X = a = $ the parser pops X off the stack and advances the input pointer to the next symbol

c) If X is a nonterminal, the program consults entry M[X,a] of the parsing table M. this entry will be either an X-production of the grammar or an error entry.

19. Define bottom up parsing.

It attempts to construct a parse tree for an input string, 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 is also known as shift reduce parsing.

20.What are the three techniques for constructing LR parsing table?

a) SLR (simple LR)
b) Canonical LR
c) LALR (Look Ahead LR)

21.What are the actions available in shift reduce parser?

- **Shift** - The next input symbol is shifted onto the top of the stack.
- **Reduce**- It must locate the left end of the handle within the stack and decide with non terminals to replace the handle.
- **Accept**- The parser announces successful completion of parsing.
- **Error**-The parser discovers that a syntax error has occurred and calls an error recovery routine.

22.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 right most derivation.

23. Define handle pruning

A rightmost derivation in reverse is called handle pruning.Reducing β to A in αβw is called handle pruning, i.e., removing the children of A from the parse tree.

If S➔*(rm) αAw ➔(rm) αβw, A→β in the position following α is a handle of αβw

24.Define 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.

25.What are the two common ways of determining precedence relations should hold between a pair of terminals?

a) Based on associative and precedence of operators.

b) Construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse tree.

26. Define LR parser.

LR parsers can be used to parse a large class of context free grammars. The technique is called LR (K) parsing.
"L" denotes that input sequence is processed from left to right
"R" denotes that the right most derivation is performed
"K" denotes that at most K symbols of the sequence are used to make adecision.

27.What are the drawbacks of LR parser?

a) Parsing tables are too complicated to be generated by hand, need an automated parser generator.
b) Cannot handle ambiguous grammar without special tricks.

28.Give the reasons for using LR parser.

a) LR parsers can handle a large class of CF languages
b) An LR parser can detect syntax errors as soon as they occur
c) The LR parsing method is the most general non-back tracking shift reduce parsing method
d) LR parsers can handle all language recognizable by LL(1).

29.What are the techniques for producing LR parsing Table?

1) Shift s, where s is a state
2) Reduce by a grammar production A→ β
3) Accept and
4) Error

30.What are the two functions of LR parsing algorithm?

a) Action function
b) GOTO function

31.What are two classes of items in LR parser?

a) Kernel items, which include the initial item, S' →.S, and all items whose dots are not at the left end.
b) Non-Kernel items, which have their dots at the left end.

32.Define augmented grammar .

If G is a grammar with start symbol S, then G' the augmented grammar for G, is G with a new start symbol S' and production S' →S.The purpose of this new starting production is to indicate to the parser when it should stop parsing and announce acceptance of the input.

33.Define LR (0) items.

LR (0) item for 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.

34.Define SLR parser.

The parsing table consisting of the parsing action and goto function determined by constructing an SLR parsing table algorithm is called SLR(1) table. An LR parser using the SLR (1) table is called SLR (1) parser. A grammar having an SLR (1) parsing table is called SLR (1) grammar.

35. Define LALR grammar

This method is often used in practice because the tables obtained by it are considerably smaller than the canonical LR tables, yet most common syntactic constructs of programming language can be expressed conveniently by an LALR grammar. If there are no parsing action conflicts, then the given grammar is said to be an LALR (1) grammar. The collection of sets of items constructed is called LALR (1) collections.

36.Define YACC tool.

Yet Another Compiler Compiler is a LALR parser generator .It is available as a command in UNIX system.
Unix system command **yacc filename .y** convert yacc program to C program y.tab.c.
By compiling y.tab.c along with library ly, using command **cc y.tab.c –ly**,it create **a.out** that performs translation specified in Yacc program.

UNIT IV

SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT

1. 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.

2. Define Syntax Directed Definition.

A Syntax Directed Definition uses a context free grammar to specify the syntactic structure of the input. 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. The grammar and the set of semantic rules constitute the syntax-directed definition.

3. 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.,

4. Define Synthesized attributes.

An attribute is called synthesized attributes if its value at a parse-tree node is determined from attribute values at the children of the node. They have the property that they can be evaluated during a single bottom up traversal of the parse tree.

5. Define annotated parse tree.

A parse tree showing the attribute values at each node is called annotated parse tree. Suppose n is a node in the parse tree labeled by a grammar symbol X, then X.a denote the value of attribute a of X at that node. The value of X.a at n is computed using the semantic rule for attribute a associated with the X production used at node n.

6. Define S-attributed definition.

Syntax directed definition that uses synthesized attributes exclusively is said to be an S-attributed definition. A parse tree for an S-attributed definition can always be annotated by evaluating the semantic rules for the attributes at each node bottom up, from leaves to root.

7. Define inherited attributes.

An inherited attribute is one whose value at a node in a parse tree is defined in terms of attributes at the parser and/or siblings of that node.Inherited attributes are convenient for expressing dependence of a programming language construct on the context in which it appears. Eg: to keep track of if an identifier appears on left or right side of an assignment statement statement in order to decide if the address or value of an identifier is needed.

8. What is dependency graph?

If an attribute b at a node in a parse tree depends on an attribute c, the semantic rule for b at that node must be evaluated after the semantic rule that defines c. The interdependencies among inherited and synthesized attributes at the nodes in a parse tree can be depicted by a directed graph called dependency graph.

9. What is topological sort?

A topological sort of a directed acyclic graph is any ordering m1,m2,…mk of the nodes of the graph such that edges go from nodes earlier in the ordering to later nodes, that is if mi→mj is an edge from m to mj, then m I appears before mj in the ordering. It gives the valid order in which the semantic rules associated with the nodes in a parse tree can be evaluated.

10.Write some methods for evaluating semantic rules.

- Parse tree methods
- Rule based methods
- Oblivious methods

11. Differentiate static checking.

Compiler must check that the source program follows the syntactic and semantic conventions of the source language .This checking is called static checking. It includes type checks, flow of control checks, uniqueness checks and name related checks.

12.Define type checker.

The type checker is a translation scheme that synthesizes the type of each expression from the types of its subexpressions. It can handle arrays pointers, statements, and functions. A type checker verifies that the type of a construct matches that expected by its context. Information gathered by a type checker is needed for code generation.

13.Define type systems.

A type system is a collection of rules for assigning type expressions to the various parts of a program. A type checker implements the type system.

14.Differentiate static and dynamic checking of types.

Checking done by a compiler is said to be static, while checking done when the target program runs is termed dynamic.

15.Define activation trees.

An activation tree is used to depict the way control enters and leaves activations.
In an activation tree,

- Each node represents an activation of a procedure
- Root represents activation of main program
- Node for a is parent of node for b if and only if control flows from activation a to b
- The node for a is to the left of the node for b if and only if the lifetime of a occurs before the lifetime of b

16.Define control stack.

Control stack is used to keep track of live procedure activations. Idea is to push the node for activation onto control stack as activation begins and to pop the node when the activation ends.

17.Define binding of names.

When an environment associates storage location s with a name x, it is called x is bound to s.

18.What are the fields in an activation record?

The information needed by a single execution of a procedure is managed using a contiguous block of storage called activation record. It consists of collection fields.

Returned value	→Used by the called procedure to return a value to the called procedure
Actual parameters	→Used by the calling procedure to supply parameters to the called procedure
Optional control link	→Points to the activation record to the caller
Optional access link	→Enter the non local data.
Saved machine state	→Holds the information about the state of machine
Local data	→Holds the data that is local to an extension of a procedure
Temporaries	→Evaluation of expressions is stored

19.Give the standard storage allocation strategies.

- Static allocation
- Stack allocation.

20.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.

21.What is dangling references?

Whenever storage can be deallocated , the problem of dangling references arises. A dangling reference occurs when there is a reference to storage that has been deallocated. It is a logical error to use dangling references, since value of deallocated storage is undefined according to the semantics of most languages.

UNIT V

CODE OPTIMIZATION ANDCODE GENERATION

1. 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.

2. 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.

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

- Structure preservation transformations
- Algebraic transformations.

4. Describe algebraic transformations.

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

X = X+ 0

X = X * 1

5. Define DAG. Give an example.

DAG is a directed acyclic graph for an expression identifies the common sub expression in the expression.

Example: DAG for the expression a- 4+c

P1 = mkleaf(id,a) P2 = mknum(num,4)P3 = mknode('-',p1,p2)

P4 = mkleaf(id,c) P5 = mknode('+',p3,p4)

6. 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.

7. 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.

8. 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.

9. Write the characteristics of peephole optimization?

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

10. 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

11. Define Common sub-expression elimination with ex.

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

12. Define Dead-code elimination with ex.
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.

13. Define Renaming of temporary variables with ex.
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.

14. Define reduction in strength with ex.
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 exponentiationroutine.

15. 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.

16. Define 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.

17. Define code generation.

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

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

- Input to the generator
- Target programs
- Memory management
- Instruction selection
- Register allocation
- Choice of evaluation order
- Approaches to code generation.

19.Give the variety of forms in target program.

- Absolute machine language.
- Relocatable machine language.
- Assembly language.

20. Give the factors of instruction selections.

- Uniformity and completeness of the instruction sets
- Instruction speed and machine idioms
- Size of the instruction sets.

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

- During register allocation phase, code generator has to pick the set of variables that will reside in register at a point in the program.
- During a subsequent register assignment phase, code generator has to pick the specific register that a variable reside in.

22. Write the addressing mode and associated costs in the target machine.

MODE	FORM	ADDRESS	ADDED COST
Absolute	M	M	1
Register	R	R	0
Indexed	c(R)	c+contents(R)	1
Indirect register	*R	contents(R)	0
Indirect indexed	*c(R)	contents(c+contents(R))	1

23. 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.

24. 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.

****