Star
Date
Page No.

Unit - 3

#) Semantic Analysis :-

It is third phase of compiler. Semantic Analysis make sure that declarations and statements of program are semantically correct. It is a collection of procedure which is called by parser as and when required by grammer. Both Syntax tree of previous phase and symbol table are used to check the consistency of given code. Type checking is an important part of semantic analysis where Compiler make sure that each operator has matching operands.

#) Semantic Errors

- Type Mismatch
- Undeclared Variable
- Reserved Identifier Misuse
- Multiple declaration of variable in a scope
- Accessing an out of scope Variable
- Actual and formal parameter mismatch.

#) Attribute Grammer

It is the special form of context free grammer where some additional information are appended to one or more of its non terminal in order to provide context sensitive information. Each attribute has well defined domain of values, such as integer, float, character, string and

Star

Date

Page No.

Expression

Attribute Grammer is a medium to provide semantics to the context free Grammer and it can help specify the Syntax & Symantics of a Programming language. Attribute Grammer can pass values or Information among the nodes or tree.

1) **Synthesized Attribute**

An Attribute is Synthesized if its value at a parent node can be determined from attribute of its Children. In General, Given a CFG production rule of the form $A = \alpha B \beta$ then an associated semantic rule of the form "A.attribute = f(α.attribute, B.attribute)" is said to specify a Synthesized Attribute of A.

These are those attributes which derive values from their Children Nodes. i.e. value of Synthesized attribute at node is computed from the values of attribute at Children nodes in parse Tree.

Computation of Synthesized Attribute -

- Write the SDD using appropriate semantic rules for each Grammar.

- The annotated parse tree is generated & attribute values are computed in bottom up manner.
- The value obtained at root node is final output.

ii) Static Allocation

Static Allocation is an allocation procedure which is used for allocation of all the data objects at compile time. In this type of allocation of data objects is done at compile time only. As in static allocation, compiler decides the extent of storage which cannot be changed with time, hence it is easy for the compiler to know the addresses of these data objects in the activation records at later stage. Static allocation is implemented in FORTRAN.
The main function of static allocation is bind data items to a particular memory location.
The static memory allocation procedure consist of determining the size of instruction & data space.

Advantages

- It is easy to implement
- It allows type checking during compilation
- It eliminates the feasibility of running out of memory.

Star
Date
Page No.

Disadvantages

- It is incompatible with recursive subprograms.
- It is not possible to use variables whose size has to be determined at run time.
- The Static Allocation can be done if the size of data object is known as compile time.

Limitations

- The size required must be known at compile time.
- Recursive procedure cannot be implemented as all locals are statically allocated.
- No data structure can be created dynamically as all data is static.

#) Heap Allocation

It is an allocation procedure in which heap is used to manage the allocation of memory. Heap helps in managing the dynamic memory allocation. In Heap allocation, creation of dynamic data object and data structure is also possible as same as Stack allocation. Heap allocation overcomes the limitations of Stack allocation. It is possible to retain the value of variables even after the activation record in heap allocation strategy which is not possible in Stack allocation. It maintains a linked

list for the free blocks and reuse the deallocated space using best fit.

Advantages

- It allows you to access variables Globally.
- Heaps doesn't have any limit on memory size
- It is used in priority Queue
- Garbage Collection

Disadvantages

- It takes more time to compute
- It can provide the maximum memory an OS can provide.
- It take too much time in execution compared to the stack.

#) Stack Allocation

The Stack allocation is a runtime storage management technique. The activation records are pushed and popped as activations begin and end respectively.
Storage for the locals in each call of the procedure is contained in the activation because a new Activation record is pushed onto the stack when the call is made.

Star

Date
Page No.

Advantages
- Not easily corrupted
- Variable cannot be resized
- It helps to manage data

Disadvantages
- Stack Memory is very limited
- Random Access is Not Possible

#) Symbol Table

A symbol table is a major data structure. Associate attributes with identifiers used in program. A symbol table is a necessary declaration of identifiers appears once in a program. Use of identifiers may appear in many places of the program text identifiers and attributes are entered by analysis phases. When processing a definition of an identifier. It stores the information about the scope and binding information about names, information about instances of various entities such as variable & function names, classes, objects etc.

- It is Built-in lexical & syntax analysis phases
- The information is collected by the analysis phases of the compiler and is used by the

Synthesis phase of the compiler to generate code

- It is used by the compiler to achieve compile time efficiency.
- It is used by various phases of the compiler
  - → Lexical Analysis
  - → Syntax Analysis
  - → Semantic Analysis
  - → Intermediate Code Generation
  - → Code optimization
  - → Target Code Generation

Uses of Symbol Table :-

- It is used to verify if a variable has been declared.
- It is used to determine the scope of a name.
- It is used to store the name of all entities in a structured form at one place

Date
Page No.

operations on Symbol Table
The Basic operations defined on a Symbol table include:-
- Allocate - to allocate a new Empty symbol Table
- Free - to remove all entries and free the storage of a Symbol table.
- insert - to insert a name in a symbol table & return a pointer to its entry.
- lookup - to search for a Name and return a pointer to its entry.
- Set_attribute - to associate an attribute with a Given Entry.
- Get_attribute - to Get an attribute associated with a Given Entry.

#) Activation Trees

A program Consist of a sequence/number of instructions Combined into number of procedures. Instructions in a procedure are executed sequentially. A procedure has a start and an end delimiter and everything inside it is called the body of procedure.
The execution of a procedure is called its activation.
When a procedure is executed it returns the Control Back to the caller. This type of control flow makes it easier to represent a

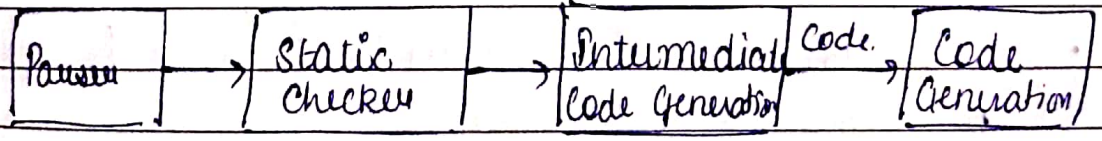
Series of activations in the form a tree known as activation tree

Properties of activation tree :-

- Each node represents an activation of a procedure
- The root represents the activation of the main Program
- The node a is parent of node b if and only if the control flows from a to b
- The node a is left to node b if the lifetime of a occurs before the birthing of b.

Intermediate Code Generation

In this analysis, the front end of a compiler translates a source program into an independent intermediate code, then the Back end of compiler uses this intermediate code to generate the target code. Intermediate Code can translates the source program into machine program It lies between high level language and machine language.



- If the compiler directly translates source code into machine code without Generating

intermediate code - then a full native compiler is required for each new machine.

- Intermediate code generator receives input from its process of phase and semantic analyzer phase. It takes input in the form of an annotated syntax tree.
- Using intermediate code, the second phase of the compiler is changed according to the target machine.
- The intermediate code can be represented in the form of postfix Notation, Syntax tree, directed Acyclic Graph, three address code, Quadruples & triples.
- Intermediate code can be represented into two ways :-
  1. High level Intermediate Code - It can represented as source code.
  2. Low - level Intermediate Code - It is close to the target machine. It is used for machine dependent.

Advantages

- It is Machine Independent. It can be executed on different platform.
- It can perform efficient code generation.
- It creates the function of code optimization easy.

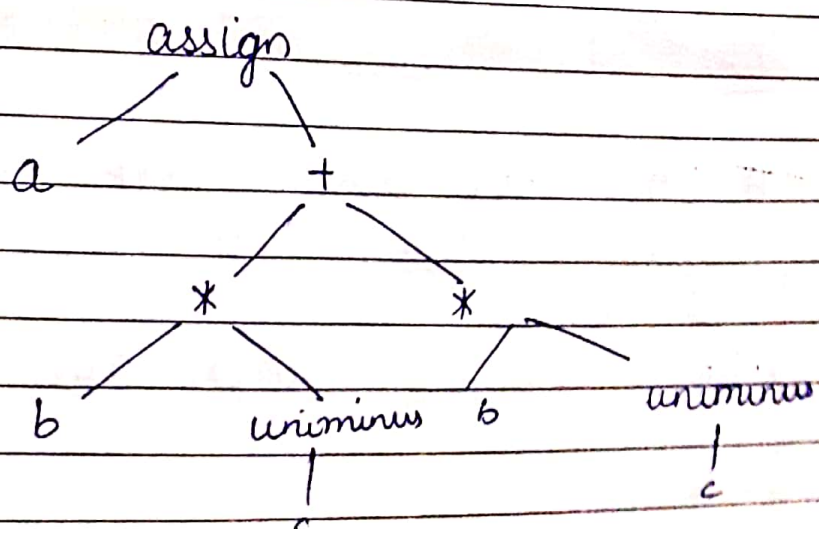
- Syntax-directed translation implements the intermediate code generation, thus by augmenting the parser, it can be folded into the parsing.
- Retargeting is facilitated.

#) Intermediate Code language

It is used to translate the source code into machine code. Intermediate code lies between the high level language and the machine language.

The Intermediate languages can be divided into
→ Syntax Tree
→ Postfix Notation
→ Three Address Code

- Syntax Tree:- A syntax tree depicts the natural hierarchical structure of a source program. A DAG gives the same information but in a more compact way because common sub expressions are identified. A syntax tree for $a := b^* - c + b^*$



Star
Date
Page No.

- **Postfix Notation**

Postfix Notation is linearized representation of a Syntax Tree. It is a list of the Nodes of the tree in which a Node appears immediately after its children. The Postfix Notation representation of

(A / (B - C)* D + E)

is ABC - / D * E +

- **Three Address Code :-**

(two operand, one for result)

A Statement involving three references, is known as three address statement. A Sequence of three address Statement is known as three address Code. Three address statements are a = b op c where a, b, c will have address (memory location). Sometime a statement might contain less than three references but it is still called a three address Statement

#) **DAG Representation for Basic Blocks**

A DAGs for Basic Blocks is a directed acyclic graph with the following - labels on Nodes.

1) The leaves of graph are labelled by

unique identifier and that identifier can be variable name or constant.

8) Interior Nodes of Graph is labelled by an operator symbol.

6) Nodes are also given a sequence of identifiers for labels to store the computed value.

→ DAG are a type of data structure. It is used to implement transformations on basic blocks.

→ DAG provides a good way to determine the common sub expressions.

→ It gives a picture representation of how the value computed by the statement is used in subsequent statements.

Algorithm

Input :- It contains a basic Block

Output :- It contains the following information

→ Each Node contains a label. For leaves is an identifier.

→ Each Node contains a list of attached identifiers to hold the computed values.

Case (i) ⇒ X := y OP z

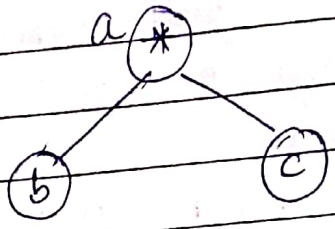
Case (ii) ⇒ X := OP y

Case (iii) ⇒ X := y

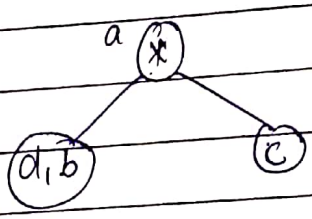
Example

$a = b * c$
$d = b$
$e = d * c$
$b = e$
$f = b + c$
$g = d + f$

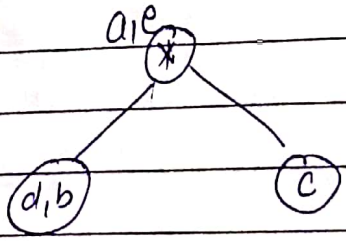
Soln Step1



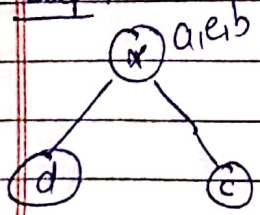
Step2



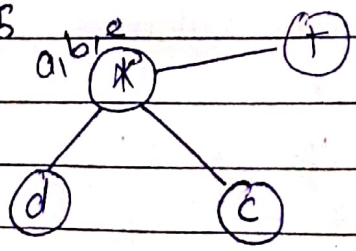
Step3



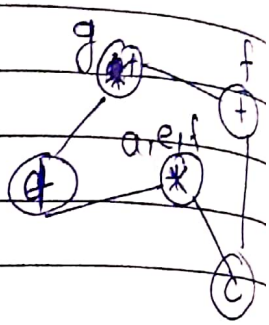
Step4



Step5



Step6



Date
Page No.

Star

Example soln

$(a+b) * (a+b+c)$

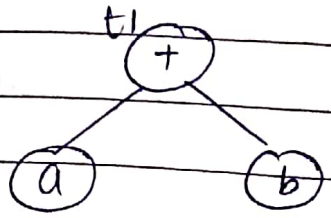
Three Address Code ⇒

$t_1 = a+b$

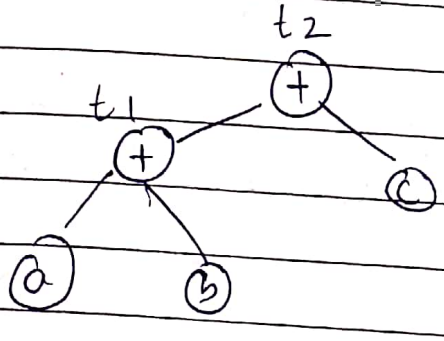
$t_2 = t_1 + b$

$t_3 = t_1 * t_2$

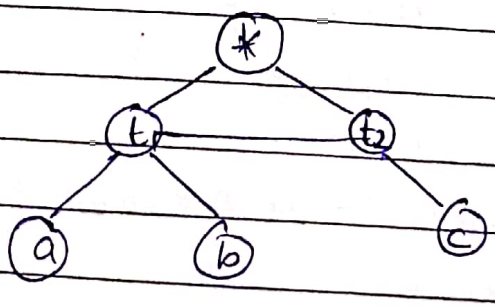
Step1



Step2



Step 2



#) Design Issues In Code Generator

① Input to the code Generator :-

- The Input to Code Generator Contains the intermediate representation of the source program and the information of symbol table. The source program is produced by the front end.
- Intermediate Representation has several choices:
  a) Postfix Notation
  b) Syntax Tree
  c) Three Address Code
- The code Generation phase needs complete error free intermediate code as an input requires

② Target Program :-

The target Program is the output of the code Generator. The output can be :-
a) Assembly language :- It allows subprogram to be separately compiled.
b) Relocate Machine language :- It makes the process of code Generation easier.
c) Absolute Machine Management :- It can be placed in a fixed location in memory and can be executed immediately.

③ Memory Management :-

- During code generation process the symbol table

entries have to be mapped to actual p addresses & levels have to be mapped to instruction addresses.
- Mapping Name in the source program to address of data is CO operating done by the front end & code Generator.
- Local variables are stack allocations in the activation record while Global variable are in static area

④ Instruction Selection :-

- Nature of Instruction set of the target machine should be complete & uniform.
- When you consider the efficiency of the target machine then the instruction speed & machine idioms are important factors.
- The Quality of the Generated Code can be determined by its Speed & Size

⑤ Register Allocation :-

Register can be accessed faster than Memory. The instruction involving operands in register are shorter & faster than involving a Memory operand

Star

Date

Page No.

⑥ Evaluation Order :-

The Efficiency of the target code can be affected by the order in which the computations are performed. Some computations orders need fewer registers to hold results of intermediate than others.

⑦ Approaches to Code Generation Issues :-

Code Generation must always generate the correct code. It is essential because of the Number of Special Cases that a code Generator might face. Some of design Goals are :-
Correct, Easily Maintainable, Testable, Efficient.

#) Basic Blocks and Flow Graphs