St.Joseph's college of engineering
Department of CSE

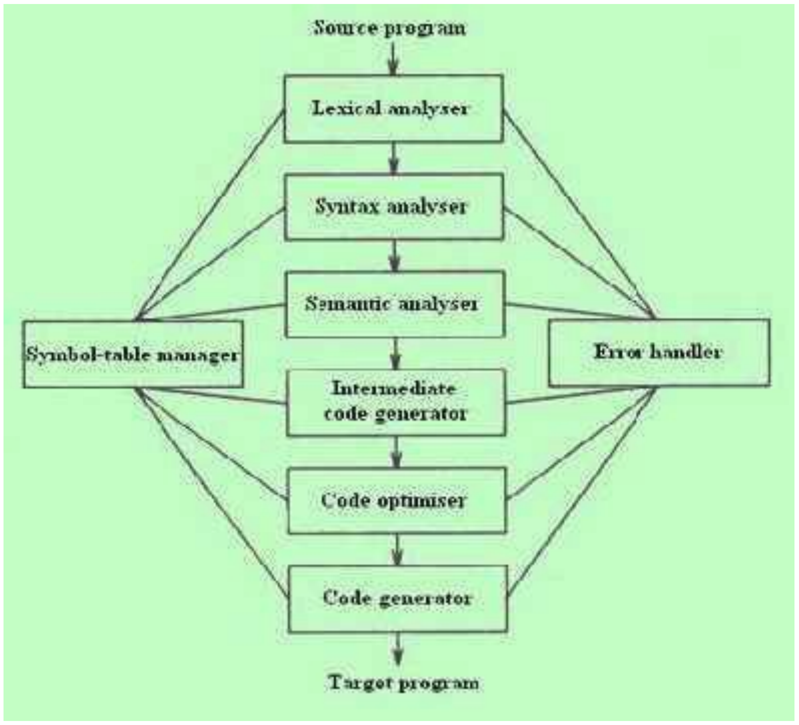
Assignment-I
CS6660-Compiler Design
Part-B

1.Explain the various phases of a compiler in detail. Also write down the output for the following expression after each phase a:= b*c-d.

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

Sub-Phases: 1)Symbol table management 2)Error handling



LEXICAL ANALYSIS:

- It is the first phase of the compiler. It gets input from the source program and produces tokens as output.
- It reads the characters one by one, starting from left to right and forms the tokens.
- **Token** : It represents a logically cohesive sequence of characters such as keywords, operators, identifiers, special symbols etc.
- Example: a +b =20
- Here, a,b,+,=,20 are all separate tokens.
- Group of characters forming a token is called the **Lexeme**.
- The lexical analyser not only generates a token but also enters the lexeme into the symbol table if it is not already there.

SYNTAX ANALYSIS:

- It is the second phase of the compiler. It is also known as parser.
- It gets the token stream as input from the lexical analyser of the compiler and generates syntax tree as the output.

Syntax tree:

- It is a tree in which interior nodes are operators and exterior nodes are operands.
- Example: For a=b+c*2, syntax tree is

SEMANTIC ANALYSIS:

- It is the third phase of the compiler.
- It gets input from the syntax analysis as parse tree and checks whether the given syntax is correct or not.
- It performs type conversion of all the data types into real data types.

INTERMEDIATE CODE GENERATION:

- It is the fourth phase of the compiler.
- It gets input from the semantic analysis and converts the input into output as intermediate code such as three-address code.
- The three -address code consists of a sequence of instructions, each of which has atmost three operands.
  Example: t1=t2+t3

CODE OPTIMIZATION:

- It is the fifth phase of the compiler.
- It gets the intermediate code as input and produces optimized intermediate code as output.
- This phase reduces the redundant code and attempts to improve the intermediate code so that faster-running machine code will result.
- During the code optimization, the result of the program is not affected.
- To improve the code generation, the optimization involves
  - deduction and removal of dead code (unreachable code).
  - calculation of constants in expressions and terms.
  - collapsing of repeated expression into temporary string.
  - loop unrolling.
  - moving code outside the loop.
  - removal of unwanted temporary variables.

CODE GENERATION:

- It is the final phase of the compiler.
- It gets input from code optimization phase and produces the target code or object code as result.
- Intermediate instructions are translated into a sequence of machine instructions that perform the same task.

- The code generation involves
  - allocation of register and memory
  - generation of correct references

- generation of correct data types
- generation of missing code

SYMBOL TABLE MANAGEMENT:

- Symbol table is used to store all the information about identifiers used in the program.
- It is a data structure containing a record for each identifier, with fields for the attributes of the identifier.
- It allows to find the record for each identifier quickly and to store or retrieve data from that record.
- Whenever an identifier is detected in any of the phases, it is stored in the symbol table.

ERROR HANDLING:

- Each phase can encounter errors. After detecting an error, a phase must handle the error so that compilation can proceed.
- In lexical analysis, errors occur in separation of tokens.
- In syntax analysis, errors occur during construction of syntax tree.
- In semantic analysis, errors occur when the compiler detects constructs with right syntactic structure but no meaning and duringtype conversion.
- In code optimization, errors occur when the result is affected by the optimization.
- In code generation, it shows error when code is missing etc

2.Write in detail about the analysis cousins of the compiler.

COUSINS OF COMPILER

1. Preprocessor
2. Assembler
3. Loader and Link-editor

PREPROCESSOR A preprocessor is a program that processes its input data to produce output that is used as input to another program. The output is said to be a preprocessed form of the input data, which is often used by some subsequent programs like compilers. They may perform the following functions :

1. Macro processing
2. File Inclusion
3. Rational Preprocessors
4. Language extension

1. Macro processing:
A macro is a rule or pattern that specifies how a certain input sequence should be mapped to an output sequence according to a defined procedure. The mapping process that instantiates a macro into a specific output sequence is known as macro expansion. **2. File Inclusion:**
Preprocessor includes header files into the program text. When the preprocessor finds an #include directive it replaces it by the entire content of the specified file.
3. Rational Preprocessors:
These processors change older languages with more modern flow-of-control and data-structuring facilities.
4. Language extension :
These processors attempt to add capabilities to the language by what amounts to built-in macros. For example, the language Equel is a database query language embedded in C.
ASSEMBLER

Assembler creates object code by translating assembly instruction mnemonics into machine code. There are two types of assemblers:

- One-pass assemblers go through the source code once and assume that all symbols will be defined before any instruction that references them.
- Two-pass assemblers create a table with all symbols and their values in the first pass, and then use the table in a second pass to generate code.

LINKER AND LOADER A **linker** or **link editor** is a program that takes one or more objects generated by a compiler and combines them into a single executable program. Three tasks of the linker are :

1. Searches the program to find library routines used by program, e.g. printf(), math routines.
2. Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references
3. Resolves references among files.

A loader is the part of an operating system that is responsible for loading programs in memory, one of the essential stages in the process of starting a program.

3.Describe how various phases could be combined as a pass in a compiler? Also briefly explain Compiler construction tools .

Compiler passes A collection of phases is done only once (single pass) or multiple times (multi pass)

- Single pass: usually requires everything to be defined before being used in source program.
- Multi pass: compiler may have to keep entire program representation in memory.

Several phases can be grouped into one single pass and the activities of these phases are interleaved during the pass. For example, lexical analysis, syntax analysis, semantic analysis and intermediate code generation might be grouped into one pass.
C and PASCAL permit one pass compilation Modula-2 requires two passes.

COMPILER CONSTRUCTION TOOLS
These are specialized tools that have been developed for helping implement various phases of a compiler. The following are the compiler construction tools:
1) **Parser Generators:**
-These produce syntax analyzers, normally from input that is based on a context-free grammar.
-It consumes a large fraction of the running time of a compiler. -Example-YACC (Yet Another Compiler-Compiler).
2) **Scanner Generator:**

-These generate lexical analyzers, normally from a specification based on regular expressions. -The basic organization of lexical analyzers is based on finite automation.
3) **Syntax-Directed Translation:**
-These produce routines that walk the parse tree and as a result generate intermediate code. -Each translation is defined in terms of translations at its neighbor nodes in the tree.
4) **Automatic Code Generators:**
-It takes a collection of rules to translate intermediate language into machine language. The rules must include sufficient details to handle different possible access methods for data.
5) **Data-Flow Engines:**

-It does code optimization using data-flow analysis, that is, the gathering of information about how values are transmitted from one part of a program to each other part.

4.(i) What is difference between a phase and pass of a compiler? Explain machine dependent and machine independent phase of compiler.(8)

Difference between a phase and pass of a compiler:

- In complier, the process of compilation can be carried out with the help of varius phases such as lexical analysis , syntax analysis , intermediate code generator, code generator and code optimization .
- Whereas in case of passes, various phases are combined together to form a single pass. An input program can be complied using a single pass or using two passes.
- Different groups of phase form corresponding front end and back ends of the compilers. The advantage of front-end and back-end model of complier is that: same source language can be compiled on different machines or several different language can be complied on the same machine.
- Compiling a source program into single pass is difficult because of forward references that may occur in the program . similarly if we group all the phases into a single pass then we may be forward to keep the entire program in the memory .And then memory requirement may be large . on the other hand it is describe to have relatively few passes, because it becomes time consuming to read and write intermediate file**.**

Front end: analysis (**machine independent**) These normally include lexical and syntactic analysis, the creation of the symbol table, semantic analysis and the generation of intermediate code. It also includes error handling that goes along with each of these phases.

Back end: synthesis (**machine dependent**) It includes code optimization phase and code generation along with the necessary error handling and symbol table operations.

(ii) Describe the following software tools i. Structure Editors ii. Pretty printers iii. Interpreters (8)

Software tools used in Analysis part:
1) **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.
- For example , it can supply key words automatically - while …. do and begin….. end.

2) Pretty printers :

- A pretty printer analyzes a program and prints it in such a way that the structure of the program becomes clearly visible.
- For example, comments may appear in a special font.

3) Static checkers :

- A static checker reads a program, analyzes it, and attempts to discover potential bugs without running the program.
- For example, a static checker may detect that parts of the source program can never be executed.

4) Interpreters :

- Translates from high level language (BASIC, FORTRAN, etc..) into machine language.
- An interpreter might build a syntax tree and then carry out the operations at the nodes as it walks the tree.
- Interpreters are frequently used to execute command language since each operator executed in a command language is usually an invocation of a complex routine such as an editor or complier.