UPES

UNIVERSITY WITH A PURPOSE

Unit 1
Compiler Structure, Phases and Passes
Bootstrapping

UPES
UNIVERSITY WITH A PURPOSE

Need of Compiler

- Computer can understand only machine language. For human being, its very tough to interact with computer in machine language.
- Human interact with computer in programming language which are very much similar to English like language.
- So a translator is required which will convert this high level instructions into machine language.

Prerequisite

- High Level Language: Machine independent similar to English like language which are understandable to user but computer can’t understand. E.g. Programing language like C, C++, Java, Pascal etc.
- Machine Language: It is a low level machine dependent binary code which a computer can understand and execute directly. For human being its very tough to understand.
- Assembly Language: It is a low level machine dependent language where a group of binary number is given a symbol. These symbols are called MNEMONIC which are easy to remember.

Question 1

Pascal is which type of language

A. Machine Language
B. Assembly Language
C. High Level Language
D. None of the above

UPES
UNIVERSITY WITH A PURPOSE

Question 1

Pascal is which type of language

A. Machine Language
B. Assembly Language
C. High Level Language
D. None of the above

Option C is correct

UPES
UNIVERSITY WITH A PURPOSE

Question 2

Assembly language is machine independent

- True
- False

Question 2

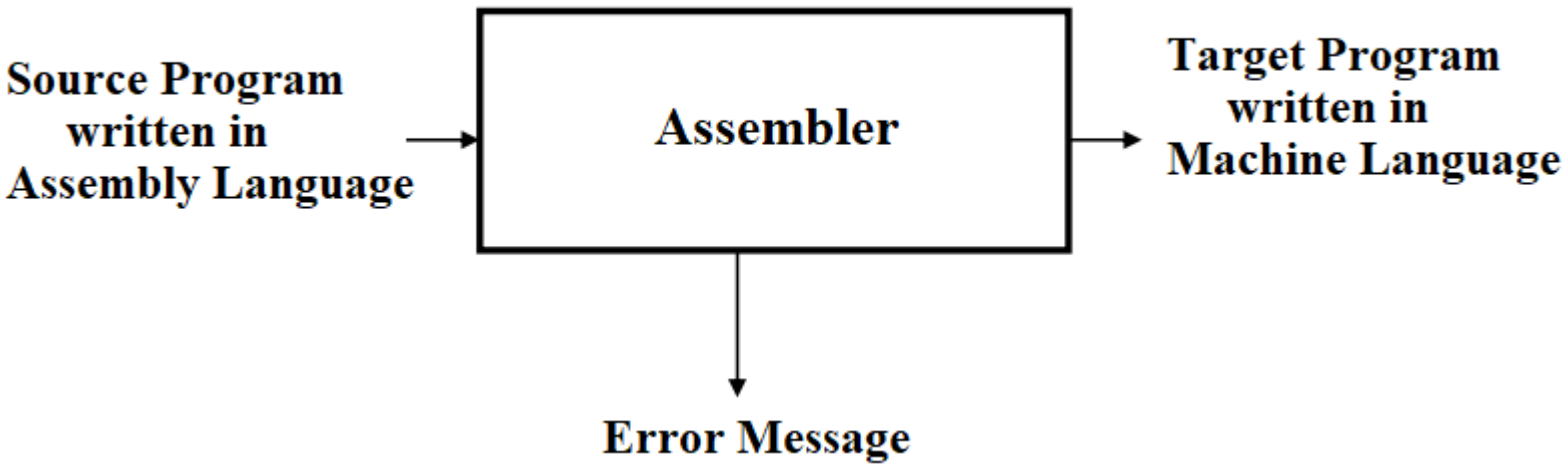
Assembly language are machine independent

- True
- False

- Its false as assembly language are machine dependent

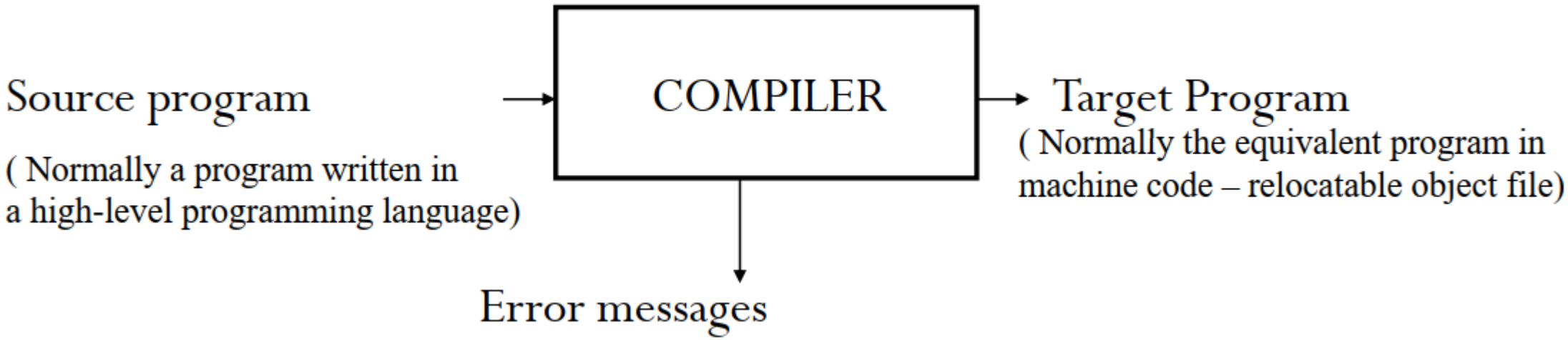
Assembler

- Assembler is a program that takes as input program written in Assembly language and translates that into equivalent program written in Machine language.



Compiler

- Compiler is a program that takes as input program written in source language (high level programing language) and translates that into equivalent program written in target language (machine code)



UPES
UNIVERSITY WITH A PURPOSE

Interpreter

- An interpreter, like a compiler, translates high-level language into low-level machine language. The difference lies in the way they read the source code or input.
- A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes.
- In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence.

Interpreter

- If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.
- Interpreter are smaller in size relative to compiler.
- Execution time of an interpreted program is more than that of corresponding compiled program.

Pre-processor

- A preprocessor, generally considered as a part of compiler, is a tool that produces input for compilers.
- It deals with macro-processing, file inclusion, comments removal etc.

Linker & Loader

- Linker is a program that links and merges various object files together in order to make an executable file. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded making the program instruction to have absolute reference.
- Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and create memory space for it.

Question 3

- Output of compiler is _____

Question 3

- Output of compiler is machine code.

Question 3

- Input of assembler is _____

Question 3

- Input of assembler is assembly language program.

Question 3

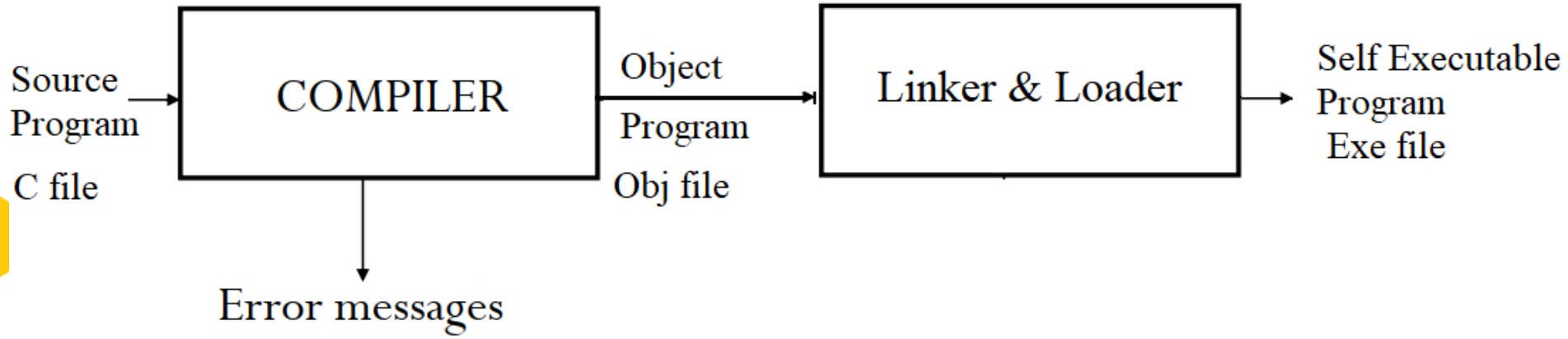
- Output of preprocessor is _____

Question 3

- Output of preprocessor is High level language program.

Compilation Process

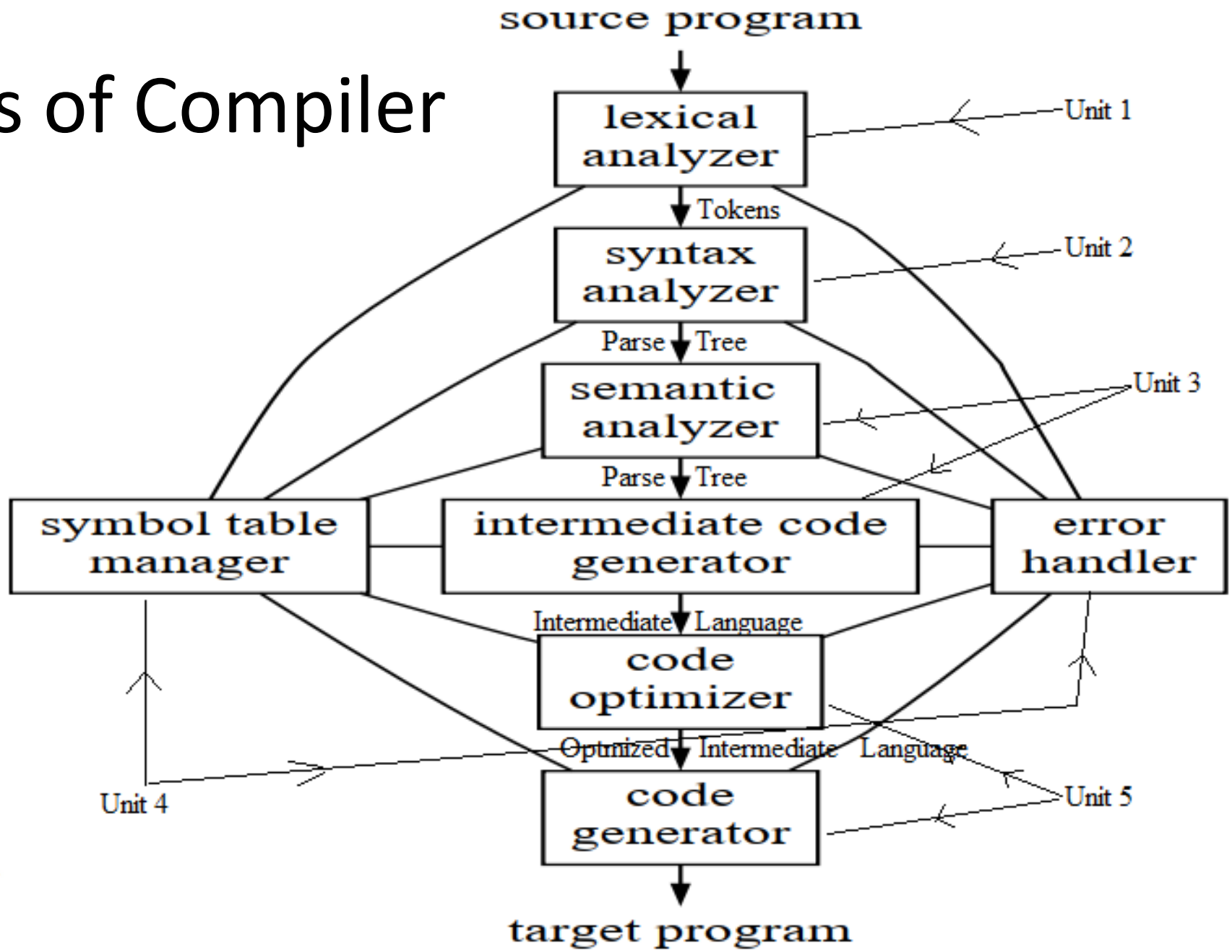
- When a program is compiled it is converted into an intermediate code called object code. This object code is a machine code without memory address location.
- Compiler just remember the name of external functions given by the user and code for build in function is added.
- Assuming that external functions code is there in library, linker combines the object code of these external functions with program to generate the executable files.
- Loader loads all the executable files in the memory and execute them.



Phases of Compiler

- Compilation process is very complex, so it is not reasonable to consider the compilation process as occurring in one single step.
- It is good to partition the compilation process into a series of sub processes called phases.

Phases of Compiler



Phases of Compiler

- Lexical Analyzer
  - Read the source program character by character
  - Returns the tokens of the source program
  - Tokens are group of characters that logically belong together.
  - Token can be keyword, operator symbols, punctuation symbols, identifiers, constant etc.
  - Every token has two parts: (1) Token type (2) Token Value
  - Some tokens has only type while some has both type and value.
  - Information related to every token is stored in symbol table through a book keeping routine.

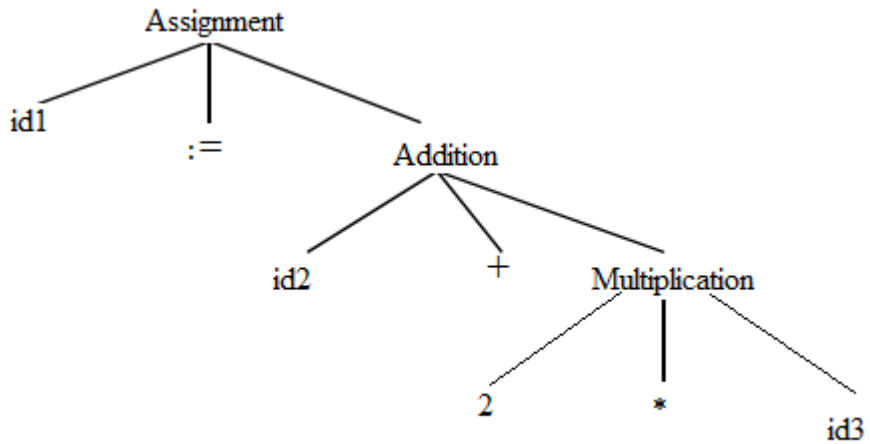
Phases of Compiler

- Lexical Analyzer
  - Whenever parser calls lexical analyzer for a token, then Lexical analyzer returns a code corresponding to that token to parser. If token also possess value then value is also returned to parser.
  - Regular expressions are used to describe tokens.
  - Finite state automaton can be used for implementation of lexical analyzer.
  - Let x=y+2z is an expression and we want to generate tokens for this
  - id1 assignment operator id2 addition operator constant 2 multiplication operator id3
  - Here id1, id2 and id3 are representing identifier

UPES
UNIVERSITY WITH A PURPOSE

Phases of Compiler

- Syntax Analyzer (Parser)
  - Syntax of the language are specified by context free grammar. Syntax analyzer checks that token appearing in its i/p are according to rules specified in context free grammar of the language or not.
  - It groups token together into a syntactic structure called parse tree. Leaf nodes in the parse tree are tokens while internal nodes represents the string of tokens that logically belong together.
  - x=y+2z



Phases of Compiler

- Semantic Analyzer
  - This phase is used for type checking of the expressions.
  - It can’t be represented by context free language used during syntax analyzer.
  - Semantic checking can be implemented independently as a separate phase, but generally it is implemented as a part of syntax analyzer, intermediate code generator or code generation phase.
  - For expression x=y+2z we can do temp1=int to real(2) if id3 is real

Question 4

Outcome of lexical analyzer phase is

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

UPES
UNIVERSITY WITH A PURPOSE

Question 4

Outcome of lexical analyzer phase is

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

Option A is correct.

UPES
UNIVERSITY WITH A PURPOSE

Question 5

Outcome of syntax analyzer phase is

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

Question 5

Outcome of syntax analyzer phase is

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

Option B is correct.

Question 6

Regular expressions are used to describe

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

UPES
UNIVERSITY WITH A PURPOSE

Question 6

Regular expressions are used to describe

A. Tokens
B. Parse Tree
C. Object Code
D. Intermediate Code

Option A is correct.

Question 7

Finite state automaton is used for implementation of

A. Push Down Automata
B. Turing Machine
C. Lexical Analyzer
D. All of the above

Question 7

Finite state automaton is used for implementation of

A. Push Down Automata
B. Turing Machine
C. Lexical Analyzer
D. All of the above

Option C is correct

UPES
UNIVERSITY WITH A PURPOSE

Question 8

Context free grammar is used to specify

A. Token of language
B. Syntax of the language
C. Semantic of the language
D. All of the above

Question 8

Context free grammar is used to specify

A. Token of language
B. Syntax of the language
C. Semantic of the language
D. All of the above

Option B is correct

Symbol Table

- In the first two phases, compiler collects information about the data objects appearing in the source program.
- Symbol table is a data structure which stores all the information related to data objects found in first two phases. It stores their name, type, value, scope and other related information.
- This module interact with all phases of compiler.
- First two phases stores information in the symbol table while last three phases fetch information from the symbol table.

Error Handler

- On occurrence of an error, compiler phase calls the error handler module.
- Error handler module checks the type of error and accordingly display the error message. This will help programmer to detect and locate the error.
- This module also interact with all phases of compiler.

Intermediate Code Generation

- Input to this phase is parse tree and output is intermediate language representation of source program which is machine independent.
- The primary difference between intermediate code and assembly code is that the intermediate code need not specify the registers to be used for each operation.
- For the expression x=y+2z
  - temp1= int to real(2)
  - temp2=id3*temp1
  - temp3=id2+temp2
  - id1=temp3

UPES
UNIVERSITY WITH A PURPOSE

Code Optimization

- This phase is used to improve the intermediate code so that the ultimate object program runs faster and takes less space.
- Its output is another intermediate code program that does the same job as the original but perhaps in a way that saves time and space.
- For the expression x=y+2z
  - temp1=id3*int to real(2)
  - Id1=id2+temp1

Code Generation

- Input to this phase is the intermediate code and output is machine dependent target program which is relocatable object file.
- For the expression x=y+2z
  - Mov R1 id3 Move the content of id3 in register R1
  - Mul R1 #2.0 Multiply the content stored in R1 with 2.0
  - Mov R2 id2 Move the content of id2 in register R2
  - Add R2 R1 Add the content of R1 in R2
  - Mov id1 R2 Move the content of R2 in id1

Question 9

Which module can interact with all phases:

A. Lexical Analyzer
B. Syntax Analyzer
C. Symbol Table
D. Code Generation

Question 9

Which module can interact with all phases:

A. Lexical Analyzer
B. Syntax Analyzer
C. Symbol Table
D. Code Generation

Option C is correct

Passes of Compiler

- Portion of one or more phase are combined into a group called pass.
- A pass read the source program or the o/p of the previous pass, makes the transformation specified by its phases and writes o/p into an intermediate file which may then be read by subsequent pass.
- The number of passes and grouping of phases into passes are usually depend on a particular language and machine.
- A multi pass compiler can be made to use less space than a single pass compiler since the space occupied by compiler program for one pass can be reused by the other pass. But it will be slower than single pass compiler because each pass read and writes an intermediate file.

Passes of Compiler

- Usually first three phases are part of one group called Analysis pass and last three phases are part of another group called Synthesis pass.
- Analysis pass (Front end) breaks the source program into constituent pieces and creates and intermediate representation of source program. This pass is machine independent. It depends only on the source program language.
- Synthesis pass (back end) constructs the desired target program from the intermediate representation. This pass is machine dependent. It does not depend on the language of source program.

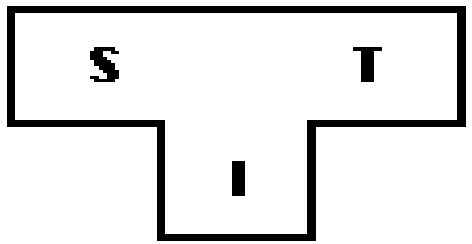
Cross Compiler

- A compiler which is capable of creating executable code for a platform other than one on which compiler is running is called cross compiler.
- For example, a compiler that runs on Windows platform and also generates a code that runs on android platform is a cross compiler.
- The process of creating executable code for a different platform is also called **retargeting**. Therefore, the cross compiler is also known as a **Retargetable compiler**.
- Keil is an example for cross compiler.

Bootstrapping

- A compiler can be characterized by three languages:
  - Source Language (language in which input program will be written)
  - Target Language (language in which output program will be produced)
  - Implementation Language (Language in which compiler is written)

- The T- diagram shows a compiler ${}^{S}C_{I}^{T}$ for Source S, Target T, implemented in I

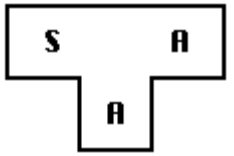


UPES
UNIVERSITY WITH A PURPOSE

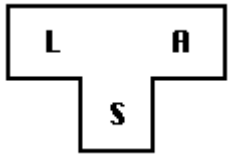
Bootstrapping

- **Bootstrapping** is the technique for producing a self-compiling **compiler** i.e. how to compile the first program in a new language. Following are the steps for bootstrapping for language L:

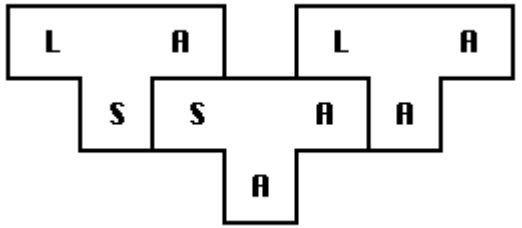
1. Create a compiler ${}^{S}C_{A}{}^{A}$, where S is the source language and A is the target as well as implementation language. S is subset of language L.



2. Create a compiler ${}^{L}C_{S}{}^{A}$, where L is the source language, A is the target language and S is the implementation language.



3. Compile ${}^{L}C_{S}{}^{A}$ using the compiler ${}^{S}C_{A}{}^{A}$ to obtain ${}^{L}C_{A}{}^{A}$. ${}^{L}C_{A}{}^{A}$ is a compiler for language L, which runs on machine A and produces code for machine A.



$$ {}^{L}C_{S}^{A} \rightarrow {}^{S}C_{A}^{A} \rightarrow {}^{L}C_{A}^{A} $$

THANK YOU

UPES
UNIVERSITY WITH A PURPOSE

Dr Anurag Jain
Assistant Professor (SG), Virtualization Department, School of Computer Science,
University of Petroleum & Energy Studies, Dehradun - 248 007 (Uttarakhand)
Email: anurag.jain@ddn.upes.ac.in , dr.anuragjain14@gmail.com
Mobile: (+91) -9729371188