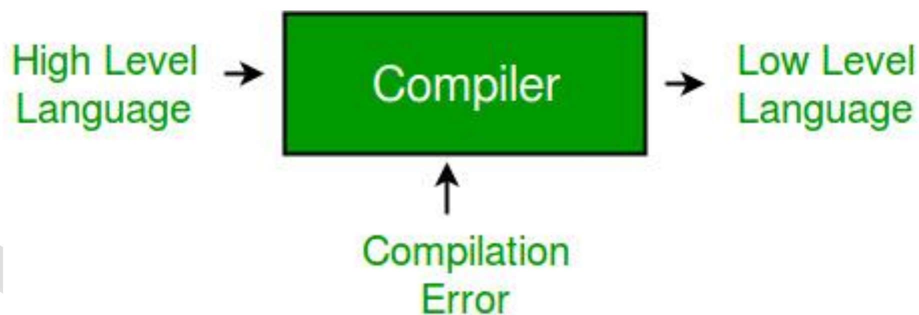
<u>Compiler Design by Md. Ajij</u>
<u>(5th Semester)</u>

1) What is a compiler?

A *compiler* is a program that translates a source program written in some high-level programming language (such as Java) into machine code for some computer architecture (such as the Intel Pentium architecture). The generated machine code can be later executed many times against different data each time.



Simply stated, a compiler is a program that reads a program written in one language-the source language-and translates it into an equivalent program in another language-the target language. The compiler reports to its user the presence of errors in the source program

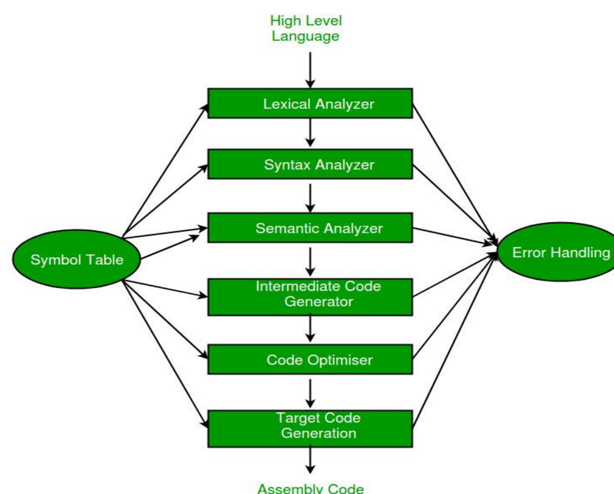
SRSVIM

2) Define cross compiler.

Cross Compilers are compilers that execute on one computer and generate object code that can execute on different platform.

For example a cross compiler that is running on windows pc can produce object code that run on MAC OS or Android OS.

3) What are the various phases of the compiler? Explain each phase in detail.



Phase 1: Lexical Analysis

Lexical Analysis is the first phase when compiler scans the source code. This process can be left to right, character by character, and group these characters into tokens.

Here, the character stream from the source program is grouped in meaningful sequences by identifying the tokens. It makes the entry of the corresponding tickets into the symbol table and passes that token to next phase.

The primary functions of this phase are:

- Identify the lexical units in a source code
- Classify lexical units into classes like constants, reserved words, and enter them in different tables. It will Ignore comments in the source program
- Identify token which is not a part of the language

Example: x = y + 10

Tokens

X	identifier
=	Assignment operator
Y	identifier
+	Addition operator
10	Number

Phase 2: Syntax Analysis

Syntax analysis is all about discovering structure in code. It determines whether or not a text follows the expected format. The main aim of this phase

is to make sure that the source code was written by the programmer is correct or not.

Syntax analysis is based on the rules based on the specific programing language by constructing the parse tree with the help of tokens. It also determines the structure of source language and grammar or syntax of the language.

Here, is a list of tasks performed in this phase:

- Obtain tokens from the lexical analyzer
- Checks if the expression is syntactically correct or not
- Report all syntax errors
- Construct a hierarchical structure which is known as a parse tree

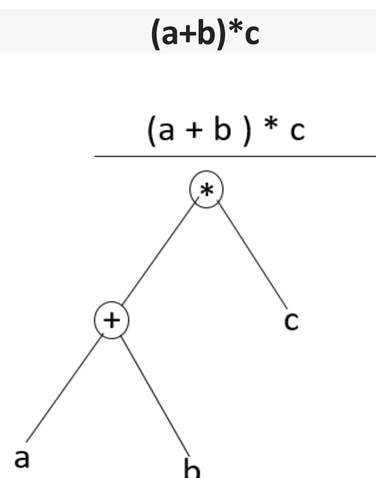
Example

Any identifier/number is an expression

If x is an identifier and y+10 is an expression, then x= y+10 is a statement.

Consider parse tree for the following example

(a+b)*c



In Parse Tree

- Interior node: record with an operator filed and two files for children
- Leaf: records with 2/more fields; one for token and other information about the token
- Ensure that the components of the program fit together meaningfully
- Gathers type information and checks for type compatibility

- Checks operands are permitted by the source language

Phase 3: Semantic Analysis

Semantic analysis checks the semantic consistency of the code. It uses the syntax tree of the previous phase along with the symbol table to verify that the given source code is semantically consistent. It also checks whether the code is conveying an appropriate meaning.

Semantic Analyzer will check for Type mismatches, incompatible operands, a function called with improper arguments, an undeclared variable, etc.

Functions of Semantic analyses phase are:

- Helps you to store type information gathered and save it in symbol table or syntax tree
- Allows you to perform type checking
- In the case of type mismatch, where there are no exact type correction rules which satisfy the desired operation a semantic error is shown
- Collects type information and checks for type compatibility
- Checks if the source language permits the operands or not

Example

```
float x = 20.2;
float y = x*30;
```

In the above code, the semantic analyzer will typecast the integer 30 to float 30.0 before multiplication.

Phase 4: Intermediate Code Generation

Once the semantic analysis phase is over the compiler, generates intermediate code for the target machine. It represents a program for some abstract machine.

Intermediate code is between the high-level and machine level language. This intermediate code needs to be generated in such a manner that makes it easy to translate it into the target machine code.

Functions on Intermediate Code generation:

- It should be generated from the semantic representation of the source program
- Holds the values computed during the process of translation
- Helps you to translate the intermediate code into target language
- Allows you to maintain precedence ordering of the source language
- It holds the correct number of operands of the instruction

Example

For example,

```
total = count + rate * 5
```

Intermediate code with the help of address code method is:

```
t1 := int_to_float(5)
t2 := rate * t1
t3 := count + t2
total := t3
```

SRSVM

Phase 5: Code Optimization

The next phase of is code optimization or Intermediate code. This phase removes unnecessary code line and arranges the sequence of statements to speed up the execution of the program without wasting resources. The main goal of this phase is to improve on the intermediate code to generate a code that runs faster and occupies less space.

The primary functions of this phase are:

- It helps you to establish a trade-off between execution and compilation speed
- Improves the running time of the target program
- Generates streamlined code still in intermediate representation
- Removing unreachable code and getting rid of unused variables
- Removing statements which are not altered from the loop

Example:

Consider the following code

```
a = intofloat(10)
b = c * a
d = e + b
f = d
```

Can become

```
b =c * 10.0
f = e+b
```

Phase 6: Code Generation

Code generation is the last and final phase of a compiler. It gets inputs from code optimization phases and produces the page code or object code as a result. The objective of this phase is to allocate storage and generate relocatable machine code.

It also allocates memory locations for the variable. The instructions in the intermediate code are converted into machine instructions. This phase coverts the optimize or intermediate code into the target language.

The target language is the machine code. Therefore, all the memory locations and registers are also selected and allotted during this phase. The code generated by this phase is executed to take inputs and generate expected outputs.

Example:

a = b + 60.0

Would be possibly translated to registers.

```
MOVF a, R1
MULF #60.0, R2
ADDF R1, R2
```

Symbol Table Management

A symbol table contains a record for each identifier with fields for the attributes of the identifier. This component makes it easier for the compiler to search the identifier record and retrieve it quickly. The symbol table also helps you for the scope management. The symbol table and error handler interact with all the phases and symbol table update correspondingly.

Error Handling Routine:

In the compiler design process error may occur in all the below-given phases:

- Lexical analyzer: Wrongly spelled tokens
- Syntax analyzer: Missing parenthesis
- Intermediate code generator: Mismatched operands for an operator
- Code Optimizer: When the statement is not reachable
- Code Generator: Unreachable statements
- Symbol tables: Error of multiple declared identifiers

Most common errors are invalid character sequence in scanning, invalid token sequences in type, scope error, and parsing in semantic analysis.

The error may be encountered in any of the above phases. After finding errors, the phase needs to deal with the errors to continue with the compilation process. These errors need to be reported to the error handler which handles the error to perform the compilation process. Generally, the errors are reported in the form of message.