Compiler Design

Intro:

- The *compiler* is a software that converts a program written in a high-level language (Source Language) to a low-level language (Object/Target/Machine Language/0’s, 1’s).

- **What is an Interpreter? 😎Sol:** 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. If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.
- **What is an assembler? 😁Sol:** An assembler translates assembly language programs into machine code. It is an example of a low level to low level language converter.
- **What are HLL and LLL? ♂ Sol: HLL** : Higher level languages are simple languages that use English and mathematical symbols like +,- ,% , /, etc. ‘ for its programme construction.

  LLL : The term low level means closeness to the way in which the machine has been built. Low level languages are machine oriented and require extensive knowledge of computer hardware and its configuration.
- **What is the diff btwn Interpreter and Compiler? ♂**

Interpreter	Compiler
Translates the program one statement at a time.	The compiler scans the whole program in one go.
It does not convert SC to OC instead it scans it line by line	It converts the source code into object code.
Due to interpreters being slow in executing the object code, it is preferred less.	The main advantage of compilers is its execution time.

Phases Of Compiler

- **Symbol Table** – It is a data structure being used and maintained by the compiler, consisting of all the identifier’s names along with their types. It helps the compiler to function smoothly by finding the identifiers quickly.

😜 ***The analysis of a source program is divided into mainly three phases. They are:***

- **Linear Analysis**- This involves a scanning phase where the stream of characters is read from left to right. It is then grouped into various tokens having a collective meaning.
- **Hierarchical Analysis**- In this analysis phase, based on a collective meaning, the tokens are categorized hierarchically into nested groups.
- **Semantic Analysis**- This phase is used to check whether the components of the source program are meaningful or not.

Lexical Analyzer –

It is also called a scanner. It takes the output of the preprocessor (which performs file inclusion and macro expansion) as the input which is in a pure high-level language. It reads the characters from the source program and groups them into lexemes (sequence of characters that “go together”). Each lexeme corresponds to a token. Tokens are defined by regular expressions which are understood by the lexical analyzer. It also removes lexical errors (e.g., erroneous characters), comments, and white space.

Syntax Analyzer –

It is sometimes called a parser. It constructs the parse tree. It takes all the tokens one by one and uses Context-Free Grammar to construct the parse tree.

Semantic Analyzer –

It verifies the parse tree, whether it’s meaningful or not. It furthermore produces a verified parse tree. It also does type checking, Label checking, and Flow control checking. Intermediate Code Generator – It generates intermediate code, which is a form that can be readily executed by a machine We have many popular intermediate codes. Example – Three address codes etc. Intermediate code is converted to machine language using the last two phases which are platform dependent. Till intermediate code, it is the same for every compiler out there, but after that, it depends on the platform. To build a new compiler we don’t need to build it from scratch. We can take the intermediate code from the already existing compiler and build the last two parts.

Code Optimizer –

It transforms the code so that it consumes fewer resources and produces more speed. The meaning of the code being transformed is not altered. Optimization can be categorized into two types: machine-dependent and machine-independent. Target Code Generator – The main purpose of the Target Code generator is to write a code that the machine can understand and also register allocation, instruction selection, etc. The output is dependent on the type of assembler. This is the final stage of compilation. The optimized code is converted into relocatable machine code which then forms the input to the linker and loader.

Acknowledgements –

- GeeksForGeeks
- Gate Smashers - Compiler design
- Follow me if u like 😊

My initial README.md