Final Assessment | Fall - 2023

Md. Shafayet Hossain

CSE - 21st Batch | **Course Code:** CSI - 411

Course Title: Compiler | **ID:** 2121210071

Answer to the Question no- 1

(a)

Types of Compiler

There are four types of compiler-

1. **Cross Compilers:** They produce an executable machine code for a platform but, this platform is not the one on which the compiler is running.
2. **Bootstrap Compilers:** These compilers are written in a programming language that they have to compile.
3. **Source to source/trans-compiler:** These compilers convert the source code of one programming language to the source code of another programming language.
4. **De-compiler:** Basically, it is not a compiler. It is just the reverse of the compiler. It converts the machine code into high-level language.

Phases of Compiler-

The typical phases of a compiler are:

1. **Lexical Analysis:** The first phase of a compiler is lexical analysis, also known as scanning. This phase reads the source code and breaks it into a stream of tokens, which are the basic units of the programming language.

2. **Syntax Analysis:** The second phase of a compiler is syntax analysis, also known as parsing. This phase takes the stream of tokens generated by the lexical analysis phase and checks whether they conform to the grammar of the programming language. The output of this phase is usually an Abstract Syntax Tree (AST).

3. **Semantic Analysis:** This phase checks whether the code is semantically correct, i.e., whether it conforms to the language's type system and other semantic rules. In this stage, the compiler checks the meaning of the source code to ensure that it makes sense.

4. **Intermediate Code Generation:** This phase generates an intermediate representation of the source code that can be easily translated into machine code.

5. **Optimization:** This phase applies various optimization techniques to the intermediate code to improve the performance of the generated machine code.

6. **Code Generation:** This phase takes the optimized intermediate code and generates the actual machine code that can be executed by the target hardware.

1(b)

Advantages of Compiler Design:

There are several advantages to using a compiler:

1. **Improved performance:** Compiled code tends to run faster than interpreted code because it has been translated into machine code that can be directly executed by the computer's processor.
2. **Portability:** Compilers allow programmers to write code in a high-level programming language that can be easily translated into machine code for a variety of different platforms.
3. **Increased Security:** Compilers can help improve the security of software by performing a number of checks on the source code, such as checking for syntax errors and enforcing type safety.
4. **Debugging support:** Most compilers include a number of debugging tools that can help programmers find and fix errors in their code.
5. **No dependencies:** Client or anyone else don't need any compiler, interpreter or third party program to be installed in their system for executing the shared executable file of source code.

Disadvantages of Compiler Design:

There are a few potential disadvantages of using a compiler in software development:

1. **Compilation time:** Depending on the size and complexity of the source code, compilation can take a significant amount of time.
2. **Error detection:** Compilers can only detect syntax errors and certain semantic errors, and may not catch all errors in the source code.
3. **Portability:** Programs compiled for a specific platform or architecture may not be able to run on other platforms or architectures without being recompiled..
4. **Execution speed:** Programs compiled from high-level languages may not be as fast as programs written in low-level languages, as the compiled code may include additional instructions for the compiler to interpret.
5. **Lack of flexibility:** Compilers can limit the flexibility of programs since changes often require recompilation.
6. **Resource consumption:** Compilers can consume system resources, particularly during compilation process, which may affect other tasks on the machine.

1(c)

Why we learn compiler design-

Compilers are an essential tool in software development, as they allow programmers to write code that is easier to read and write, can be easily compiled and run on different devices and platforms, and can be optimized for performance.

There are several reasons why compilers are used in software development:

1. **Ease of programming:** High-level programming languages are easier for humans to read and write than machine code, which is a series of numbers and symbols that can be difficult for humans to understand. By using a compiler to translate high-level language into machine code, programmers can write code more quickly and easily.
2. **Portability:** Compilers allow programmers to write code that can be easily compiled and run on a wide variety of devices and platforms. This is because the source code is independent of the underlying hardware and is only translated into machine code when it is compiled.
3. **Abstraction:** Compilers provide a level of abstraction between the programmer and the underlying hardware, allowing programmers to focus on the logic of their programs without having to worry about the specific details of the hardware.
4. **Performance:** Compilers can optimize the machine code generated from the source code, resulting in faster and more efficient programs.

Top-Down Parsing: Top-Down Parsing technique is a parsing technique which starts from the top level of the parse tree, moves downwards, and evaluate rules of grammar. The top-down parsing technique tries to identify the leftmost derivation for an input. It evaluates the rules of grammar while parsing. Consequently, each terminal symbol in the top-down parsing is produced by multiple productions of grammar rules.

Since top-down parsing uses leftmost derivation, hence in this parsing technique, the leftmost decision selects what production rule is used to construct the string.

Bottom-Up Parsing: Bottom-Up Parsing technique is again a parsing technique which starts from the lowest level of the parse tree, move upwards and evaluates the rules of grammar. Therefore, the bottom-up parsing technique makes an attempt to decrease the input string to the start symbol of the grammar.

The bottom-up parsing technique makes use of rightmost derivation. The main rightmost decision is to select when a production rule is used to reduce the string to get the starting symbol of the parsing tree.

Answer to the Question no- 2

(a)

Language Processing System

The computer is an intelligent combination of software and hardware. Hardware is simply a piece of mechanical equipment and its functions are being compiled by the relevant software. The hardware considers instructions as electronic charge, which is equivalent to the binary language in software programming. The binary language has only 0s and 1s. To enlighten, the hardware code has to be written in binary format, which is just a series of 0s and 1s. Writing such code would be an inconvenient and complicated task for computer programmers, so we write programs in a high-level language, which is Convenient for us to comprehend and memorize. These programs are then fed into a series of devices and operating system (OS) components to obtain the desired code that can be used by the machine. This is known as a language processing system.

Components of language processing system:

Preprocessor: It includes all header files and also evaluates whether a macro (A macro is a piece of code that is given a name. Whenever the name is used, it is replaced by the contents of the macro by an interpreter or compiler. The purpose of macros is either to automate the frequency used for sequences or to enable more powerful abstraction) is included. It takes source code as input and produces modified source code as output. The preprocessor is also known as a macro evaluator, the processing is optional that is if any language that does not support #include macros processing is not required.

Compiler: The compiler takes the modified code as input and produces the target code as output.

Assembler: The assembler takes the target code as input and produces real locatable machine code as output.

Linker: Linker or link editor is a program that takes a collection of objects (created by assemblers and compilers) and combines them into an executable program.

Loader: The loader keeps the linked program in the main memory.

Executable code: It is low-level and machine-specific code that the machine can easily understand. Once the job of the linker and loader is done the object code is finally converted it into executable code.

2(b)

Cross Compiler

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a cross compiler executes on machine X and produces machine code for machine Y.

Compilers are the tool used to translate high-level programming language to low-level programming language. The simple compiler works in one system only, but what will happen if we need a compiler that can compile code from another platform, to perform such compilation, the cross compiler is introduced.

2(c)

Source Compiler:

A source to source compiler (S2S compiler) is also referred to by three other names, the first is the source to source translator, the second is trans-compiler and the third one is transpiler. If we try to summarize the work on the source to source compiler is a sentence, it would be as follows:

A S2S Compiler is given as input the source code of a program to which it returns a source code with the overall same functionality in the same or different programming language.

Unlike the general compiler whose work is to convert a high-level programming language to a machine language that is binary, the source to source compiler converts one source code from one programming language to another programming language which is at the same level of compilation from machine language. For example, while the traditional compiler may convert *C* to *assembly* or *Java* to byte code, the source to source compiler may convert one scripting language to another such as JavaScript to Python, or C++ to Java.