SYSTEM SOFTWARE

**(IMPORTANT TOPICS SHORTTT NOTE)**

**UNIT 1**

**System Software:** System software refers to the collection of programs that manage and control the hardware components of a computer system. These programs are responsible for providing an interface between the computer hardware and the application software. Examples of system software include operating systems, device drivers, system utilities, and programming language translators.

**Loader:** A loader is a system software that loads an executable program into memory and prepares it for execution. The loader performs various functions, such as allocating memory, resolving references, and relocating addresses.

There are two types of loaders: absolute loaders and relocating loaders.

**Assembler:** An assembler is a program that translates assembly language code into machine language code. Assembly language is a low-level programming language that is closely related to the architecture of the computer's CPU. The assembler translates assembly language code into machine language code that can be executed by the computer's CPU.

**Compilers and Interpreters:** Compilers and interpreters are programs that translate high-level programming languages into machine language code. Compilers translate the entire program at once and generate an executable file that can be executed directly by the CPU. Interpreters, on the other hand, translate the program line by line and execute each line as it is translated.

**Operating System Types:** Operating systems can be classified into four types: real-time, single-user, multi-user, and distributed. A real-time operating system is used in systems that require immediate response times, such as air traffic control systems. A single-user operating system is used in personal computers. A multi-user operating system is used in servers and mainframe computers. A distributed operating system is used in networks of computers.

**Single-pass Assembler:** A single-pass assembler is an assembler that translates the source code into machine code in a single pass. In a single-pass assembler, the assembler reads the source code only once and generates the machine code as it reads the source code.

**Machine Language**, High-level Language, and Assembly Language: Machine language is the lowest level of programming language and consists of binary code that can be executed directly by the computer's CPU. High-level language is a programming language that is closer to human language and requires a compiler or an interpreter to translate it into machine language. Assembly language is a low-level programming language that is closely related to the architecture of the computer's CPU and requires an assembler to translate it into machine language.

**UNIT 2**

**Macro Processor:** A macro processor is a program that performs text substitution on the source code before it is compiled. The macro processor replaces the macro calls with the corresponding macro code before the source code is compiled.

**Macro Calls:** A macro call is a statement in the source code that invokes a macro. A macro call specifies the name of the macro and the arguments to be passed to the macro.

**Features of Macros:** Macros have several features, such as the ability to define formal parameters, the ability to expand into multiple statements, and the ability to perform conditional execution.

**Formal Parameters in Macro:** Formal parameters are variables that are used in the macro definition to represent the arguments passed to the macro. The formal parameters are replaced with the actual arguments when the macro is called.

**UNIT 3**

**Different Types of Loaders Based on Functionalities:** There are several types of loaders based on their functionalities, such as absolute loaders, relocating loaders, linking loaders, and dynamic loaders.

**Loading - Absolute Loader:** An absolute loader is a loader that loads the executable program into a specific memory location. An absolute loader generates machine language code that contains absolute addresses.

**Linking and Relocating Loader Scheme:** A linking and relocating loader scheme is a loader that combines several object modules into a single executable program. The loader resolves external references between the object modules and relocates the program to the correct memory locations.

**Relocation and Linking-Concept:** Relocation is the process of adjusting the memory addresses in an object module to match the actual memory locations where the program will be loaded. Linking is the process of combining multiple object modules into a single executable program and resolving external references between the object modules.

**Linker and Loader Difference:** The main difference between a linker and a loader is that a linker combines multiple object modules into a single executable program, while a loader loads the executable program into memory and prepares it for execution.

**Overlays:** Overlays are a technique used to overcome memory constraints in a computer system. An overlay is a portion of a program that is loaded into memory only when it is needed and is then unloaded when it is no longer needed. Overlays are used to load only the portions of a program that are required for a particular task, conserving memory space.

**Dynamic Loading- Advantageous and Disadvantageous:** Dynamic loading is a technique used to load only the portions of a program that are needed at runtime. This technique has several advantages, such as conserving memory space and reducing the startup time of a program. However, dynamic loading can also slow down the execution of a program due to the overhead of loading and unloading code at runtime.

**UNIT 4**

**Phases of a Compiler:** A compiler has several phases, such as lexical analysis, syntax analysis, semantic analysis, code generation, and optimization. In lexical analysis, the source code is divided into tokens. In syntax analysis, the tokens are organized into a parse tree. In semantic analysis, the compiler checks the semantics of the program. In code generation, the compiler generates the machine code. In optimization, the compiler optimizes the machine code to improve its efficiency.

**Need of Symbol Table in Assembler:** A symbol table is used by an assembler to keep track of the symbols used in the source code, such as variable names and function names. The symbol table is used to resolve references to symbols and to generate the machine code.

**Compiler- Structure and Diagram:** A compiler has several components, such as the front end, middle end, and back end. The front end consists of the lexical analyzer, syntax analyzer, and semantic analyzer. The middle end consists of the optimizer. The back end consists of the code generator.

**Optimization- Various Techniques:** Compiler optimization refers to the techniques used by the compiler to improve the efficiency of the machine code. Some optimization techniques include constant folding, loop unrolling, and common subexpression elimination. These techniques aim to reduce the number of instructions executed and to minimize memory accesses.

**Machine Dependent and Independent Optimization:** Machine-dependent optimization refers to the optimization techniques that depend on the architecture of the target machine. Machine-independent optimization refers to the optimization techniques that are not specific to any particular architecture and can be used on any machine.

**UNIT V:**

**YAAC- Working with an Example:** YAAC (Yet Another Compiler Compiler) is a tool used to generate parsers and lexers for programming languages. It uses a grammar file to generate the parser and a regular expression file to generate the lexer. An example of using YAAC is to generate a parser and lexer for a new programming language.

**Structure of LEX Program with Example:** LEX is a tool used to generate a lexer for programming languages. It uses regular expressions to recognize patterns in the input text and generate tokens.

The structure of a LEX program consists of three parts:

the definitions section,

the rules section,

and the user subroutines section.

An example of a LEX program is to generate tokens for a programming language that uses keywords, identifiers, and operators.