__/__/____

Sample Paper 1

Q1.) Define System Programming with example

Ans.) Development of Software that will perform certain tasks and when working together as a single unit they allow entire system to run.

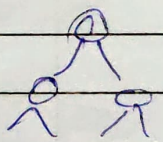
e.g Operating System, Data backup Server, Web Site Server, etc.

Q2.) List various techniques of code optimization.

Code optimization.
Compiler - 5th phase.

Phases of compiler.

HLL
↓
lexical Analyzer.
↓ Stream of Token
Syntax Analyzer.
↓ Phase Tree
Semantic Analyzer.
↓ Parse Tree (Symatically verified.)
Intermediate Code Generation
↓ Three address code.
Code Optimization
↓ Intermediate code.
Target code generation.
↓
Assembly language



__/__/____

Code optimization

Platform Dependent	Platform Independent
* Peephole optimization.	* loop optimization.
* Instruction level Parallelism.	* Constant folding.
* Data level Parallelism.	* Constant propogation.
* Redundant level parallelism	* Common Subparsion Elimination.

Q3.) Why it is required to relocate the programs sometimes.

To adjust all address dependent locations, such as address constains, to correspond to the allocated space.

Q4.) Highlighting the need of linking in programming?

When we write any code in our compiler no matter the language it will first go in the translator which will convert this high level language to machine code which will then go to linkers this linkers main function is to link the object with the library functions for e.g printf to stdio.h and then it will be converted to the object code.

Q5) Discuss the term cold Booting?

Ans Coold Booting refers to the term when we start our System from hybernate state and all the hardware and Software were already in rest state and we turn on our computer or laptop we call it as cold Booting also known as hard booting or Dead Booting.

Section B

4 x 5 = 20

Q6) Elaborate the term nested macro calls? Explain with the help of coding example.

Ans

Macro

Macro is just like set of functions which contains steps and can be called in future codes. Macro does not return value.

```
I Defining
  MACRO
name → cal A P
  ADD AREG A P } Encapsulat
  SUB BREG A P }
  MEND
```

II Calling Cal & → Parameter (place &P)

Types of Parameters.

(1) Positional Parameters.

```
Macro
M1  &P1 , &P2 , &P3
     ↑     ↑     ↑
M1  BREG ,  A ,  B
```

(2) Keyword Parameters.

```
Macro
M1  &P1=, &P2=, &P3=
 :
 :
M1  &P1=A, &P2=B, &P3=C
```

(3) Default Parameters.

```
Macro
M1  &P1=A , &P2=B
M1  &P1=  , &P2=C
```

And it will override it.

It will take default value

(4) Mixed Parameters.

contains Mixed Parameters.

Nested Macro.

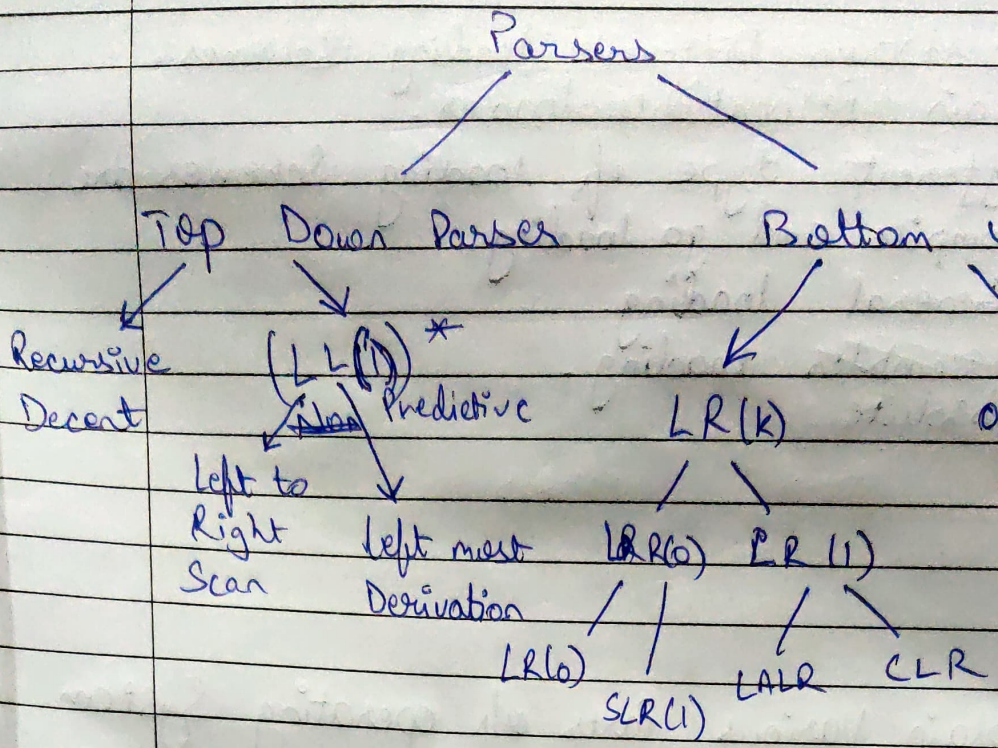
```
Macro
  CAL   &P
MOVER   AREA  &P
ADD     &P,   '1'
MEND
```

```
Macro
Cal 1   &P1, &P2, &P3
Cal     &P1
Cal     &P2
Cal     &P3
```

Cal 1 is outer Macro.
Cal is inner Macro.

Q7) How many types of Parser are there. Explain Recursive Descent Parser.
Parsing - Deriving a String from a given grammer.
Part of Syntax analysis.
Uses parse tree. (CFG) Context Free grammer.

__/__/____



Recursive Descent → Brute force.

__/__/____

Q8) List Down Different loading Schemes. Explain Relocatable loader.

Ans Different type of loading Schemes are
(1) Compile and go loading.
(2) General loading.
(3) ~~Assembler~~ Absolute loading.

Q9) Explain Various Parts of operating System.

(1) Resource Manager — When a multiple user request for data this is then used to manage the resources so that the task handling is done easily.
(2) Process Management — Multiple Process are open and we have to see which to prioratize so we use CPU Scheduling Algorithms.
(3) Storage Management (HD) — How to store data using File System (NTFS)(FTSF)
(4) Memory Management (RAM) —
(5) Security & Privacy. — Windows uses carbaro Security.

__/__/__

Sample Paper 3

Q1.) Briefly define functions of Macro Assembler.

function of Macro are.

(1) Defining
Macro
Cal &P
ADD &REG &P
SUB &REG &P
MEND

(2) Calling
Cal x

(3) Expand

Q2.) Define the term Ambiguity wrt Parsing.

Ambiguity in NLP (Natural language Process)

. When a Sentence have more than one meaning.
There are basically 5 type of Ambiguity
(1) Lexical
(2) Sytactic
(3) Semantic
(4) Anaphoric
(5) Pragmatic.

__/__/__

(1) lexical
When one word have many meaning
e.g I won Silver medal.
She has silver her hair.

(2) Syntactic
This type of Ambiguity occurs when sentence is passed in different ways or gramatically different...

(3) Semantic.
meaning of sentence is different even after the meaning of individual word have been resolved.
e.g * Seema loves her mother and Shreya does too

(4) Anaphoric Ambiguity.
when same meaning of sentence is repeated several times.
e.g my mother liked the house very much but she couldn't buy it.

(5) Pragmatic Ambiguity - When a sentence gives multiple interpretation or it is not specific.
e.g : I like you too.

~~190~~ 38
~~15~~ 12-6
3

Q3) List various program extentions during program writing, translation & execution.

Q4.) Justify the need of operating System in Program Execution.

Operating System is System Software which acts as a bridge between user and hardware, when we want to access the hardware.

Operating System major goal is to provide convinience.

(1) Resource governer.
(2) Memory management.
(3) Storage management.
(4) Security & Protection.

Q5.) List the features of MS DOS.

(1) Command line interface.
(2) Device independent I/O process.
(3) Directory Structure.
e.g Root Directory; Sub Directory, File texting.
(4) Written in Assembly language.
(5) Error Recovery.
(6) Device Management
(7) Dos text editor.

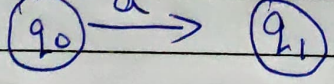
(Q6.) Discuss System Software development with all steps.

(1) Planning → Idea
(2) Defining / Analysis → Cost, Time, SRS doc
(3) Designing phase → Appearence & what you want to see. Or layout
(4) Coding / implementation —
(5) Testing
(6) Deployment / Maintainance.

(Q7.) Briefly explain all tuples of DFA. Make a transition diagram of all the regular grammar given as
— String ended with 101

DFA — Deterministic Finite Automata.
long Representation

DFA $(Q, \Sigma, \delta, q_0, F)$



If we apply alphabets in any of the automata state it will g take you to some state.
Q → Set of finite state.
Σ → Set of alphabets.
δ → Transition
q_0 → Start
F → Set Of Finite Final State.
$F \subseteq Q$

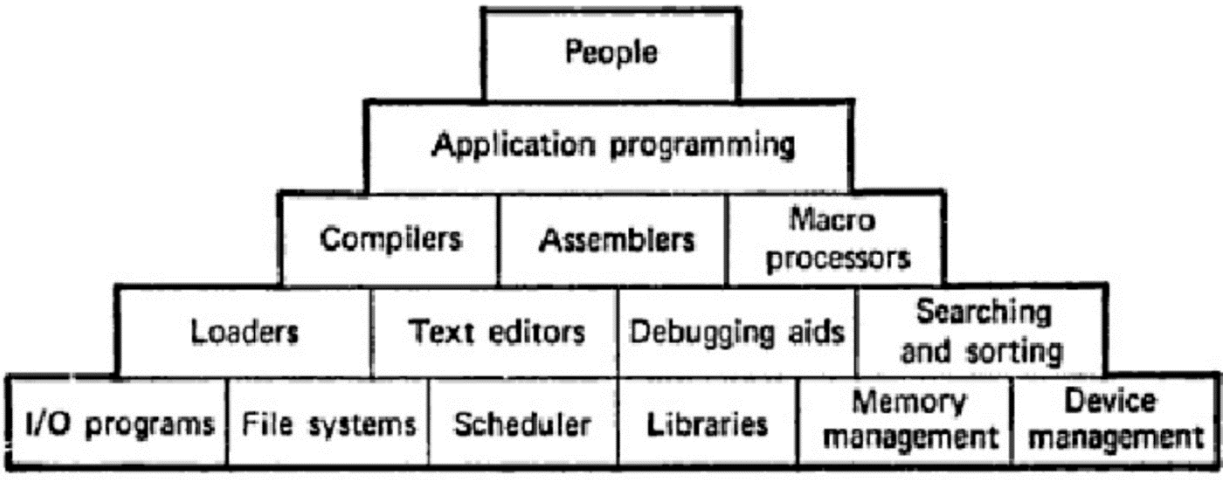
1. Define System Programming with example.

System programming involves designing and writing computer programs that allow the computer hardware to interface with the programmer and the user, leading to the effective execution of application software on the computer system.

Typical system programs include the operating system and firmware, programming tools such as compilers, assemblers, I/O routines, interpreters, scheduler, loaders and linkers as well as the runtime libraries of the computer programming languages.

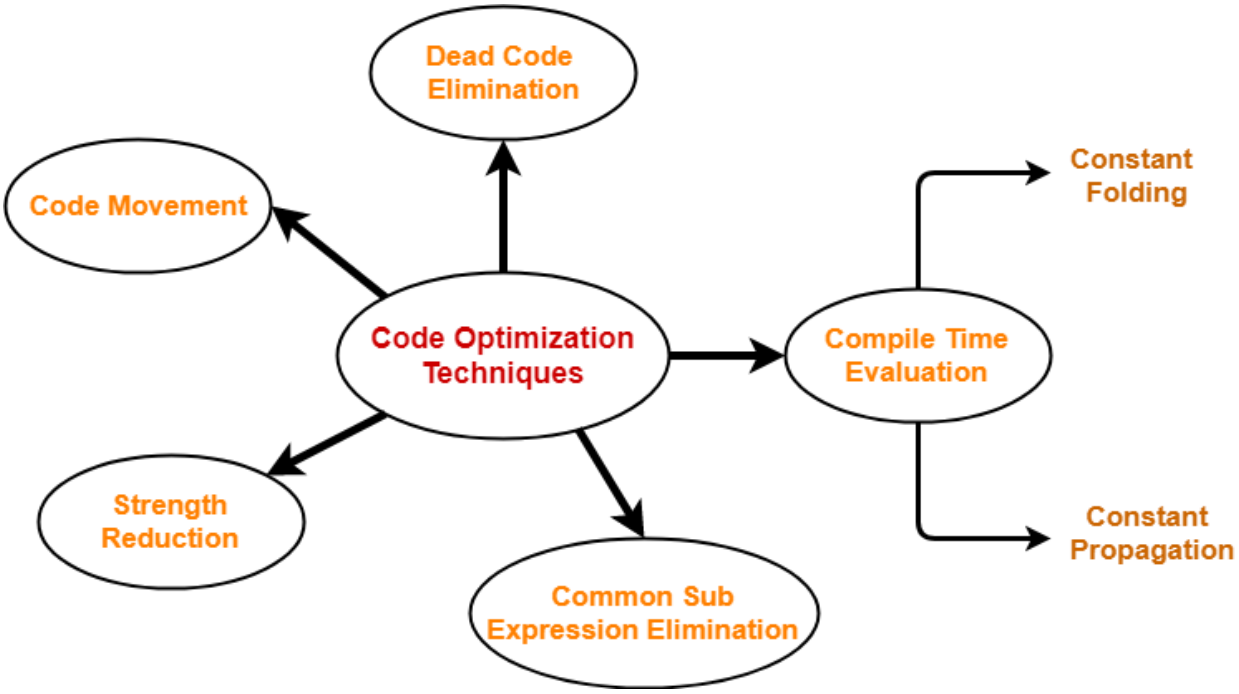
The system programs are responsible for the development and execution of a program and they can be used by the help of system calls because system calls define different types of system programs for different tasks.

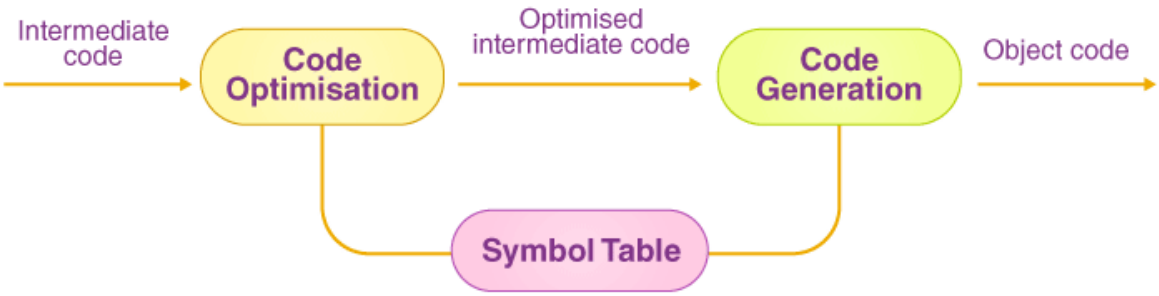
- File management
- Status information
- Programming language supports
- Programming Loading and execution
- Communication
- Background services



2. List Various techniques of Code optimization

Code optimization is a program modification strategy that endeavours to enhance the intermediate code, so a program utilises the least potential memory, minimises its CPU time and offers high speed.





Machine-independent Optimization

In this optimization, the compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations.

Machine-dependent Optimization

Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references. Machine-dependent optimizers put efforts to take maximum advantage of memory hierarchy.

3. Why it is required to relocate the programs sometimes.

Relocation is the process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction must transfer control to the proper destination address at execution. In other words, relocatable files must have information that describes how to modify their section contents, thus allowing executable and shared object files to hold the right information for a process's program image. Relocation entries are these data.

Relocation is the process of assigning load addresses for position-dependent code and data of a program and adjusting the code and data to reflect the assigned addresses.[1][2] Prior to the advent of multiprocess systems, and still in many embedded systems, the addresses for objects were absolute starting at a known location, often zero.

Since multiprocessing systems dynamically link and switch between programs it became necessary to be able to relocate objects using position-independent code. A linker usually performs relocation in conjunction with symbol resolution, the process of searching files and libraries to replace symbolic references or names of libraries with actual usable addresses in memory before running a program.

Relocation is typically done by the linker at link time, but it can also be done at load time by a relocating loader, or at run time by the running program itself. Some architectures avoid relocation entirely by deferring address assignment to run time; as, for example, in stack machines with zero address arithmetic or in some segmented architectures where every compilation unit is loaded into a separate segment.

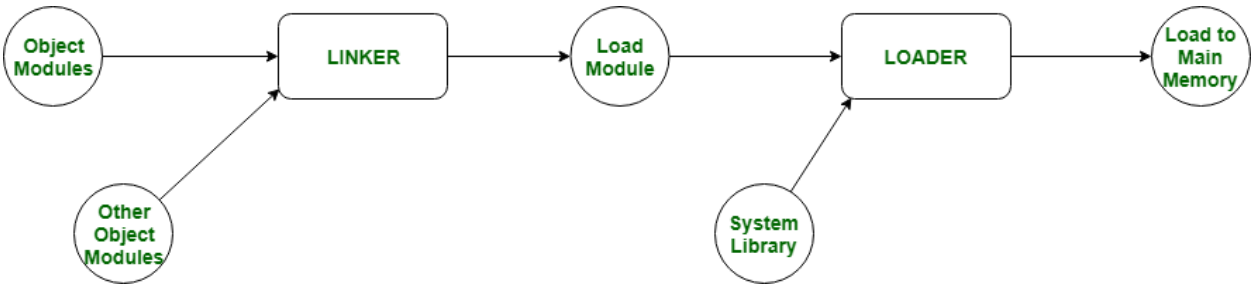
4. Highlight the need of Linking in programming?

Linker is a program in a system which helps to link object modules of a program into a single object file. It performs the process of linking. Linkers are also called as link editors. Linking is a process of collecting and maintaining piece of code and data into a single file.

Linker also links a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for the loader. Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.

The job of the linker is to combine all of the object files and the start-up code into one machine language file called the executable and assign addresses for global variables. The linker relocates the data and instructions in the object files so that they are not all on top of each other.

Source code -> compiler -> Assembler -> Object code -> Linker -> Executable file -> Loader.



5. Discuss the term cold Booting.

A cold boot refers to the general process of starting the hardware components of a computer, laptop or server to the point that its operating system and all startup applications and services are launched. Cold boot is also known as hard boot, cold start or dead start.

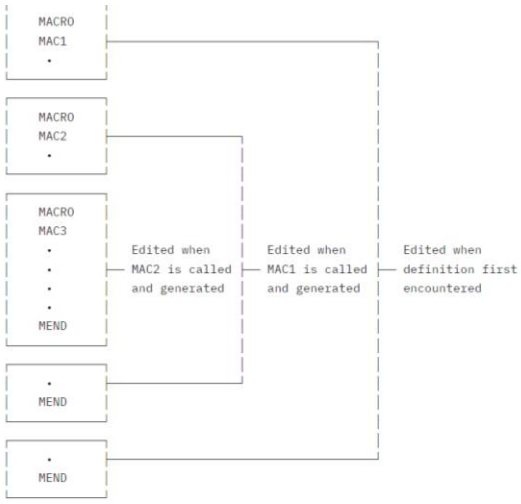
A cold boot removes power and clears memory (RAM) of all internal data and counters that keep track of operations, which are created by the OS and applications when they run. Erratic program behavior is often cured with a cold boot, also known as a "hard boot."

The technique of restarting a system from a power-off state and returning it to normal function is known as cold booting. On the other hand, Warm booting is the technique of restarting an already-on system without completely turning it off. Cold booting conducts a more thorough system reset than warm booting.

6. Elaborate the term nested macro calls? Explain with the help of a coding example

A nested macro instruction definition is a macro instruction definition you can specify as a set of model statements in the body of an enclosing macro definition. This lets you create a macro definition by expanding the outer macro that contains the nested definition.

All nested inner macro definitions are effectively "black boxes": there is no visibility to the outermost macro definition of any variable symbol or sequence symbol within any of the nested macro definitions. This means that you cannot use an enclosing macro definition to tailor or parameterize the contents of a nested inner macro definition.



7. How many types of Parser are there. Explain Recursive Descent Parser.

Types of Parser:

The parser is mainly classified into two categories, i.e. Top-down Parser, and Bottom-up Parser. These are explained below:

Top-Down Parser:

The top-down parser is the parser that generates parse for the given input string with the help of grammar productions by expanding the non-terminals i.e. it starts from the start symbol and ends on the terminals. It uses left most derivation.

Further Top-down parser is classified into 2 types: A recursive descent parser, and Non-recursive descent parser.

Recursive descent parser is also known as the Brute force parser or the backtracking parser. It basically generates the parse tree by using brute force and backtracking.

Non-recursive descent parser is also known as LL(1) parser or predictive parser or without backtracking parser or dynamic parser. It uses a parsing table to generate the parse tree instead of backtracking.

Bottom-up Parser:

Bottom-up Parser is the parser that generates the parse tree for the given input string with the help of grammar productions by compressing the non-terminals i.e. it starts from non-terminals and ends on the start symbol. It uses the reverse of the rightmost derivation.

Further Bottom-up parser is classified into two types: LR parser, and Operator precedence parser.

LR parser is the bottom-up parser that generates the parse tree for the given string by using unambiguous grammar. It follows the reverse of the rightmost derivation.

LR parser is of four types:

(a)LR(0)

(b)SLR(1)

(c)LALR(1)

(d)CLR(1)

Operator precedence parser generates the parse tree from given grammar and string but the only condition is two consecutive non-terminals and epsilon never appears on the right-hand side of any production.

The operator precedence parsing techniques can be applied to Operator grammars.

Operator grammar: A grammar is said to be operator grammar if there does not exist any production rule on the right-hand side.

1. as ε(Epsilon)

2. Two non-terminals appear consecutively, that is, without any terminal between them operator precedence parsing is not a simple technique to apply to most the language constructs, but it evolves into an easy technique to implement where a suitable grammar may be produced.

8. List down different Loading schemes. Explain Relocatable loader.

General Loader Scheme In this loader scheme, the source program is converted to object program by some translator (assembler). The loader accepts these object modules and puts machine instruction and data in an executable form at their assigned memory. The loader occupies some portion of main memory.

Absolute Loader Absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at specified locations in the memory. This type of loader is called absolute because no relocation information is needed; rather it is obtained from the programmer or assembler.

Direct Linking Loaders The direct linking loader is the most common type of loader. The loader cannot have the direct access to the source code. The assembler should give the following information to the loader i. The length of the object code segment ii. The list of all the symbols which are not defined in the current segment but can be used in the current segment. iii. The list of all the symbols which are defined in the current segment but can be referred by the other segments.

9. Explain various parts of operating System.

An operating system is a large and complex system that can only be created by partitioning into small parts. These pieces should be a well-defined part of the system, carefully defining inputs, outputs, and functions.

Although Windows, Mac, UNIX, Linux, and other OS do not have the same structure, most operating systems share similar OS system components, such as file, memory, process, I/O device management. The components of an operating system play a key role to make a variety of computer system parts work together. There are the following components of an operating system, such as:

1. Process Management
2. File Management
3. Network Management
4. Main Memory Management
5. Secondary Storage Management
6. I/O Device Management
7. Security Management
8. Command Interpreter System

Operating system components help you get the correct computing by detecting CPU and memory hardware errors. An OS is a complex system. It has many system components like file management, process management, network management, I/O device management, main memory management, secondary-storage management, and security management. There are other activities that an OS performs. It helps check a program's capability, execute I/O operations, detects errors, ensures correct computing, and ensures information exchange among processes.

10.

Explain w.r.t Assembler

a) Data Structures of Pass 1

Operation Code Table (OPTAB) It is used to lookup mnemonic operation codes and translates them to their machine language equivalents. In more complex assemblers the table also contains information about instruction format and length.

Symbol Table (SYMTAB) This table includes the name and value for each label in the source program, together with flags to indicate the error conditions (e.g., if a symbol is defined in two different places).

Location Counter (LOCCTR) Apart from the SYMTAB and OPTAB, this is another important variable which helps in the assignment of the addresses. LOCCTR is initialized to the beginning address mentioned in the START statement of the program. After each statement is processed, the length of the assembled instruction is added to the LOCCTR to make it point to the next instruction. Whenever a label is encountered in an instruction the LOCCTR value gives the address to be associated with that label.

In pass 1 the OPTAB is used to look up and validate the operation code in the source program and to find the instruction length for incrementing LOCCTR. In pass 2, it is used to translate the operation codes to machine language.

During Pass 1, labels are entered into the symbol table along with their assigned address value as they are encountered. All the symbols address value should get resolved at the pass 1.

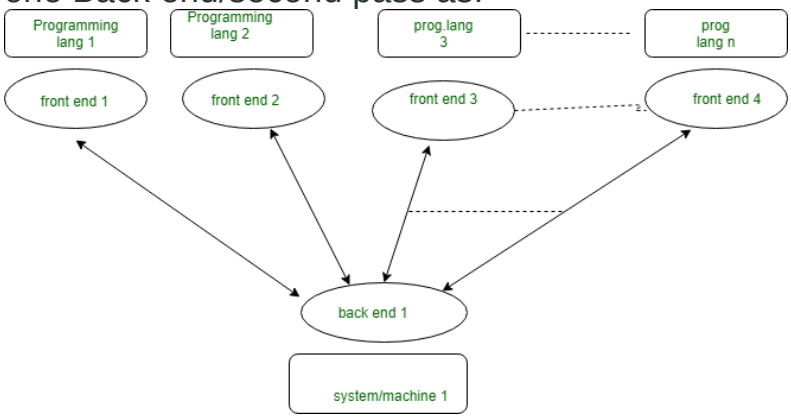
During Pass 2, symbols used as operands are looked up the symbol table to obtain the address value to be inserted in the assembled instructions. SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.
LOCCTR is initialized to the beginning address mentioned in the START statement of the program. After each statement is processed, the length of the assembled instruction is added to the LOCCTR to make it point to the next instruction. Whenever a label is encountered in an instruction the LOCCTR value gives the address to be associated with that label.

b) Multipass Assembler

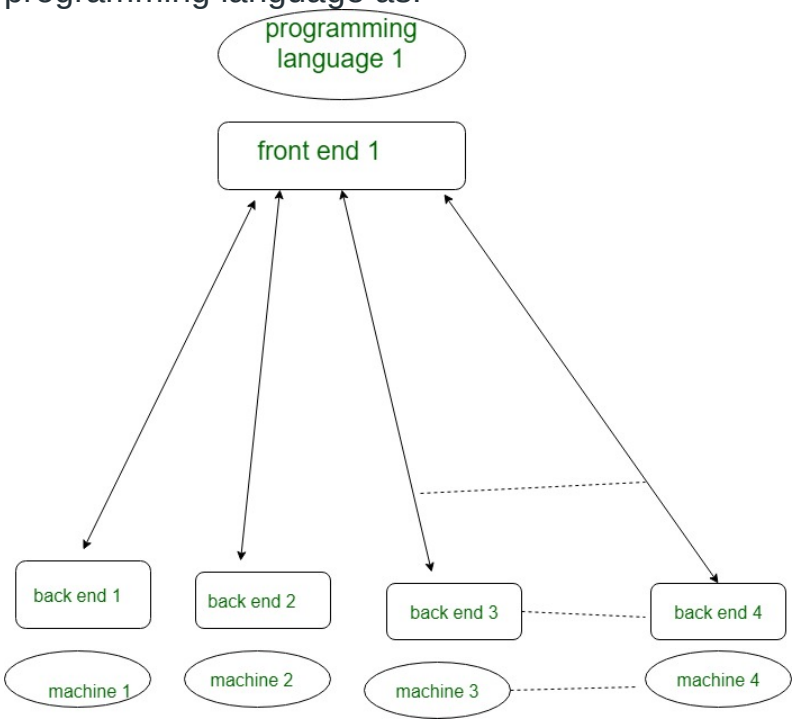
Multipass assembler means more than one pass is used by assembler. • Multipass assembler is used to eliminate forward references in symbol definition. • Forward References Problem - The assembler specifies the symbols should be declared anywhere in the in the program.

With multi-pass Compiler we can solve these 2 basic problems:

1. If we want to design a compiler for different programming language for same machine. In this case for each programming language there is requirement of making Front end/first pass for each of them and only one Back end/second pass as:



2. If we want to design a compiler for same programming language for different machine/system. In this case we make different Back end for different Machine/system and make only one **Front end** for same programming language as:



Differences between Single Pass and Multipass Compilers:

Parameters	Single pass	multi Pass
Speed	Fast	Slow
Memory	More	Less
Time	Less	More
Portability	No	Yes

11.

List various tools of Compilation. Discuss the case studies of
LEX
YACC

The compiler writer can use some specialized tools that help in implementing various phases of a compiler. These tools assist in the creation of an entire compiler or its parts. Some commonly used compiler construction tools include:

Parser Generator –

It produces syntax analyzers (parsers) from the input that is based on a grammatical description of programming language or on a context-free grammar. It is useful as the syntax analysis phase is highly complex and consumes more manual and compilation time.

Example: PIC, EQM

Scanner Generator –

It generates lexical analyzers from the input that consists of regular expression description based on tokens of a language. It generates a finite automaton to recognize the regular expression.

Example: Lex

Syntax directed translation engines –

It generates intermediate code with three address format from the input that consists of a parse tree. These engines have routines to traverse the parse tree and then produces the intermediate code. In this, each node of the parse tree is associated with one or more translations.

Automatic code generators –

It generates the machine language for a target machine. Each operation of the intermediate language is translated using a collection of rules and then is taken as an input by the code generator. A template matching process is used. An intermediate language statement is replaced by its equivalent machine language statement using templates.

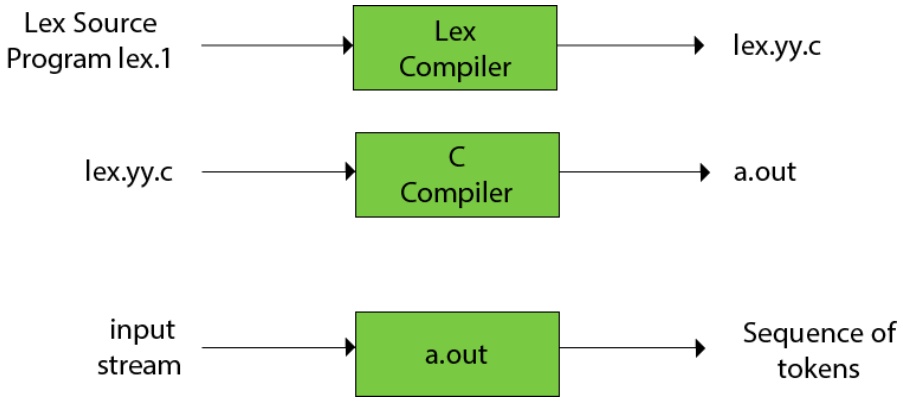
Data-flow analysis engines –

It is used in code optimization. Data flow analysis is a key part of the code optimization that gathers the information, that is the values that flow from one part of a program to another. Refer – data flow analysis in Compiler

Compiler construction toolkits –

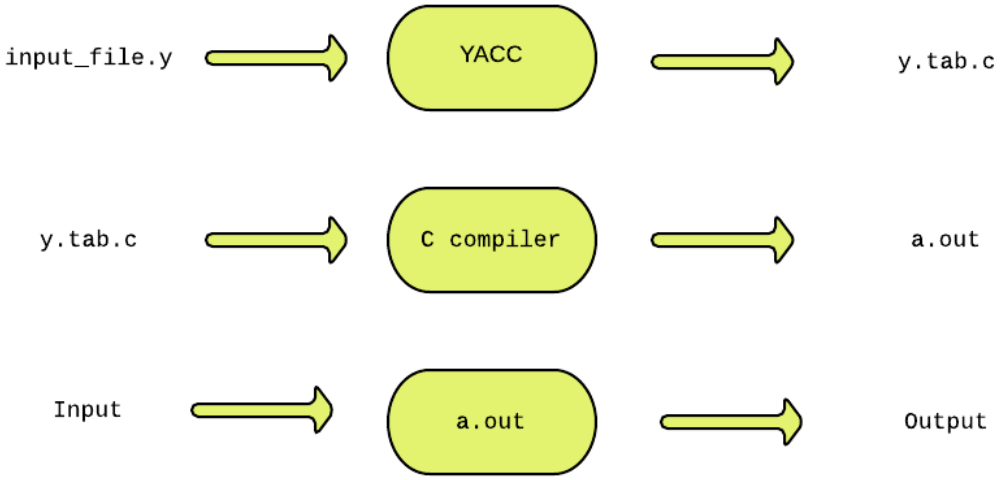
It provides an integrated set of routines that aids in building compiler components or in the construction of various phases of compiler.

- Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.



A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR(1) (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.



12.

Define Editor. Justify their need. Elaborate each type with examples.

Editors or text editors are software programs that enable the user to create and edit text files. In the field of programming, the term editor usually refers to source code editors that include many special features for writing and editing code. Notepad, Wordpad are some of the common editors used on Windows OS and vi, emacs, Jed, pico are the editors on UNIX OS. Features normally associated with text editors are — moving the cursor, deleting, replacing, pasting, finding, finding and replacing, saving etc.

Types of Editors

There are generally five types of editors as described below:

1. **Line editor**: In this, you can only edit one line at a time or an integral number of lines. You cannot have a free-flowing sequence of characters. It will take care of only one line.
   Ex : Teleprinter, edlin, teco

2. **Stream editors**: In this type of editors, the file is treated as continuous flow or sequence of characters instead of line numbers, which means here you can type paragraphs.
   Ex : Sed editor in UNIX
3. **Screen editors**: In this type of editors, the user is able to see the cursor on the screen and can make a copy, cut, paste operation easily. It is very easy to use mouse pointer.
   Ex : vi, emacs, Notepad

4. **Word Processor**: Overcoming the limitations of screen editors, it allows one to use some format to insert images, files, videos, use font, size, style features. It majorly focuses on Natural language.
5. **Structure Editor**: Structure editor focuses on programming languages. It provides features to write and edit source code.
   Ex : Netbeans IDE, gEdit.

Some other editors:

- **Full Screen Editors**: In computers, a full-screen editor or distraction-free editor is a text editor that occupies full display with the purpose of sidelining the user from the OS and the other applications. Ex : Acme, Coderoom, FocusWriter
- **Multiple Window Editor**: Multiple window editor allows you to work on more than one file at a time and cut and paste text from file into another via yanking and putting. The two fundamental concepts that lie behind multi-window editors are buffer and windows.
- **DOS-Editor**: MS-DOS editor or sometimes also known as just edit is a character based text editor that comes with MS-DOS and a 32-bit version of windows. Previously, it was QBASIC running in editor mode but after DOS-7, it became a standalone program.
- **VI editor** : The vi editor (short name for the visual editor) is a screen editor which is available in UNIX OS. Vi has no menus instead it uses a combination of keystrokes to accomplish tasks.
- **Online Editors**: Online text editors is an interface for editing the texts within a web browser. It aims to reduce the efforts made by the user by directly editing and updating into a valid HTML markup language.
  Ex : CKEditor, SnapEditor, designmode by Internet Explorer.

Editing Process

We all by now understand that editors are the program which is used to create, edit and modify a document. A document may include some images, files, text, equations, and diagrams as well. But we will be limited to text editors only whose main elements are character strings.

The document editing process mainly compromises of the following four tasks :

- The part of the document to edited or modifies is selected.

- determining how to format this lines on view and how to display it.
- Specify and execute the operations that modify the document.
- Update the view properly.

The above steps include filtering, formatting, and traveling.

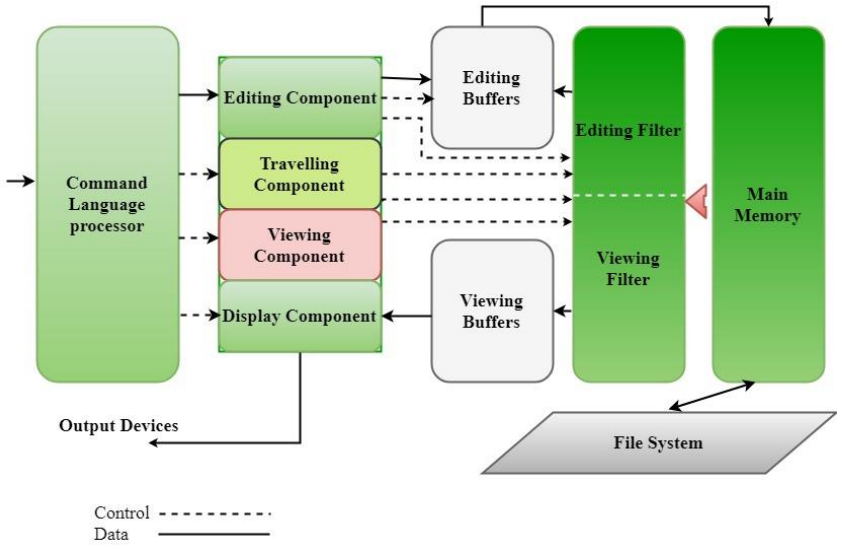
- Formatting : Visibility on display screen.
- Filtering : Finding out the main/important subset.
- Traveling : Locating the area of interest.

User Interface of editors: The user interface of editors typically means the input, output and the interaction language. The input devices are used to enter text, data into a document or to process commands

Input Devices : Input devices are generally divided as text input, button devices and locator devices. Text device is a keyboard. Button devices are special function keys. The locator devices include the mouse. There are special voice devices as well which writes into text whatever you speak.

- ***Output Devices :*** TFT monitors, Printers, Teletypewriters, Cathode ray tube technology, Advanced CRT terminals.
- ***Interaction language :*** The interaction language could be, typing oriented or text command-oriented or could be menu oriented user interface as well.

Editor Structure



The command language processor accepts commands, performs functions such as editing and viewing. It involves traveling, editing, viewing and display. Editing operations are specified by the user and display operations are specified by the editor. Traveling and viewing components are invoked by the editor or the user itself during the operations.

Editing component is a module dealing with editing tasks. The current editing area is determined by the current editing pointer associated with the editing component. When editing command is made, the editing component calls the editing filter, generates a new editing buffer. Editing buffer contains the document to be edited at the current editor pointer location.

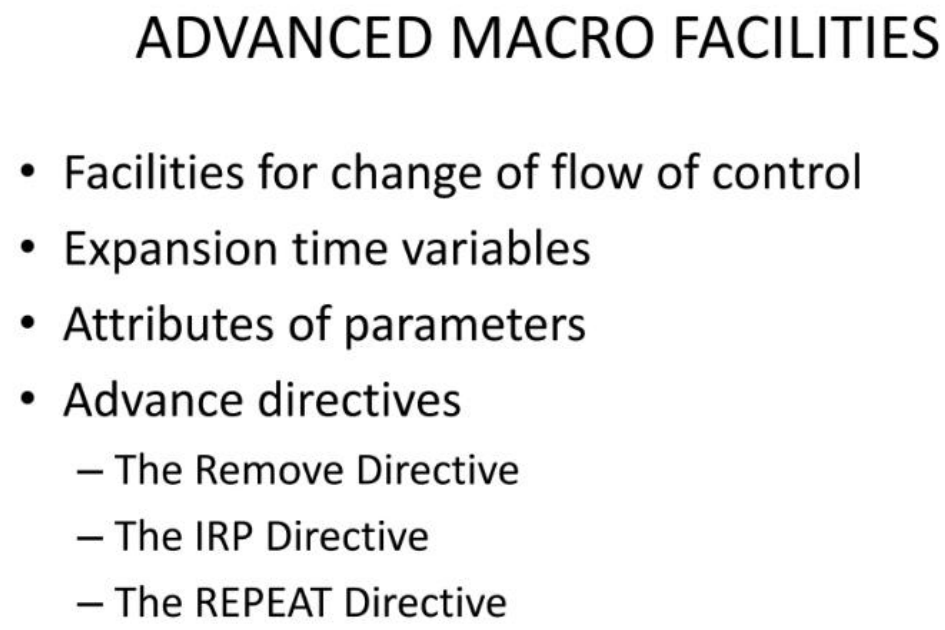
In viewing a document, the start of the area to be viewed is determined by the current viewing pointer. Viewing component is a collection of modules used to see the next view. Current viewing can be made to set or reset depending upon the last operation.

When display needs to be updated, the viewing component invokes the viewing filter, generates a new buffer and it contains the document to be viewed using the current view buffer. Then the viewing buffer is pass to the display component which produces the display by buffer mapping.

The editing and viewing buffers may be identical or completely disjoint. The editing and viewing buffers can also partially overlap or can be contained one within the another. The component of the editor interacts with the document from the user on two levels: main memory and the disk files system.

1. Highlight the term Advanced Macro facilities

Advanced macro facilities are aimed to supporting semantic expansion. Used for: Performing conditional expansion of model statements and in writing expansion time loops.



For change of flow of control:-

Two features are provided for this purpose

1) **Expansion time Sequencing Symbol**
2) **Expansion time Statements (AIF,AGO,ANOP)**

Sequencing Symbol(SS)

SS is defined by putting it in the label field of a stmt in the macro body.

Statements (syntax)

AIF-> AIF (<expression>) <Sequencing Symbol (SS)>

Expression includes relational expression (LT, NE etc.). If condition is true then control is transferred to SS.

AGO-> AGO <SS> → unconditional transfer

ANOP-> <SS> ANOP → used to define SS

2. List various Debugging Techniques.

- Different techniques are better suited to different cases. There is no precise best method. It all depends on the type of codebase you are testing and your preferences. Following are widely used debugging techniques:

1. Trace-based debugging
2. Spectrum-based debugging
3. Delta debugging

- Trace-based debugging is traditional and the most common debugging technique used in most debugging tools today.
- Trace-based debugging is predicated on the concept of breakpoints.
- A breakpoint is a pausing or stopping point added to the program to examine the state of program execution up until that point.
- In spectrum-based debugging, also known as spectrum-based fault localization (SFL), the debugging process is done by monitoring the statements included in a particular execution tree.
- This is achieved by using the program spectrum to identify the active part of the program during its run.
- The process of delta debugging (DD) is to minimize automated test cases.
- It takes test cases that may cause the error and prepares an error report. From that error report, minimal test cases are selected based on their high probability of producing the error.

The process of finding bugs or errors and fixing them in any application or software is called debugging. To make the software programs or products bug-free, this process should be done before releasing them into the market.

3. Justify the role of Relocation in programming.

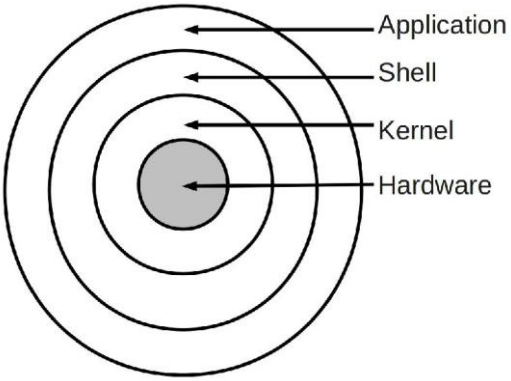
Relocation is the process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction must transfer control to the proper destination address at execution.

In other words, relocatable files must have information that describes how to modify their section contents, thus allowing executable and shared object files to hold the right information for a process's program image. Relocation entries are these data.

A relocating loader is capable of loading a program to begin anywhere in memory: The addresses produced by the compiler run from 0 to L–1. After the program has been loaded, the

addresses must run from N to N +L–1. Therefore, the relocating loader adjusts, or relocates, each address in the program.

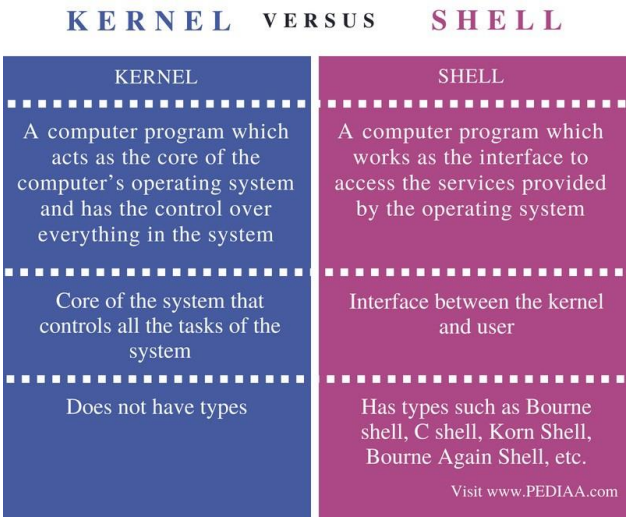
4. Define the term Shell.



The shell is the outermost layer of the operating system. Shells incorporate a programming language to control processes and files, as well as to start and control other programs. The shell manages the interaction between you and the operating system by prompting you for input, interpreting that input for the operating system, and then handling any resulting output from the operating system.

Shells provide a way for you to communicate with the operating system. This communication is carried out either interactively (input from the keyboard is acted upon immediately) or as a shell script. A *shell script* is a sequence of shell and operating system commands that is stored in a file.

When you log in to the system, the system locates the name of a shell program to execute. After it is executed, the shell displays a command prompt. This prompt is usually a $ (dollar sign). When you type a command at the prompt and press the Enter key, the shell evaluates the command and attempts to carry it out. Depending on your command instructions, the shell writes the command output to the screen or redirects the output. It then returns the command prompt and waits for you to type another command.



5. Justify the importance of Booting in Operating System.

In computing, booting is the process of starting a computer as initiated via hardware such as a button or by a software command. After it is switched on, a computer's central processing unit (CPU) has no software in its main memory, so some process must load software into memory before it can be executed.

Booting is basically the process of starting the computer. When the CPU is first switched on it has nothing inside the Memory. In order to start the Computer, load the Operating System into the Main Memory and then Computer is ready to take commands from the User.

Cold Boot	Warm Boot
Cold booting commonly goes by hard booting in computer terminology.	Warm booting is also called as soft booting.
The system starts up from a completely powerless state.	The system returns to its original state without interrupting power.
It completely resets the hardware and clears the system off the temporary memory.	It doesn't necessarily reset the components and the power source, thereby keeping the memory intact ever after a reboot.
A cold boot is usually done when the system doesn't respond to a warm boot.	A warm boot is generally done when a program fails to respond and the system freezes in between a session.
Shutting it off from the power source or unplugging the supply resets the system.	Pressing the ctrl, alt and delete keys simultaneously or initiating a reset command will reboot the system without hampering power.
It runs self-diagnosis tests thereby resetting the hardware and memory.	It forbids a full system diagnosis thereby reducing the reboot time.

6. Compare Assembler Vs Compiler. Discuss about Elements of Assembly Language Programming.

- Real Number Constants.
- Character Constants.
- String Constants.
- Reserved Words.
- Instruction mnemonics, such as MOV, ADD, and MUL.
- Register names.
- Directives.
- Attributes

Assembly language is basically like any other language, which means that it has its words, rules and syntax. The basic elements of assembly language are:

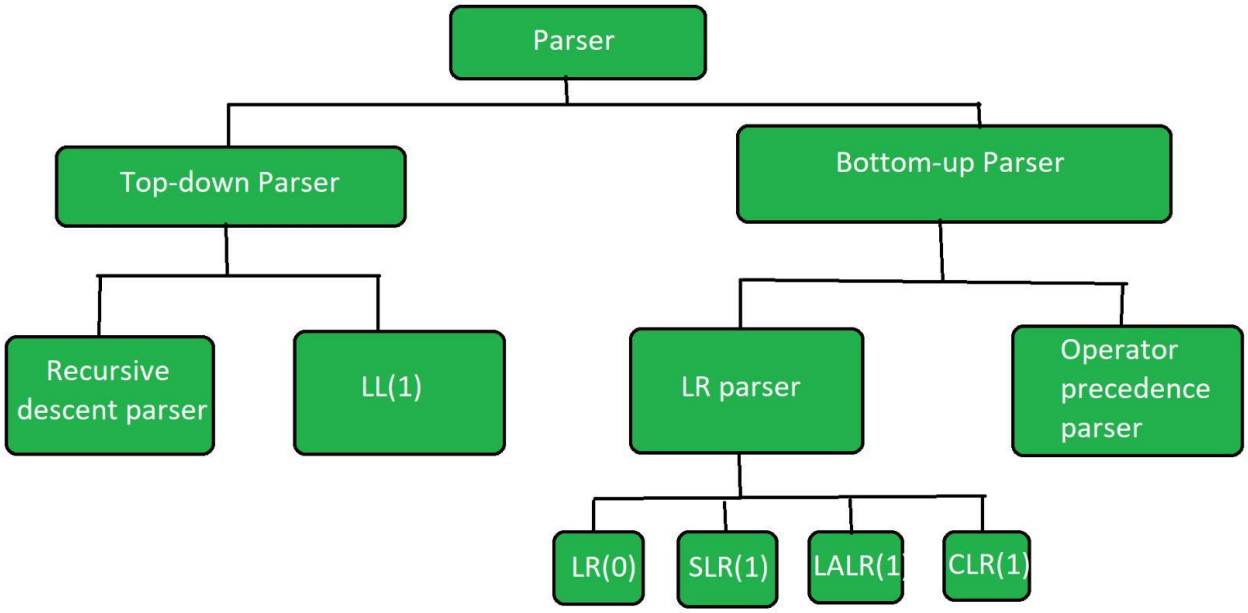
- Labels;
- Orders;
- Directives; and
- Comments.

PARAMETERS	COMPILER	INTERPRETER	ASSEMBLER
Conversion	It converts the high-defined programming language into Machine language or binary code.	It also converts the program-developed code into machine language or binary code.	It converts programs written in the assembly language to the machine language or binary code.
Scanning	It scans the entire program before converting it into binary code.	It translates the program line by line to the equivalent machine code.	It converts the source code into the object code then converts it into the machine code.
Error Detection	Gives the full error report after the whole scan.	Detects error line by line. And stops scanning until the error in the previous line is solved.	It detects errors in the first phase, after fixation the second phase starts.
Code generation	Intermediate code generation is done in the case of Compiler.	There is no intermediate code generation.	There is an intermediate object code generation.
Execution time	It takes less execution time comparing to an interpreter.	An interpreter takes more execution time than the compiler.	It takes more time than the compiler.
Examples	C, C#, Java, C++	Python, Perl, VB, PostScript, LISP, etc...	GAS, GNU
			https://ipwithease.com

7. Define Parsing. What is Shift reduce parsing. Check acceptability of the following string w.r.t given grammar.

String- abbcde

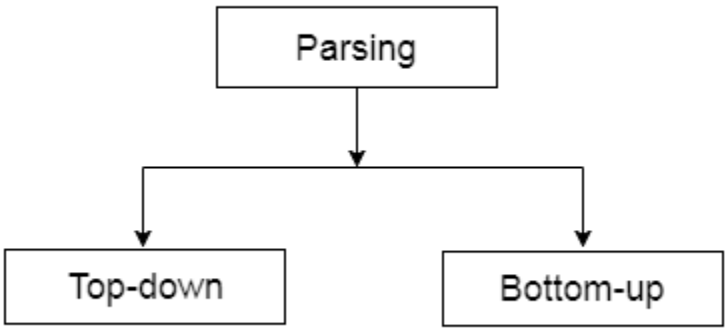
Grammar- S->aABe A->Abc/b B->d



Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase.

A parser takes input in the form of sequence of tokens and produces output in the form of parse tree.

Parsing is of two types: top-down parsing and bottom-up parsing.



Bottom up parsing

- Construct a parse tree for an input string beginning at leaves and going towards root OR
- Reduce a string w of input to start symbol of grammar

Consider a grammar

S → aABe
A → Abc | b
B → d

And reduction of a string

a b b c d e
a A b c d e
a A d e
a A B e
S

The sentential forms happen to be a *right most derivation in the reverse order.*

S → a A B e
→ a A d e
→ a A b c d e
→ a b b c d e

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom-up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of the shift-reduce parser is the LR parser.
This parser requires some data structures i.e.

- An input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Example 1 – Consider the grammar

S –> S + S
S –> S * S
S –> id

Perform Shift Reduce parsing for input string "id + id + id".

Stack	Input Buffer	Parsing Action
$	id+id+id$	Shift
$id	+id+id$	Reduce S->id
$S	+id+id$	Shift
$S+	id+id$	Shift
$S+id	+id$	Reduce S->id
$S+S	+id$	Reduce S->S+S
$S	+id$	Shift
$S+	id$	Shift
$S+id	$	Reduce S->id
$S+S	$	Reduce S->S+S
$S	$	Accept

8. Discuss Linking in -

MS-DOS

MS-DOS Editor is a text editor developed by Microsoft and published as a bundled application with MS-DOS version 5.0 in June, 1991. It was included with later versions of MS-DOS and several versions of Windows.

A linker is special program that combines the object files, generated by compiler/assembler and other pieces of code to originate an executable file has .exe extension. In the object file, linker searches and append all libraries needed for execution of file.

Design of Linker

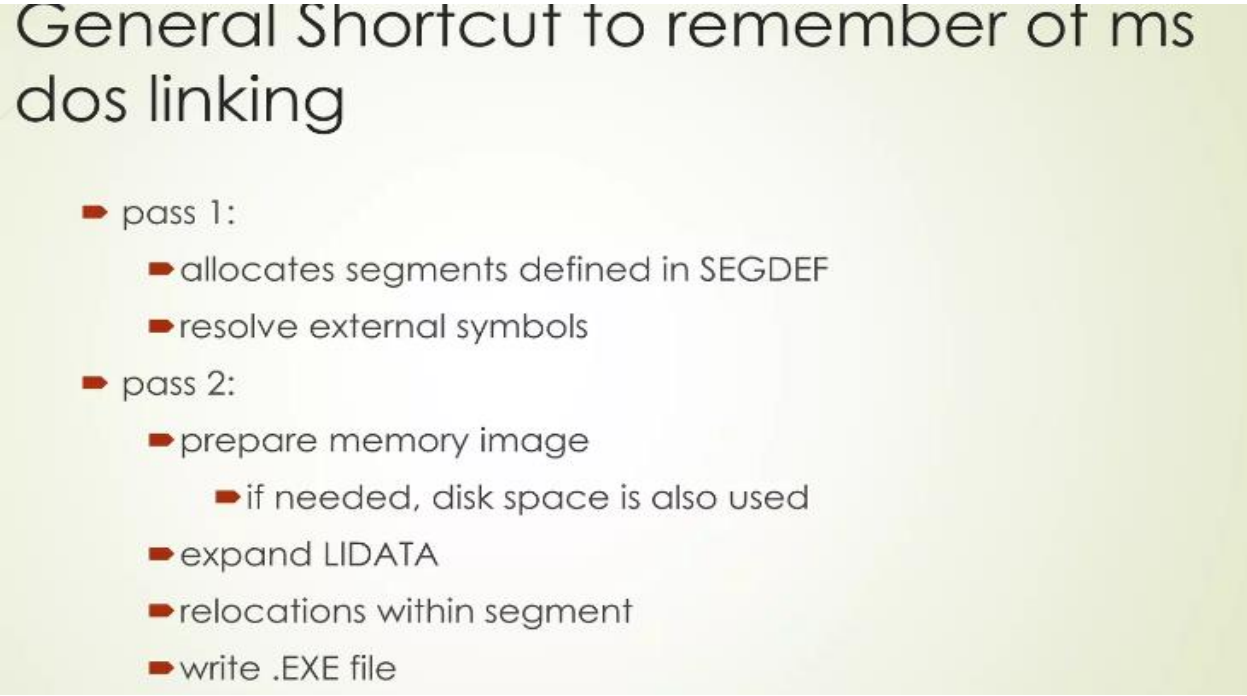
- For linking in MS Dos system we will design a program named LINKER which performs ***both linking and relocation of absolute segments and of re locatable segments that cannot be combined with other re locatable segments.***
- Its output is a binary program which resembles a program with .COM extension in MS DOS.
- This program is not relocated by the loader prior the execution .
- **(Note:- the difference between the LINKER and the LINK program of MS DOS :LINK produces a program with .EXE extension, which is relocated by the loader prior to execution)**

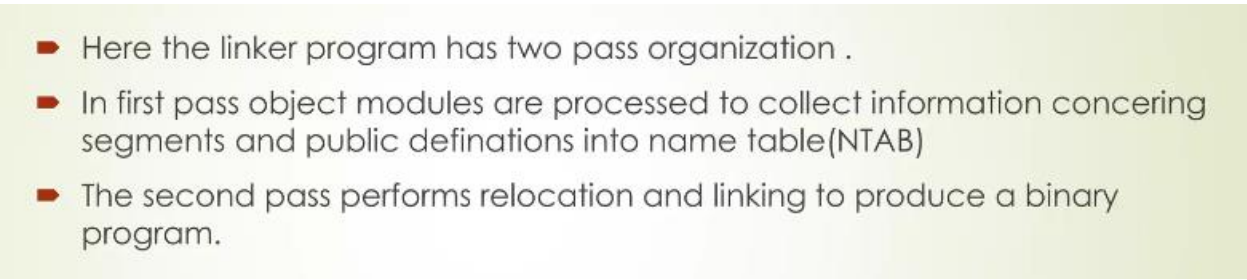
Specification

- The LINKER invocation command has the following format:
- LINKER *<object module names> ,<executable file>,*

  <load origin>, <list of library files>

- The linker performs relocation and linking of all named object modules to produce a binary program with the specified load origin.
- The program is stored in the file with the name <executable file>.If the LINKER comes across an external symbol that is not defined in any of the object modules named in th LINKER command, it locates ian object module in one of the library files included in <list of library files>that contains a public definitions.
- This method of resolving an external reference by automatically including an object module form a library file is called "***AUTOLINKING".***





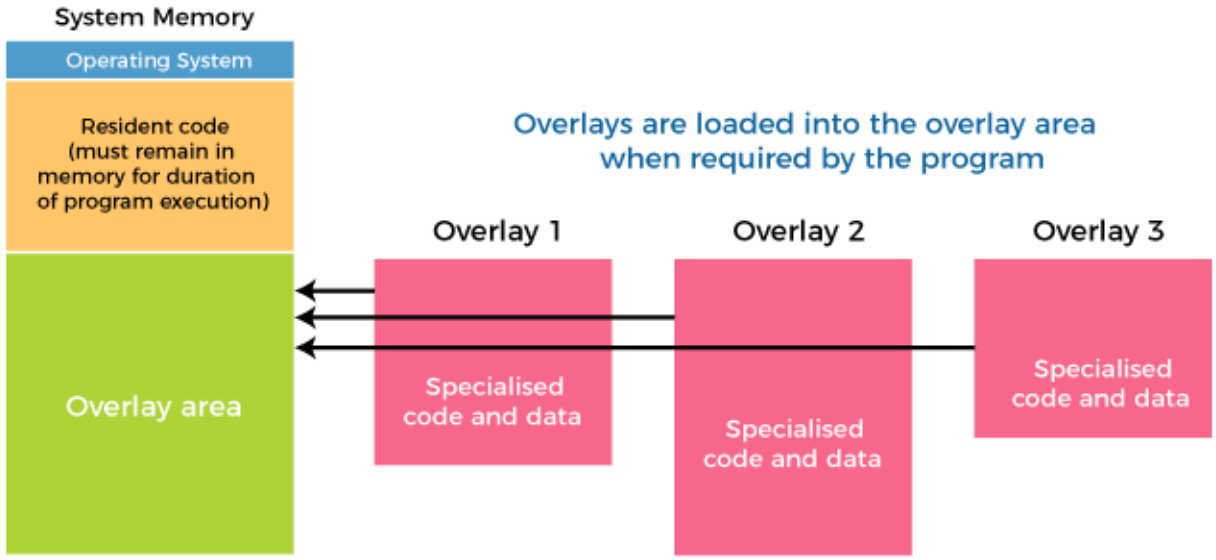
Overlay Structured programs

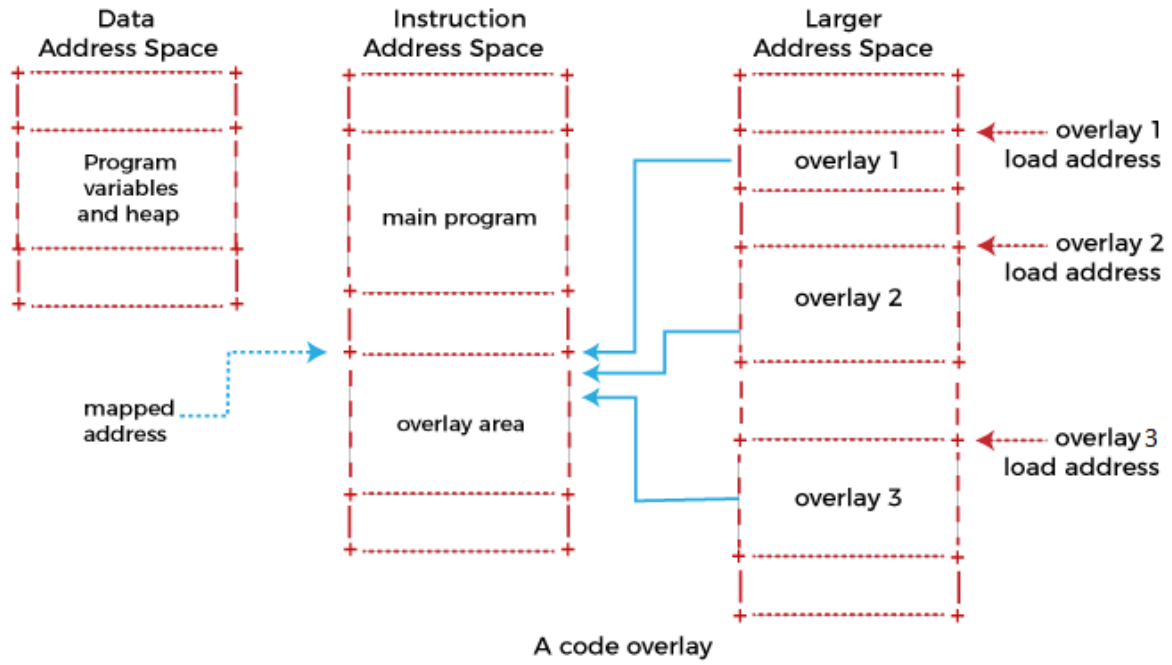
In an overlay structure, a region is a contiguous area of virtual storage within which segments can be loaded independently of paths in other regions. An overlay program can be designed in single or multiple regions.

The concept of *overlays* is that it will not use the complete program simultaneously whenever a process is running. It will use only some part of it. Then overlays concept says that whatever part you require, you load it, and once the part is done, you just unload it, which means pull it back and get the new part you required and run it.

Formally, "The process of *transferring a block* of program code or other data into internal memory, replacing what is already stored".

Sometimes it happens that compared to the size of the biggest partition, the size of the program will be even more. Then, in that case, you should go with overlays.





Advantages of Overlays

Overlays in memory management have the following advantages, such as:

- Reduce memory requirement.
- Reduce time requirement.

Disadvantages of Overlays

Overlays also have some disadvantages, such as:

- The programmer must specify the overlap map.
- Programmer must know memory requirements.
- The overlapped module must be completely disjoint.
- The programming design of the overlay structure is complex and not possible in all cases.

9. Why Editors are used. How will differentiate Multi window editor vs Screen Editor.

Editors or text editors are software programs that enable the user to create and edit text files. In the field of programming, the term editor usually refers to source code editors that include many special features for writing and editing code. Notepad, Wordpad are some of the common editors used on Windows OS and vi, emacs, Jed, pico are the editors on UNIX OS. Features normally associated with text editors are — moving the cursor, deleting, replacing, pasting, finding, finding and replacing, saving etc.

Multiple Window Editor: Multiple window editor allows you to work on more than one file at a time and cut and paste text from file into another via yanking and putting. The two fundamental concepts that lie behind multi-window editors are buffer and windows.

Buffer holds the text to be edited. The text may come from a file or a brand new text that you want to write on a file. A file only has one buffer associated with it. Windows provides a view to the buffer to see what the buffer holds and edit and modify it. A buffer may have multiple windows.

Screen editors: In this type of editors, the user is able to see the cursor on the screen and can make a copy, cut, paste operation easily. It is very easy to use mouse pointer.

The Screen editor tool is a part of the Image editing mode. Here, you can specify the visible part of the screenshot (i.e. to crop the picture). This is extremely helpful when you want to crop a large image and display only a certain part of it.

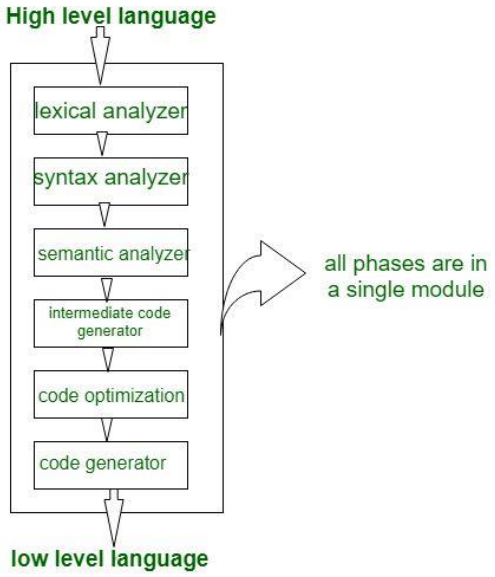
10. Describe Multi pass Assembler. List all the variants of Assembler.

The Assembler is a Software that converts an assembly language code to machine code. It takes basic Computer commands and converts them into Binary Code that Computer's Processor can use to perform its Basic Operations. An assembly language is a low-level language. It gives instructions to the processors for different tasks. It is specific for any processor.

An opcode basically gives information about the particular instruction. The symbolic representation of the opcode (machine level instruction) is called mnemonics. Programmers use them to remember the operations in assembly language.

1. One-Pass Assembler

These assemblers perform the whole conversion of assembly code to machine code in one go.



2. Multi-Pass/Two-Pass Assembler

These assemblers first process the assembly code and store values in the opcode table and symbol table. And then in the second step, they generate the machine code using these tables.

a) Pass 1

Symbol table and opcode tables are defined.

keep the record of the location counter.

Also, processes the pseudo instructions.

b) Pass 2

Finally, converts the opcode into the corresponding numeric opcode.

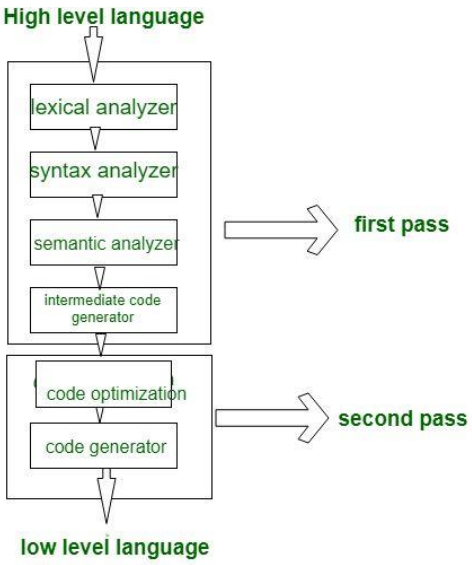
Generates machine code according to values of literals and symbols.

Some Important Terms

Opcode table: They store the value of mnemonics and their corresponding numeric values.

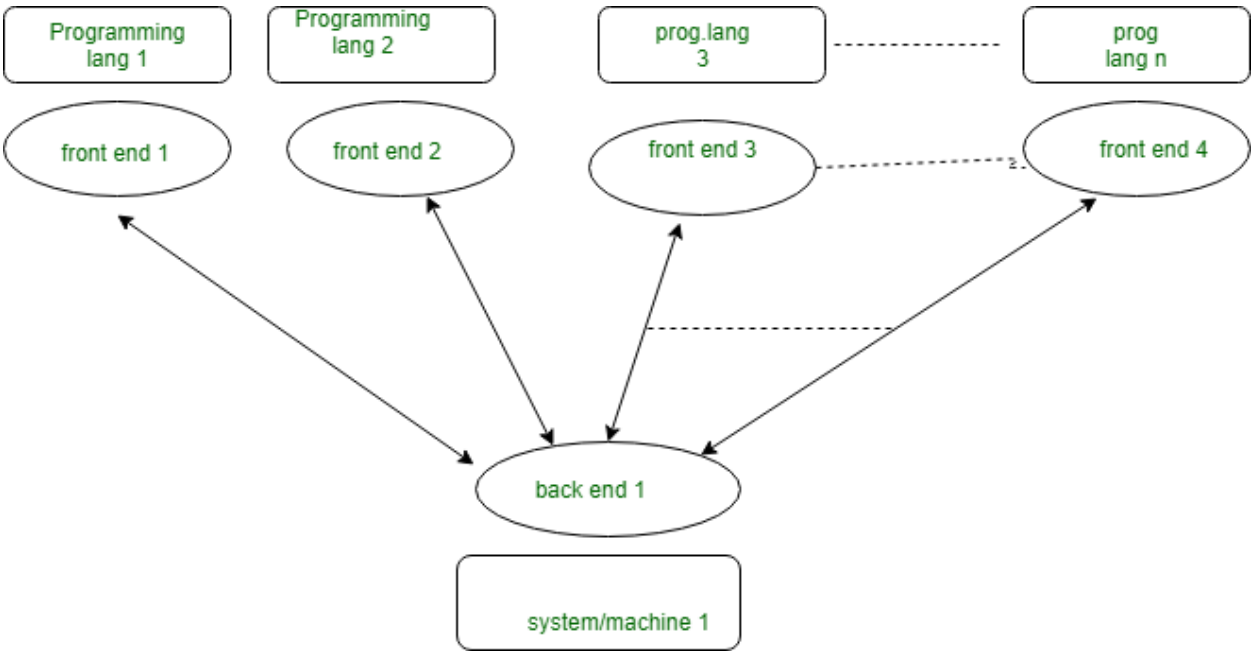
Symbol table: They store the value of programming language symbols used by the programmer, and their corresponding numeric values.

Location Counter: It stores the address of the location where the current instruction will be stored.

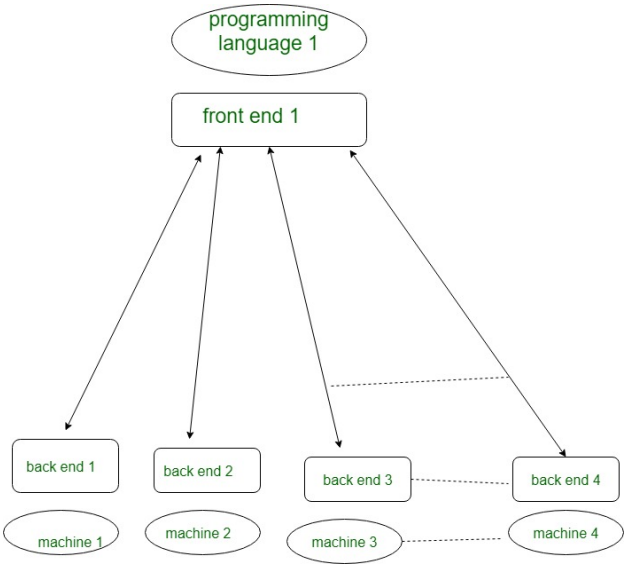


With multi-pass Compiler we can solve these 2 basic problems:

If we want to design a compiler for different programming language for same machine. In this case for each programming language there is requirement of making Front end/first pass for each of them and only one Back end/second pass as:



If we want to design a compiler for same programming language for different machine/system. In this case we make different Back end for different Machine/system and make only one Front end for same programming language as:

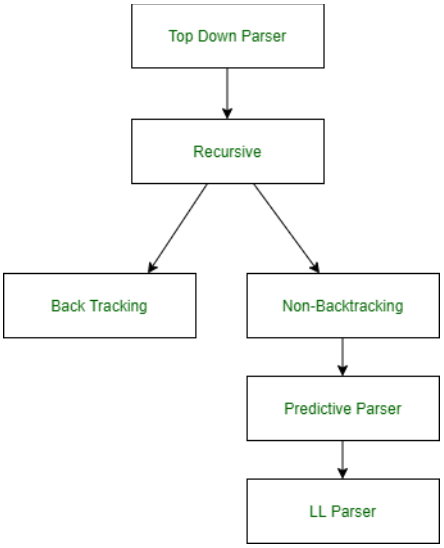


Differences between Single Pass and Multipass Compilers:

Parameters	Single pass	multi Pass
Speed	Fast	Slow

Memory	More	Less
Time	Less	More
Portability	No	Yes

11. Implement Predictive Parser with an example. Which grammar is used in Top Don Parsing.



Predictive Parser :
A predictive parser is a recursive descent parser with no backtracking or backup. It is a top-down parser that does not require backtracking. At each step, the choice of the rule to be expanded is made upon the next terminal symbol.
Consider

A -> A1 | A2 | ... | An

If the non-terminal is to be further expanded to ‘A’, the rule is selected based on the current input symbol ‘a’ only.

Predictive Parser Algorithm :

Make a transition diagram(DFA/NFA) for every rule of grammar.

Optimize the DFA by reducing the number of states, yielding the final transition diagram.

Simulate the string on the transition diagram to parse a string.

If the transition diagram reaches an accept state after the input is consumed, it is parsed.

Consider the following grammar –

E->E+T|T

T->T*F|F

F->(E)|id

After removing left recursion, left factoring

E->TT'

T'->+TT'|ε

T->FT''

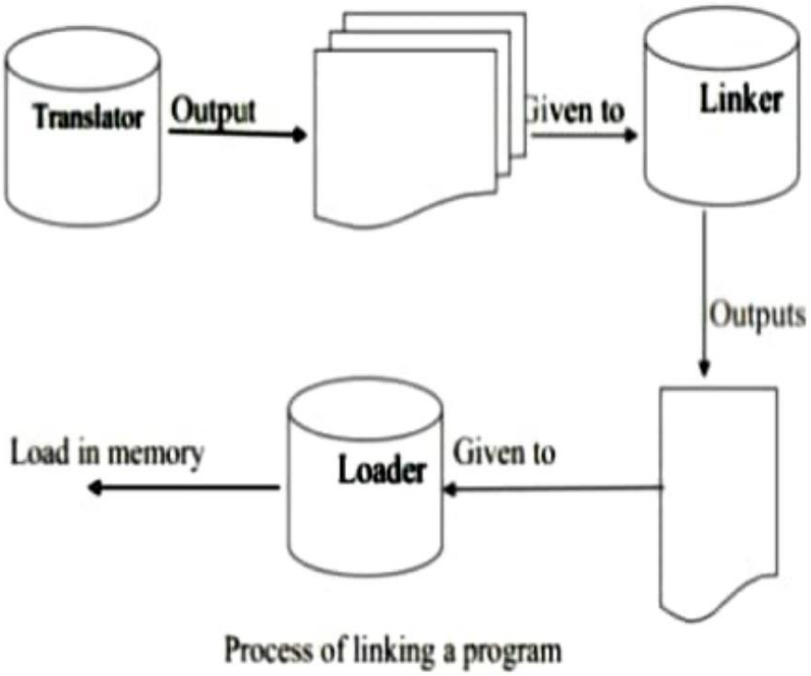
T''->*FT''|ε

F->(E)|id

12. Write short notes on following

Linking Loader

A loader which combines the functions of a relocating loader with the ability to combine a number of program segments that have been independently compiled into an executable program. Loader is the system program which is responsible for preparing the object program for execution and initiate the execution. The loader does the job of coordinating with the OS to get initial loading address for the .EXE file and load it into the memory.



Process of linking a program

DIRECT-LINKING LOADERS

- A direct-linking loader is a relocatable loader.
- It has advantage of allowing programmer multiple procedure segments and multiple data segments.
- Complete freedom in referencing data or instructions contained in other segments, provides flexible intersegment referencing.

• Function of Loader:

Allocation:

Allocates the space in the memory where the object program would be loaded for Execution.

Linking:

It links two or more object codes and provides the information needed to allow references between them.

Loading:

It brings the object program into the memory for execution.

Dynamic Linking Loader is a general re-locatable loader

• Allowing the programmer multiple procedure segments and multiple data segments and giving programmer complete freedom in referencing data or instruction contained in other segments.

• The assembler must give the loader the following information with each procedure or data segment.

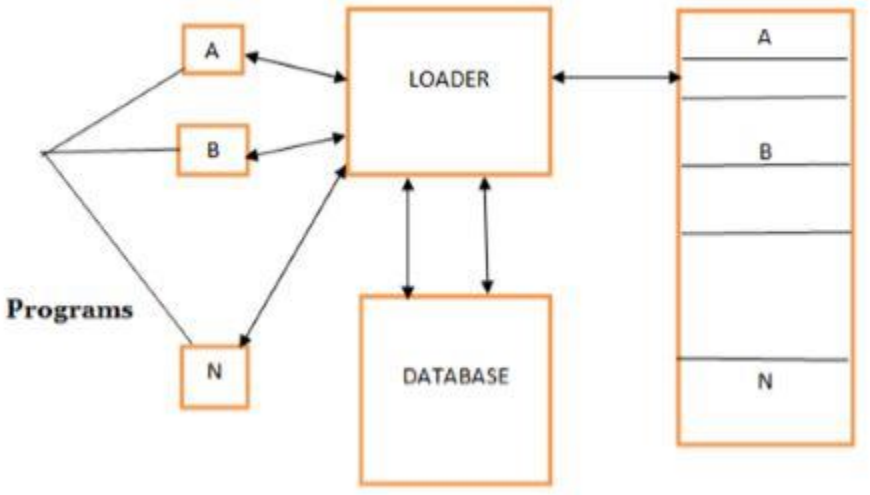
• Dynamic linking defers much of the linking process until a program starts running. It provides a variety of benefits that are hard to get otherwise.

• Dynamically linked shared libraries are easier to create than static linked shared libraries.

• Dynamically linked shared libraries are easier to update than static linked shared libraries.

• The semantics of dynamically linked shared libraries can be much closer to those of unshared libraries.

• Dynamic linking permits a program to load and unload routines at runtime, a facility that can otherwise be very difficult to provide.



General Loader scheme

In this loader scheme, the source program is converted to object program by some translator (assembler). The loader accepts these object modules and puts the machine instruction and data in an executable form at their assigned memory. The loader occupies the same portion of main memory.

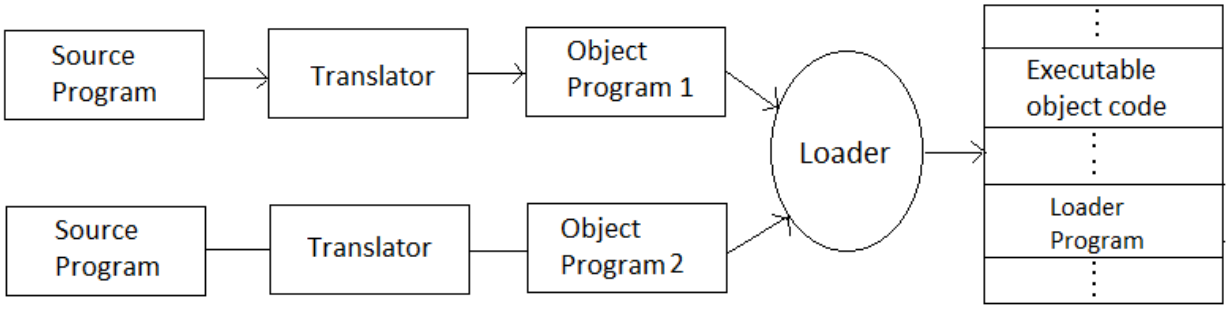


Fig. General Loader Scheme

Advantages:

The program need not be retranslated each time while running it. Thus us because initially when source program gets executed and object program gets generated. If a program is not modified, then the loader can make use of this object program to convert it to executable form.

There is no wastage of memory because the assembler is not placed in the memory instead of it, loader occupies some portion of the memory. And the size of the loader is smaller than assembler so more memory is available to the user.

It is possible to write source program with multiple programs and multiple languages because the source program is first converted to an object program always and loader accepts these object modules to convert it to an executable format.

Types of Loaders

- **C**ompile-Go Loader
- **G**eneral Loader
- **A**bsolute Loader
- **R**elocating Loader
- **P**ractical Relocating Loader
- **L**inking Loader

1. Briefly define functions of Macro Assembler

An assembler that brings high-level language features to assembly language programming. It translates a single multi-argument source line o code into a sequence of machine instructions.

A macro assembler is able to generate a program segment, which is defined by a macro, when the name of the macro appears as an opcode in a program. The macro assembler is still capable of regular assembler functioning, generating a machine instruction for each line of assembly language code; but like a compiler, it can generate many machine instructions from one line of source code. Its instruction set can be expanded to include new mnemonics, which generate these program segments of machine code.

Recognize macro definitions

Save the macro definition

Recognize macro calls

Expand macro calls

- The fundamental functions common to all macro processors are:
- Macro Definition
- Macro Invocation
- Macro Expansion

2. Define the term Ambiguity w.r.t Parsing

A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivative or more than one parse tree for the given input string. If the grammar is not ambiguous then it is called unambiguous.

Check whether the given grammar G is ambiguous or not.

1. E → E + E
2. E → E - E
3. E → id

Solution:

From the above grammar String "id + id - id" can be derived in 2 ways:

First Leftmost derivation

1. E → E + E

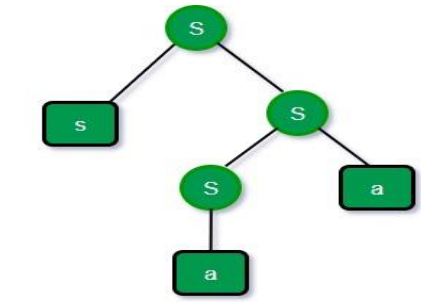
2. → id + E
3. → id + E - E
4. → id + id - E
5. → id + id- id

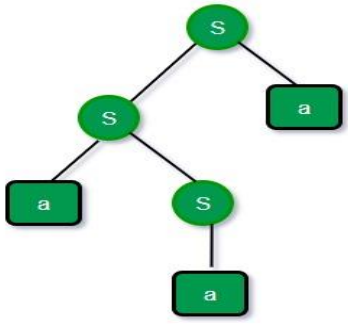
Second Leftmost derivation

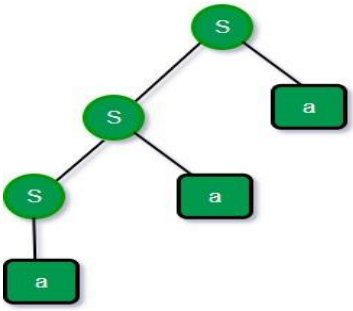
1. E → E - E
2. → E + E - E
3. → id + E - E
4. → id + id - E
5. → id + id - id

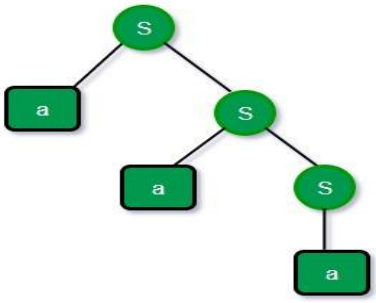
Since there are two leftmost derivation for a single string "id + id - id", the grammar G is ambiguous.

A grammar that produces more than one parse tree for some sentence is said to be ambiguous.
Eg- consider a grammar
S -> aS | Sa | a
Now for string aaa, we will have 4 parse trees, hence ambiguous



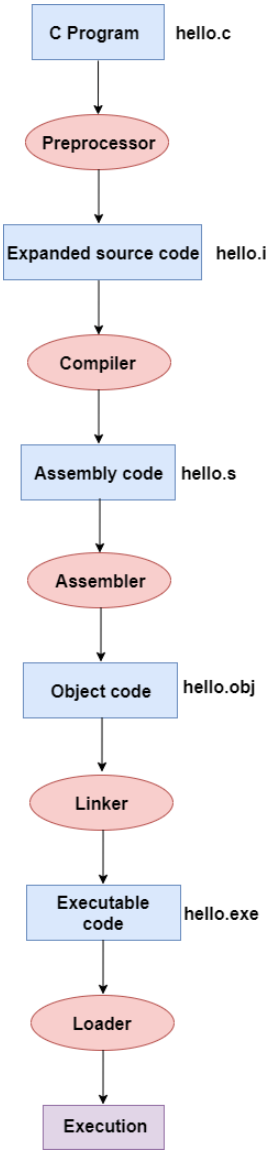


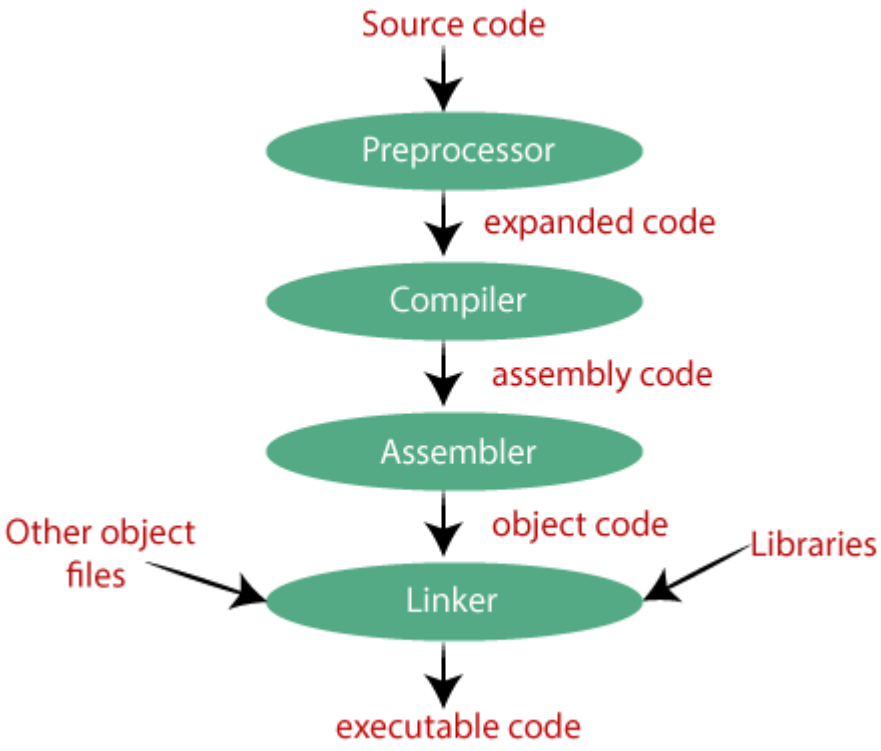




3. List various program extensions during program writing, translation & Execution.

File extensions indicate the file format or file type. It is start dot (.) symbol and filename. File extensions consist of three or four characters, although in rare cases it could be two. They are typically letters or digits. Such as .txt, .mp3, .pptx etc. This file extension helps the operating system to know what type of file or program to run when you double-click on it. Here list of computer file extension and their meaning.

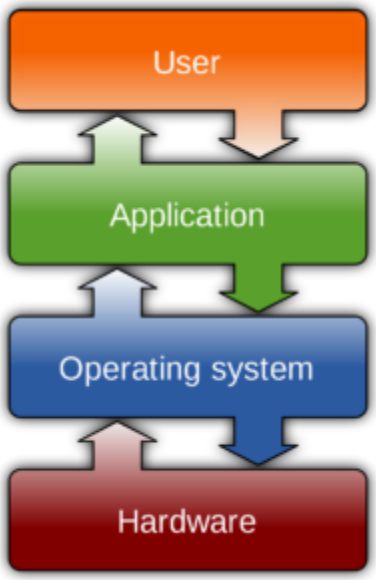




4. Justify the need of Operating System in program Execution.

The Need for Operating System: Operating System is a program that acts as an Interface between the system hardware and the user making the tasks easier. It is important software which runs on a computer and controls the set of instructions and wisely utilizes each part of the computer.

All the working of a computer system depends on the OS at the base level. Further, it performs all the functions like handling memory, processes, the interaction between hardware and software, etc. Now, let us study the need for the operating system.



- Interface between the user and the computer
- Booting
- Managing the input/output devices
- Multitasking
- Platform for other application software
- Manages the memory
- Manages the system files
- Provides Security
- Acts as an Interface

5. List features of MS-DOS Editor.

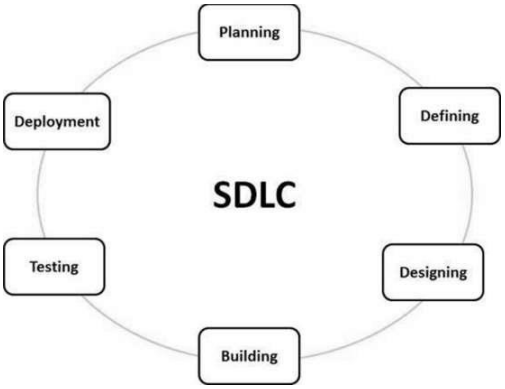
A DOS editor is a text editor that is included with DOS, which manages text-based files within a computer file system.

- MS-DOS Editor uses a text user interface and its color scheme can be adjusted.
- It has a multiple-document interface in which its version 2.0 (as included in DOS 7 or Windows 9x) can open up to 9 files at a time while earlier versions (included in DOS 5 and 6) are limited to only one file.
- The screen can be split vertically into two panes which can be used to view two files simultaneously or different parts of the same file.
- It can also open files in binary mode, where a fixed number of characters are displayed per line, with newlines treated like any other character.

- This mode shows characters as hexadecimal characters (0-9 and A-F). Editor converts Unix newlines to DOS newlines and has mouse support. Some of these features were added only in version 2.0.

6. Discuss System Software development with all steps.

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.



Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Stage 5: Testing the Product

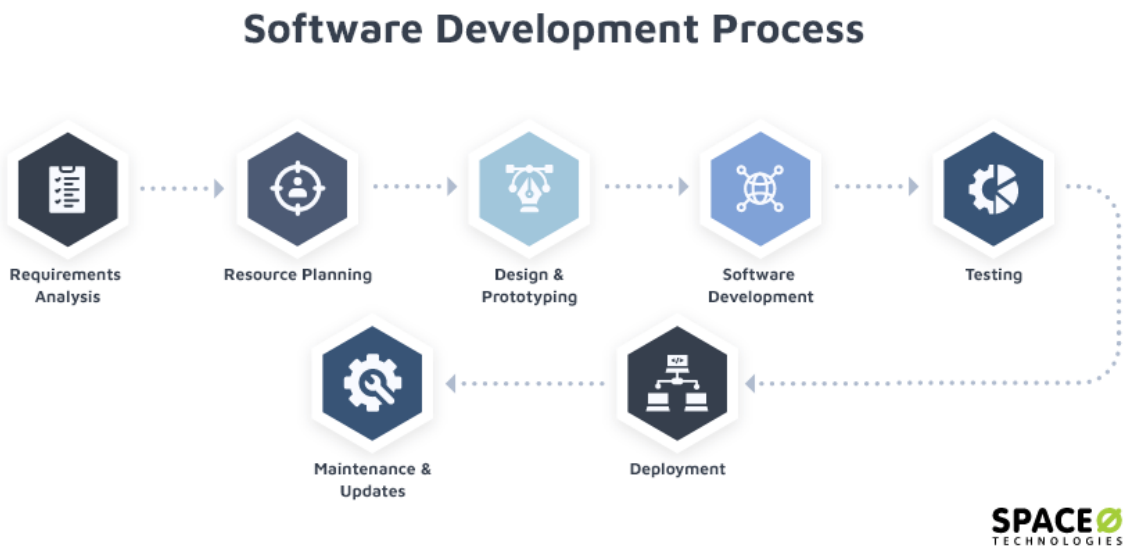
This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model



7. Briefly explain all tuples of DFA. Make a transition diagram of the regular grammar given as
String ended with 101

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

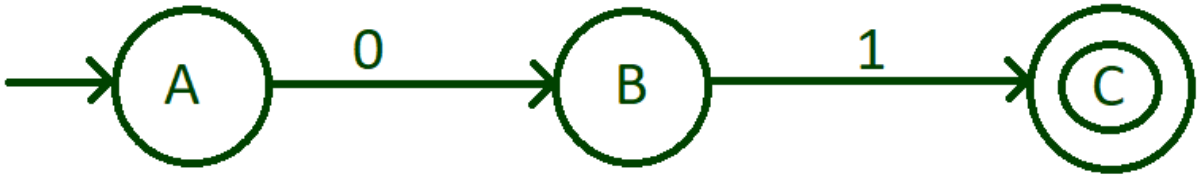
A DFA is a collection of 5-tuples same as we described in the definition of FA.

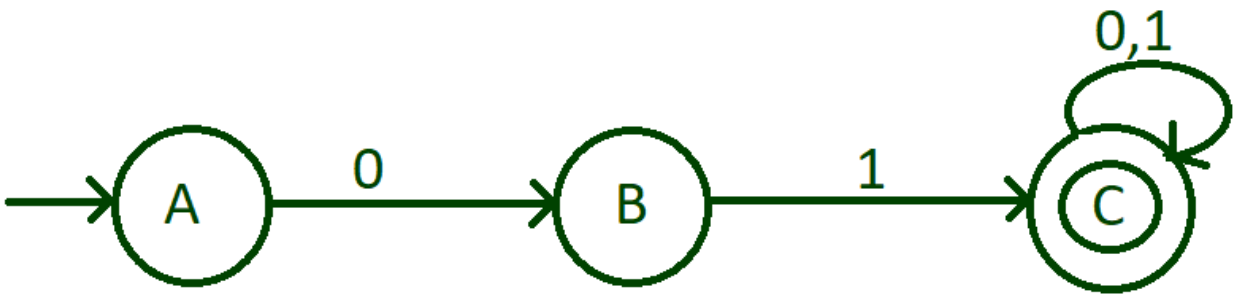
1. Q: finite set of states

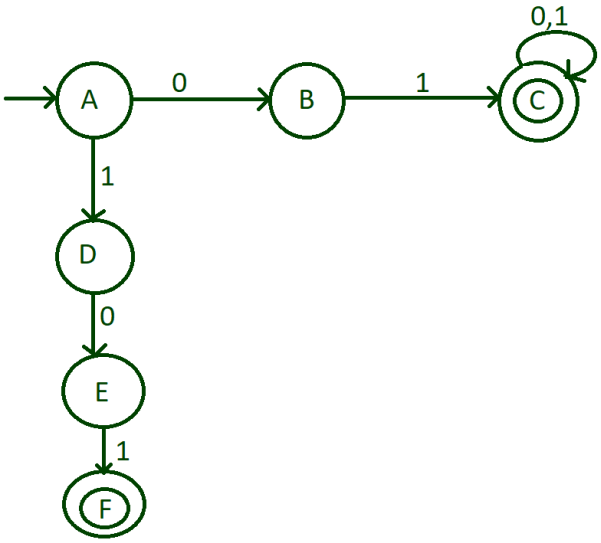
2. ∑: finite set of the input symbol
3. q0: initial state
4. F: final state
5. δ: Transition function

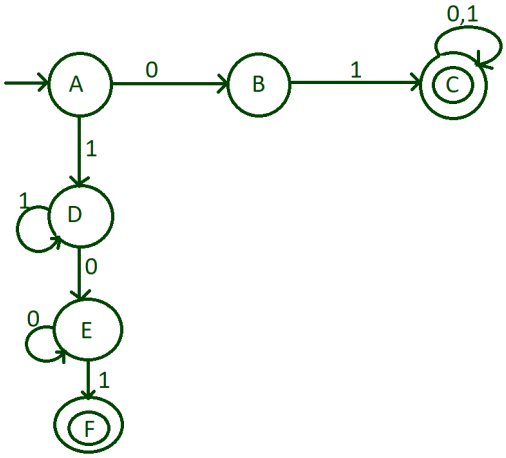
Transition function can be defined as:

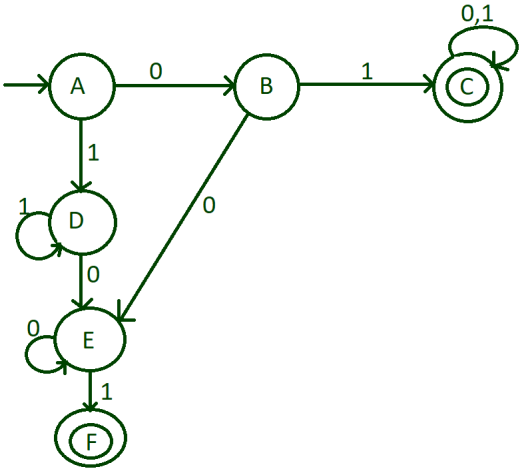
1. δ: Q x ∑→Q

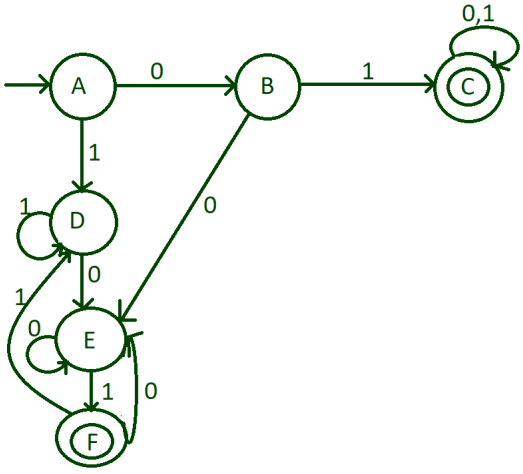








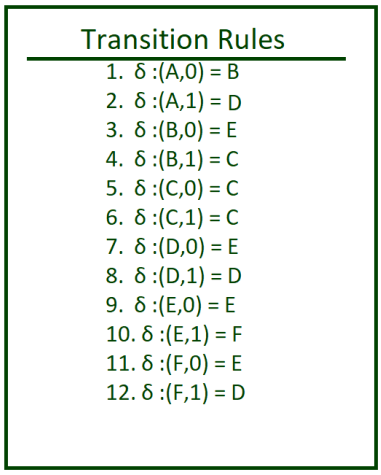




Transition table and Transition rules of the above DFA:

State	Input (0)	Input (1)
—>A	B	D

State	Input (0)	Input (1)
B	E	C
C*	C	C
D	E	D
E	E	F
F*	E	D



8. Define YACC. Implement with an example.

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR(1) (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

- YACC provides a tool to produce a parser for a given grammar.
- YACC is a program designed to compile a LALR (1) grammar.
- It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
- The input of YACC is the rule or grammar and the output is a C program.

These are some points about YACC:

Input: A CFG- file.y

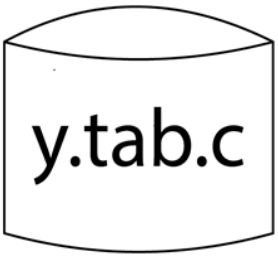
Output: A parser y.tab.c (yacc)

- The output file "file.output" contains the parsing tables.
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.

The basic operational sequence is as follows:

This file contains the desired grammar in YACC format.

It shows the YACC program.



It is the c source program created by YACC.

C Compiler

Executable file that will parse grammar given in gram.Y

9. How Memory management & file Management taken place in Operating System.

In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is required:

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.

- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

File management is defined as the process of manipulating files in computer system, it management includes the process of creating, modifying and deleting the files.

The following are some of the tasks performed by file management of operating system of any computer system:

1. It helps to create new files in computer system and placing them at the specific locations.
2. It helps in easily and quickly locating these files in computer system.
3. It makes the process of sharing of the files among different users very easy and user friendly.
4. It helps to stores the files in separate folders known as directories. These directories help users to search file quickly or to manage the files according to their types or uses.
5. It helps the user to modify the data of files or to modify the name of the file in the directories.

- File Attributes
  It specifies the characteristics of the files such as type, date of last modification, size, location on disk etc. file attributes help the user to understand the value and location of files. File attributes is one most important feature. It is uses to describe all the information regarding particular file.
- File Operations
  It specifies the task that can be performed on a file such as opening and closing of file.
- File Access permission
  It specifies the access permissions related to a file such as read and write.

- File Systems
  It specifies the logical method of file storage in a computer system. Some of the commonly used files systems include FAT and NTFS.

10. Explain following terms w.r.t. Assembler.

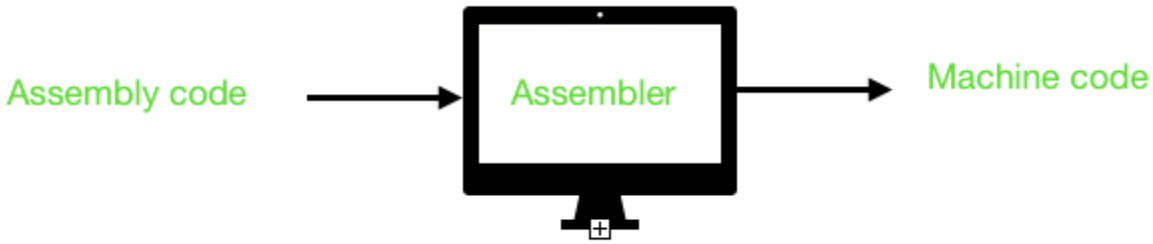
- **Design criteria of Assembler**
- **Assembler for Intel x86**

(a)

- Minimising part count.
- Modularity.
- Built-in fasteners.
- Part symmetry.
- Mistake-proofing.
- Use of standard parts.

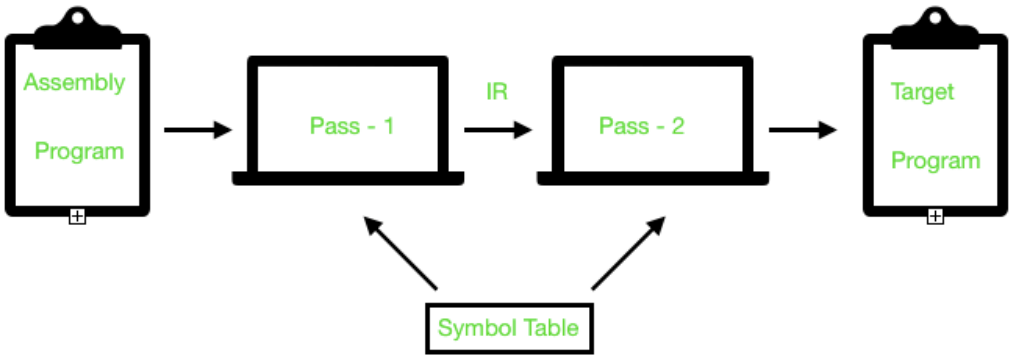
- Use of reasonable tolerances.

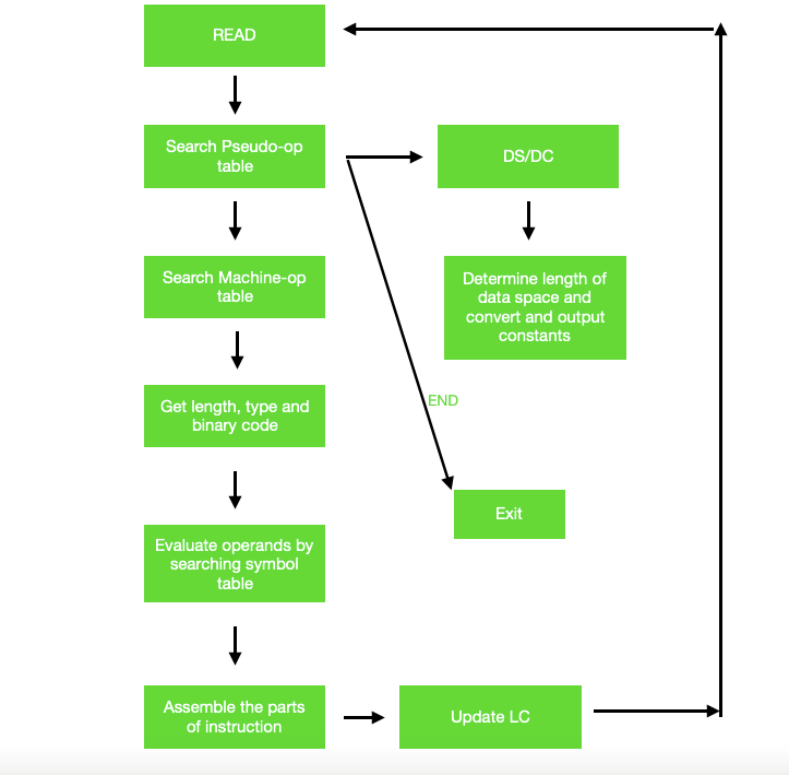
Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.



It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- Pass-1:
    1. Define symbols and literals and remember them in symbol table and literal table respectively.
    2. Keep track of location counter
    3. Process pseudo-operations
- Pass-2:
    1. Generate object code by converting symbolic op-code into respective numeric op-code
    2. Generate data for literals and look for values of symbols

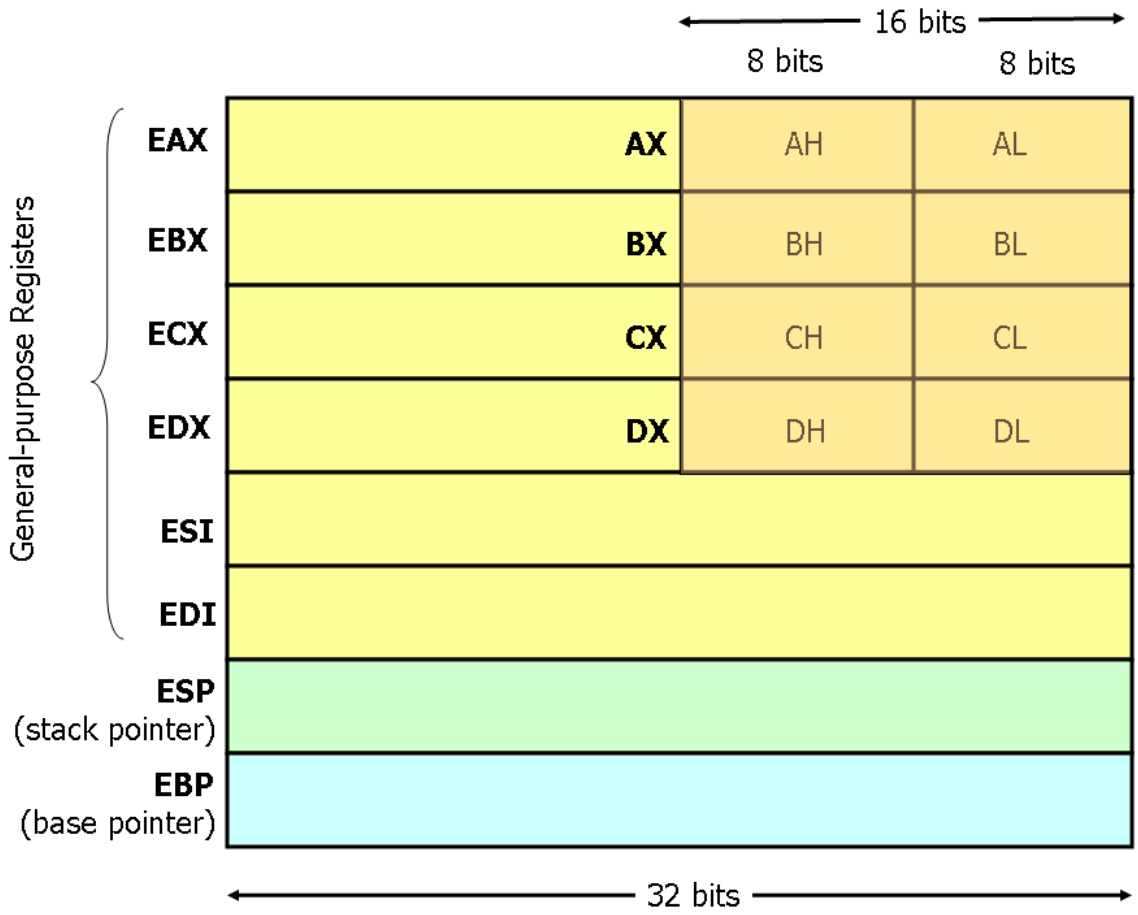




(b)

This guide describes the basics of 32-bit x86 assembly language programming, covering a small but useful subset of the available instructions and assembler directives. There are several different assembly languages for generating x86 machine code. The one we will use in CS216 is the Microsoft Macro Assembler (MASM) assembler. MASM uses the standard Intel syntax for writing x86 assembly code.

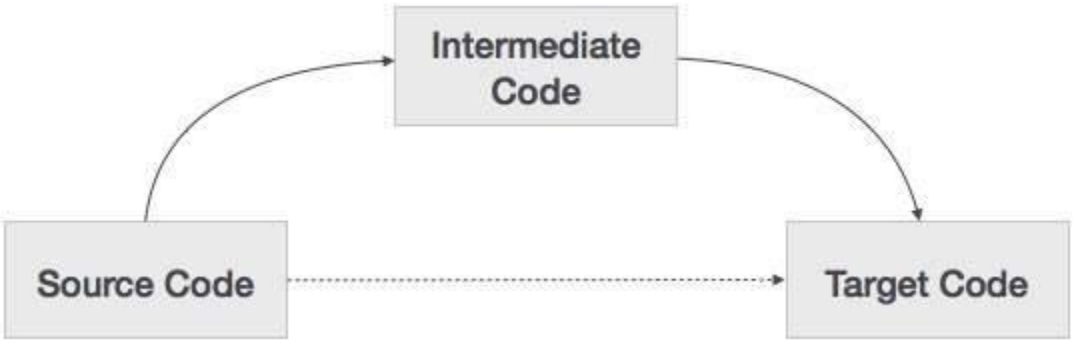
The full x86 instruction set is large and complex (Intel's x86 instruction set manuals comprise over 2900 pages), and we do not cover it all in this guide. For example, there is a 16-bit subset of the x86 instruction set. Using the 16-bit programming model can be quite complex. It has a segmented memory model, more restrictions on register usage, and so on. In this guide, we will limit our attention to more modern aspects of x86 programming, and delve into the instruction set only in enough detail to get a basic feel for x86 programming.



This tool takes x86 or x64 assembly instructions and converts them to their binary representation (machine code). It can also go the other way, taking a hexadecimal string of machine code and transforming it into a human-readable representation of the instructions. It uses GCC and obj dump behind the scenes.

11. Highlight the importance of Intermediate code representation. Show Three Address code, Quadruple & triples representation with a suitable example.

A source code can directly be translated into its target machine code, then why at all we need to translate the source code into an intermediate code which is then translated to its target code? Let us see the reasons why we need an intermediate code.



- If a compiler translates the source language to its target machine language without having the option for generating intermediate code, then for each new machine, a full native compiler is required.
- Intermediate code eliminates the need of a new full compiler for every unique machine by keeping the analysis portion same for all the compilers.
- The second part of compiler, synthesis, is changed according to the target machine.
- It becomes easier to apply the source code modifications to improve code performance by applying code optimization techniques on the intermediate code.

Intermediate codes can be represented in a variety of ways and they have their own benefits.

- High Level IR - High-level intermediate code representation is very close to the source language itself. They can be easily generated from the source code and we can easily apply code modifications to enhance performance. But for target machine optimization, it is less preferred.
- Low Level IR - This one is close to the target machine, which makes it suitable for register and memory allocation, instruction set selection, etc. It is good for machine-dependent optimizations.

Intermediate code can be either language specific (e.g., Byte Code for Java) or language independent (three-address code).

Three-Address Code

Intermediate code generator receives input from its predecessor phase, semantic analyzer, in the form of an annotated syntax tree. That syntax tree then can be converted into a linear representation, e.g., postfix notation. Intermediate code tends to be machine independent code. Therefore, code generator assumes to have unlimited number of memory storage (register) to generate code.

For example:

```
a = b + c * d;
```

The intermediate code generator will try to divide this expression into sub-expressions and then generate the corresponding code.

```
r1 = c * d;
r2 = b + r1;
a = r2
```

r being used as registers in the target program.

A three-address code has at most three address locations to calculate the expression. A three-address code can be represented in two forms : quadruples and triples.

Quadruples

Each instruction in quadruples presentation is divided into four fields: operator, arg1, arg2, and result. The above example is represented below in quadruples format:

Op	arg_1	arg_2	result
*	c	d	r1
+	b	r1	r2
+	r2	r1	r3
=	r3		a

Triples

Each instruction in triples presentation has three fields : op, arg1, and arg2.The results of respective sub-expressions are denoted by the position of expression. Triples represent similarity with DAG and syntax tree. They are equivalent to DAG while representing expressions.

Op	arg_1	arg_2
*	c	d
+	b	(0)
+	(1)	(0)
=	(2)	

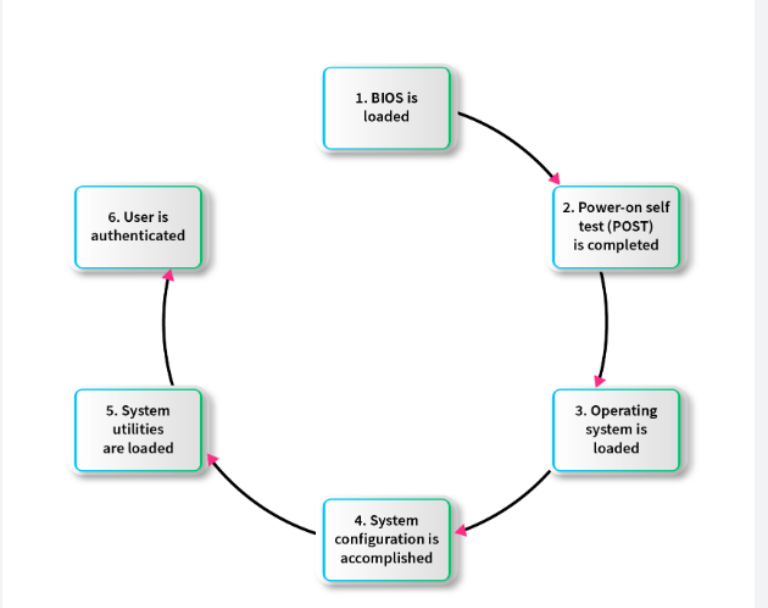
Triples face the problem of code immovability while optimization, as the results are positional and changing the order or position of an expression may cause problems.

Indirect Triples

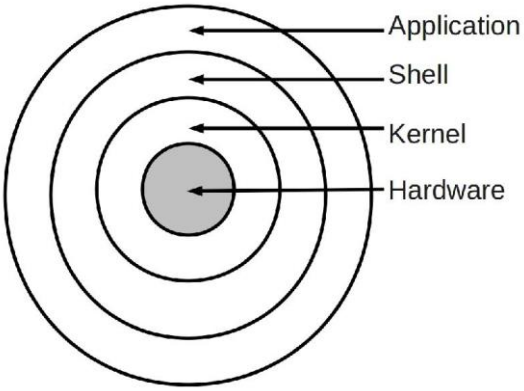
This representation is an enhancement over triples representation. It uses pointers instead of position to store results. This enables the optimizers to freely re-position the sub-expression to produce an optimized code.

12. How Booting is performed in Operating System. Explain Design of Shell & kernel in Operating System.

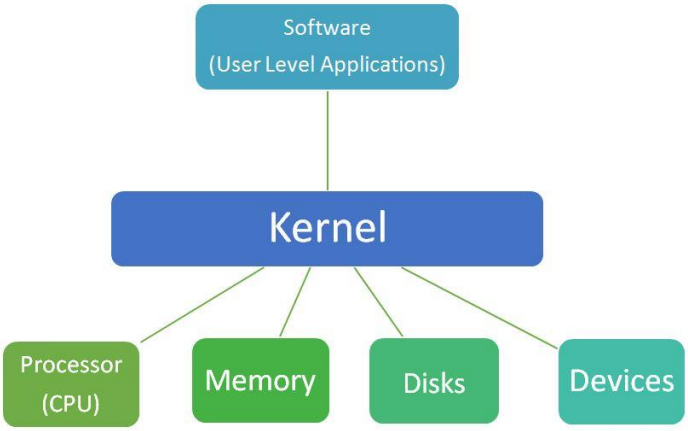
In computing, booting is the process of starting a computer as initiated via hardware such as a button or by a software command. After it is switched on, a computer's central processing unit (CPU) has no software in its main memory, so some process must load software into memory before it can be executed.



A shell is an interface between a user and the operating system. It lets us give commands to the system and start other programs. Your task is to program a simple shell similar to for example Bash, which probably is the command shell you normally use when you use a Unix/Linux system.



Kernel is central component of an operating system that manages operations of computer and hardware. It basically manages operations of memory and CPU time. It is core component of an operating system. Kernel acts as a bridge between applications and data processing performed at hardware level using inter-process communication and system calls.



Types of Kernels

1. Monolithic Kernel (Unix)

It is one of types of kernels where all operating system services operate in kernel space. It has dependencies between systems components. It has huge lines of code which is complex.

2. Micro Kernel (Mach)

It is kernel types which has minimalist approach. It has virtual memory and thread scheduling. It is more stable with less services in kernel space. It puts rest in user space.

3. Hybrid Kernel (Windows NT)

It is the combination of both monolithic kernel and microkernel. It has speed and design of monolithic kernel and modularity and stability of microkernel.

4. Exo Kernel (Nemesis)

It is the type of kernel which follows end-to-end principle. It has fewest hardware abstractions as possible. It allocates physical resources to applications.