Q.2 a. Define process. What are the various states of a process? Also discuss the security problems in an OS.

Answer:

A process or task is a portion of a program in some stage of execution. A program can consist of several processes, each working on their own or as a unit, perhaps communicating with each other. Each process that runs in an operating system is assigned a process control block that holds information about the process, such as a unique process ID (a number used to identify the process), the saved state of the process, the process priority and where it is located in memory.

A process in a computer system may be in one of the possible state as follows:

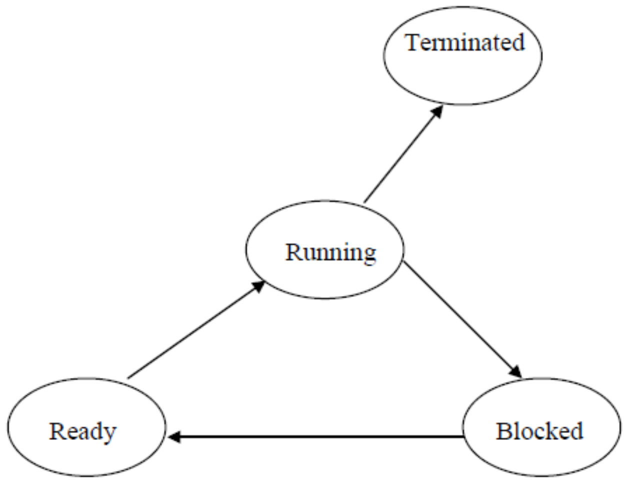
• Running: A CPU is currently allocated to the process and the process is in execution.

•Blocked: The process is waiting for a request to be satisfied, or an event to occur. Such a process cannot execute even if a CPU is available.

• Ready: The process is not running, however it can execute if a CPU is allocated to it, means the process is not blocked.

• Terminated: The process has finished its execution.

A state transition is caused by the occurrence of some event in the system. When a process in the running state makes an I/O request, it has to enter blocked state awaiting completion of the I/O. When the I/O completes, the process state changes from blocked to ready. Similar state changes occur when a process makes some request, which cannot be satisfied by OS straightway. The process state changes to blocked until the request is satisfied, when its state changes to ready once again. A ready process becomes running when the CPU allocated to it. The fundamental state transition is shown in figure below.



problems in an OS :Security must consider external environment of the system, and protect it from:

✦unauthorized access.

✦malicious modification or destruction
✦accidental introduction of inconsistency.

Protecting data against unauthorised use is a major concern for operating systems designers and administrators.
– Data confidentiality – prevent unauthorised access
– Data integrity – prevent unauthorised tampering
– System availability – prevent denial of service

• Issues to consider (at design stage)
– Casual users browsing information that's not for them
– Snooping by skilled insiders
– Organised and determined a
ttempts to make money
– Espionage etc.

• System "bloat" is definitely bad for security
• Accidental data loss probably more common than determined
security evasion

.

b. Compare and contrast Multiprogramming, Multitasking and Multiprocessing.

Answer:

A multiprogramming operating system is system that allows more than one active user program (or part of user program) to be stored in main memory simultaneously. Multi programmed operating systems are fairly sophisticated. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on mass storage awaiting allocation of main memory. If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. A time-sharing system is a multiprogramming system.

Multitasking: Multitasking is a logical extension of multiprogramming. In such systems there are more than one user interacting the system at the same time. By allowing a large number of users to interact concurrently with a single computer, multitasking dramatically lowered the cost of providing computing capability, made it possible for individuals and organizations to use a computer without owning one, and promoted the interactive use of computers and the development of new interactive applications.

A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit. The term multiprocessing is generally used to refer to large computer hardware complexes found in major scientific or commercial applications. The multiprocessor system is characterized by-increased system throughput and application speedup-parallel processing. The main feature of this architecture is to provide high speed at low cost in comparison to uni- processor.

c. Discuss the fundamental properties of the following operating systems:
(i) Batch operating system
(ii) Time-sharing operating system

Answer:

(i) Batch Processing System: Refer **Page No 277-278 section 9.3 of Text Book.**

(ii) Time Sharing: Sharing of a computing resource among many users by means of multiprogramming and multi-tasking is known as timesharing. By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability, made it possible for individuals and organizations to use a computer without owning one, and promoted

Q.3 a. Consider a system with five processes < P0, P1, P2, P3, P4> and three resource types named P, Q and R. Resource type P has 12 Instances, Q has 9 and R has 11 instances. Suppose at time t0 we have the given situation as:

Process	**Allocation**			**Max**			**Available**		
	P	**Q**	**R**	**P**	**Q**	**R**	**P**	**Q**	**R**
P0	**2**	**5**	**4**	**4**	**6**	**5**	**1**	**2**	**1**
P1	**1**	**0**	**2**	**2**	**1**	**3**			
P2	**4**	**0**	**3**	**5**	**2**	**5**			
P3	**3**	**2**	**1**	**4**	**4**	**2**			
P4	**1**	**0**	**0**	**2**	**1**	**1**			

(i) Show the content of Need Matrix.
(ii) Is the given system in safe state? If yes, then generate the Safe sequence using Banker's Algorithm.

Answer:

We Know that :
Need [i, j] = Max [i, j] – Allocation [i, j]
So the content of Need Matrix is:

	P	Q	R
P0	2	1	1
P1	1	1	1
P2	1	2	2
P3	1	2	1
P4	1	1	1

Applying Banker's Safety Algo on the given system:

For Pi , If Need < = Available
Then Pi is in safe sequence
Available = Available + Allocation
So For the Processes:
(i) P0 Need = 2 1 1
Available =1 2 1
Condition is false so P0 must wait.
(ii) P1 Need = 1 1 1
Available = 1 2 1
P1 will be kept in safe sequence.
Available = Available + Allocation = 1 2 1 + 1 0 2 = 2 2 3
(iii) P2 Need = 1 2 2
Available = 2 2 3
P2 will be kept in safe sequence
Available = Available + Allocation = 2 2 3 + 4 0 3 = 6 2 6.
(iv) P3 Need = 1 2 1
Available = 6 2 6
P3 will be kept in safe sequence
Available = Available + Allocation = 6 2 6 + 3 2 1 = 9 4 7
(v) P4 Need = 1 1 1
Available = 9 4 7
P4 will be kept in safe sequence
Available = Available + Allocation = 9 4 7 + 1 0 0 = 10 4 7.
Now we have only one process P0 in waiting state. As current available P0 can be kept in safe sequence:
We take P0 whose Need = 2 1 1
Available = 10 4 7
Now P0 will be kept in safe sequence
Available = Available + Allocation = 10 4 7 + 2 5 4 = 12 9 11.
So the safe sequence is **< P1, P2, P3, P4, P0 > and the system is in Safe State.**

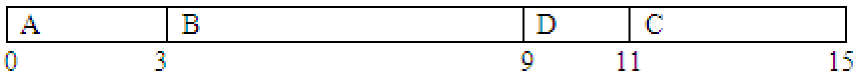
b. Consider the set of Processes < A, B, C, D > assumed to have arrived at the time sequence <0, 1, 4, 6> having Burst time < 3, 6, 4, 2> respectively. Draw the Giant Chart illustrating their execution using Shortest Job First (SJF) Algorithm, Shortest Remaining Time First (SJF Preemptive Algorithm) and calculate the Average Turn Around time and Average waiting time for both scheduling algorithms.

Answer:

Consider the set of Processes < A, B, C, D > assumed to have arrived at the time sequence <0, 1, 4, 6> having Burst time < 3, 6, 4, 2> respectively. Draw the Giant Chart illustrating their execution using Shortest Job First (SJF) Algorithm, Shortest Remaining Time First (SJF Preemptive Algorithm) and calculate the Average Turn Around time and Average waiting time for both scheduling algorithms. Comment on your result which algorithm is better and why?

Answer:

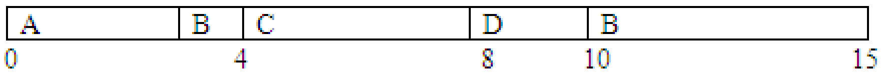
For SJF the Giant chart will be:



<u>**Calculation of Waiting Time:**</u>
Process A = 0
Process B = 3-1 = 2
Process C = (11-4) = 7
Process D = (9-6) = 3
Avg. Waiting Time= (0+2+7+3) / 4 = 12/4 = 3 Unit

<u>**Calculation of Turn Around Time:**</u>
Turn Around Time (TAT) = Waiting Time (WT) + Burst Time (CPU Processing Time)
Process A = 0+3 = 3
Process B = 2+6 = 8
Process C = 7+4 = 11
Process D = 3+2 = 5
Avg Turn Around Time = (3+8+11+5) / 4 = 27/4 = 6.75 Unit.

For SRTF the Giant chart will be:



<u>**Calculation of Waiting Time:**</u>
Process A = 0
Process B = (3-1) + (10-4) = 8
Process C = (4-4) = 0
Process D = (8-6) = 2
Avg. Waiting Time= (0+8+0+2) / 4 = 10/4 = 2.5 Unit

<u>**Calculation of Turn Around Time:**</u>
Turn Around Time (TAT) = Waiting Time (WT) + Burst Time (CPU Processing Time)
Process A = 0+3 = 3
Process B = 8+6 = 14
Process C = 0+4 = 4
Process D = 2+2 = 4
Avg Turn Around Time = (3+14+4+4) / 4 = 25/4 = 6.25 Unit.

Q.4 a. Describe the implementation of semaphores in attaining process synchronization. Explain any two classical process synchronization problems.

Answer: Refer Page no. 408,

413-414 Section 13.5 and 13.3 of Text Book

b. Explain the linked and indexed methods for allocating disk space to files.

Answer: Refer Page No.569-571 section 17.3 of Text Book

Q.5 a. Consider the following reference string: (10)
8 5 1 2 5 3 5 4 2 3 5 3 2 1 2 5 1 8 5 1
How many page faults will occur for First in First Out and Least Recently Used page replacement algorithms assuming three frames? Also calculate the page Fault rate in each case. Assume all frames are initially empty.

Answer:

Using FIFO (First in First Out) algorithm which states that: "<u>Replace that page from memory which is oldest entered in the frame of memory".</u>

FIFO: Ref String is **(8 5 1 2 5 3 5 4 2 3 5 3 2 1 2 5 1 8 5 1)**

8	8	8	2		2	2	4	4	4
	5	5	5	H	3	3	3	2	2
		1	1		1	5	5	5	3

5			5	5			8	8	8
2	H	H	1	1	H	H	1	5	5
3			3	2			2	2	1

No. of Page Faults = 15

Rate of Page Fault = (No. of Page Faults) / (No. of Frames) = 15/3 = 5.

Using LRU (Least Recently Used) algorithm which states that: <u>"Replace that page from memory which is least recently used".</u>

LRU: Ref String is (8 5 1 2 5 3 5 4 2 3 5 3 2 1 2 5 1 8 5 1)

8	8	8	2		2		4	4	4
	5	5	5	H	5	H	5	5	3
		1	1		3		3	2	2

5			1		1		1		
3	H	H	3	H	5	H	5	H	H
2			2		2		8		

No. of Page Faults = 12

Rate of Page Fault = (No. of Page Faults) / (No. of Frames) = 12/3 = 4

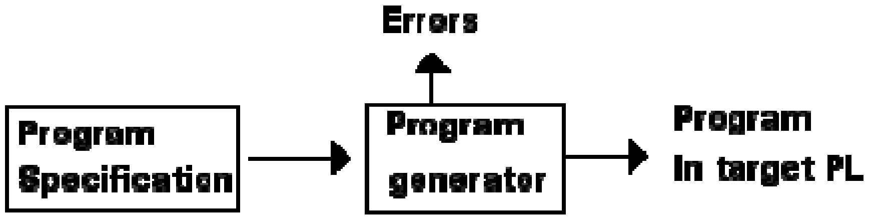
b. Explain the difference between contiguous memory allocation and non-contiguous memory allocation. (6)

Answer: Refer Pages 471, 479 of Text Book

Q.6 a. What are the various fundamental language processing activities? Discuss in detail. (4)

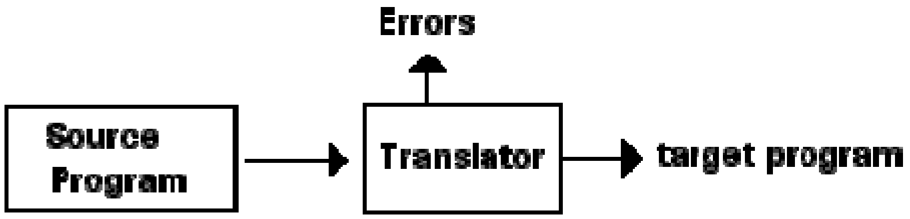
Answer:

There are two different types of language processing activities:
1. Program generation activities
2. Program execution activities
Program generation activities: A program generation activity aims at automatic generation of a program. The source language is a specification language of an application domain and the target language is typically a procedure oriented programming language. the following figure shows program generation activity



The program generator is a software system which accepts the specification of a program to be generated and generates a program in the target. PL. The program generator introduces a new domain between the application and PL domains. We call this the program generator domain. The specification gab is now gab between the application domain and the program generator domain. This gab is smaller than the gab between the application domain and PL domain.

Program execution activities:
A program execution activity organizes system. Two model program executions are:
1. Translation 2. Interpretation
Translation: The program translation models bridges execution gap by translating a program written in a PL, called the source program into an equivalent program in the machine or assembly language of the computer system.



Interpretation: The interpreter reads the source program and stores it in its memory. During interpretation it takes a statement, determines its meaning and performs actions which implement it.

b. Explain the criteria to categorize data structures used for language processors.

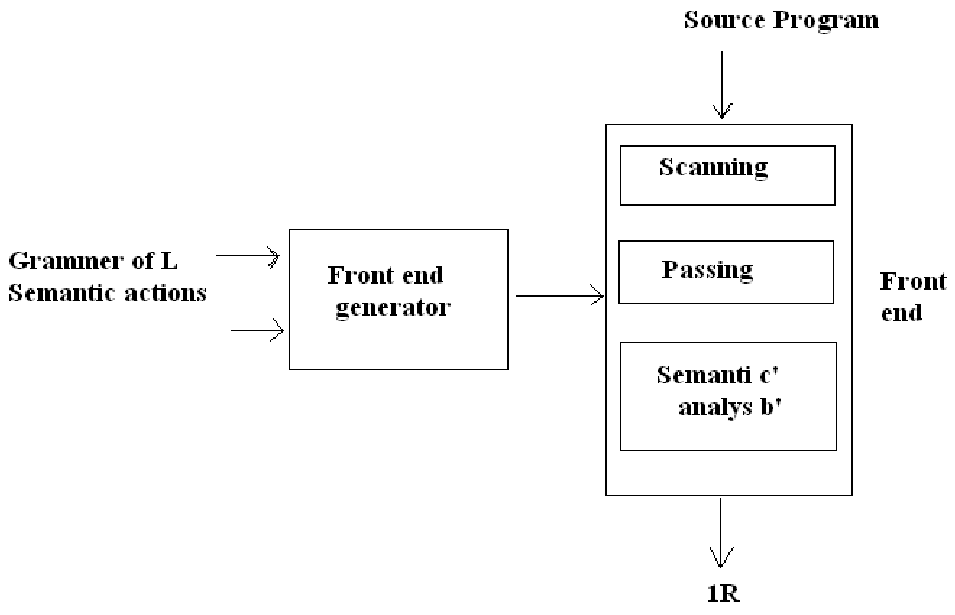
Answer:
The data structures used in language processing can be classified on the basis of the following criteria:
1. Nature of data structure (whether a linear or non-linear data structure)
2. Purpose of a data structure (whether a search data structure or an allocation data
structure)
3. Life time of a data structure (whether used during language processing or during
target program execution)
A linear data structure consists of a linear arrangement of elements in the memory. A linear data structure requires a contiguous area of memory for its elements. This poses a problem in situations where the size of a data structure is difficult to predict. The elements of non linear data structures are accessed using pointers. Hence the elements need not occupy contiguous area of memory. Search Data structures are used during language processing to maintain attribute information concerning different entities in the source program. In this the entry for an entity is created only once, but may be searched for large number of times. Allocation data structures are characterized by the fact that the address of memory area allocated to an entity is known to the users. So no search operations are conducted.

c. Explain Language Processor Development Tools through a graphic diagram. Mention the names of Language Processor Development Tools that are broadly used. (8)

Answer:

Language processor development tools (LPDTs) focuses on generation of the analysis phase of language processors. The LPDT requires the following two inputs:
1. Specification of a grammar of language L
2. Specification of semantic actions to be performed in the analysis phase.

It generates programs that perform lexical, syntax and semantic analysis of the source program and construct the IR. These programs collectively form the analysis phase of the language



Widely used LPDT's are:

Lex - A Lexical Analyzer Generator

Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. It is well suited for editor-script type transformations and for segmenting input in preparation for a parsing routine. Lex source is a table of regular expressions and corresponding program fragments.
The table is translated to a program, which reads an input stream, copying it to an output stream and partitioning the input into strings which match the given expressions. As each such string is recognized the corresponding program fragment is executed. The recognition of the expressions is performed by a deterministic finite automaton generated by Lex. The program fragments written by the user are executedin the order in which the corresponding regular expressions occur in the input stream.

YACC: Yet another Compiler-Compiler

Computer program input generally has some structure; in fact, every computer program that does input can be thought of as defining an ``input language" which it accepts. An input language may be as complex as a programming language, or as simple as a sequence of numbers. Unfortunately, usual input facilities are limited, difficult to use, and often are lax about checking their inputs for validity. YACC provides a general tool for describing the input to a computer program. The YACC user specifies the structures of his input, together with code to be invoked a each such structure is recognized. YACC turns such a specification into a subroutine that han- dles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine.

Q.7 a. Distinguish between Bottom up parsing and top down parsing. Design an algorithm for Operator Precedence Parsing.

Answer:

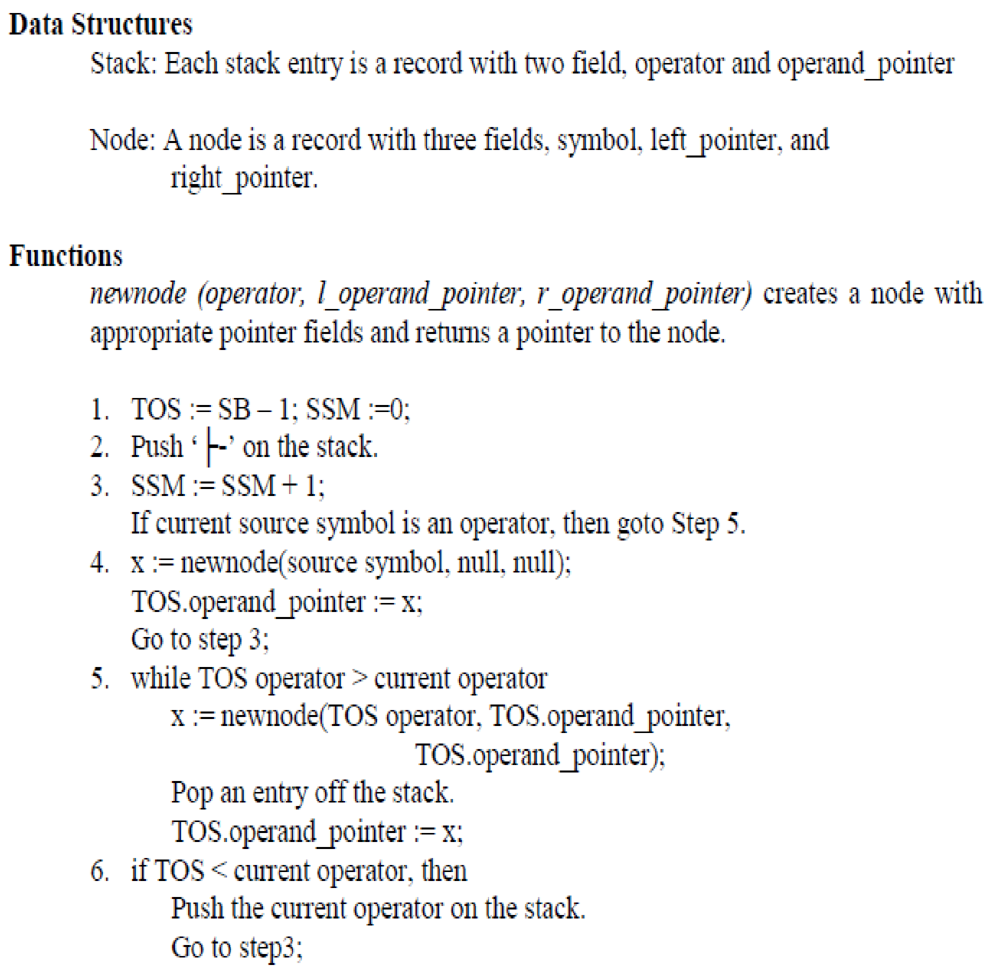
Top Down Parsing: Top down parsing according to a grammar G attempts to derive a string matching a source string through a sequence of derivations starting with the distinguished symbol of G. For a valid source string α, a top down parse thus determines a derivation sequence $S \Rightarrow \ldots\ldots \Rightarrow \ldots\ldots \Rightarrow \alpha$ The following features are needed to implement top down parsing:

• **Source string marker (SSM):** Source string marker points to the first unmatched symbol in the source string.

• **Prediction making mechanism:** This mechanism systematically selects the RHS alternatives of a production during prediction making. It must ensure that any string L_G can be derived from S.

• **Matching and backtracking mechanism:** This mechanism matches every terminal symbol generated during a derivation with the source symbol pointed to by source string marker. Backtracking is performed if the match fails. This involves resetting current sentential form (CSF) and source string marker to earlier values.

Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing. Recursive descent parsing- It is a top down parsing without backtracking. This parsing technique uses a set of recursive procedures to perform parsing. Salient advantages of recursive descent parsing are its simplicity and generality. It can be implemented in any language supporting recursive procedures.



Data Structures

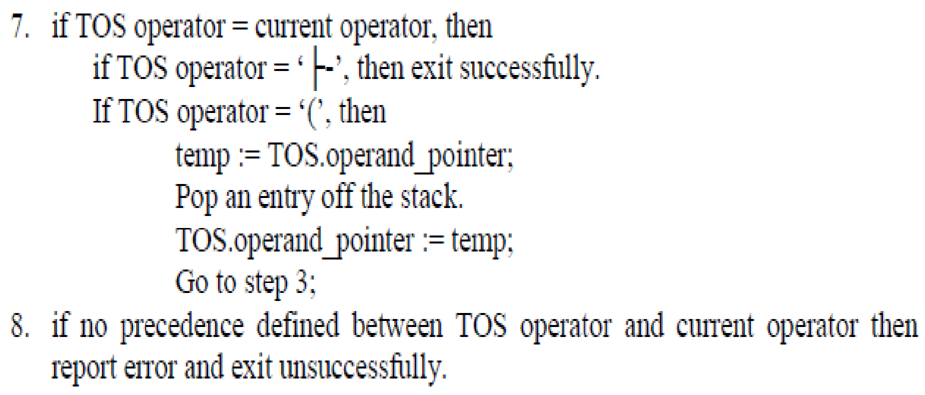
Stack: Each stack entry is a record with two field, operator and operand_pointer

Node: A node is a record with three fields, symbol, left_pointer, and right_pointer.

Functions

newnode (operator, l_operand_pointer, r_operand_pointer) creates a node with appropriate pointer fields and returns a pointer to the node.

```
1. TOS := SB – 1; SSM :=0;
2. Push '├-' on the stack.
3. SSM := SSM + 1;
   If current source symbol is an operator, then goto Step 5.
4. x := newnode(source symbol, null, null);
   TOS.operand_pointer := x;
   Go to step 3;
5. while TOS operator > current operator
     x := newnode(TOS operator, TOS.operand_pointer,
                  TOS.operand_pointer);
     Pop an entry off the stack.
     TOS.operand_pointer := x;
6. if TOS < current operator, then
     Push the current operator on the stack.
     Go to step3;
```



```
7. if TOS operator = current operator, then
     if TOS operator = '├-', then exit successfully.
     If TOS operator = '(', then
         temp := TOS.operand_pointer;
         Pop an entry off the stack.
         TOS.operand_pointer := temp;
         Go to step 3;
8. if no precedence defined between TOS operator and current operator then
   report error and exit unsuccessfully.
```

b. Describe the data structure and algorithm for the first pass of the Linker.

Answer: Refer Page No. 240 section 7.4.1 of text book

Q.8 a. Explain the structure of load-and-go assembler in detail. (6)

Answer:

Load and Go Assembler

The simplest assembler program is the load and go assembler. It accepts as input a program whose instructions are essentially one to one correspondence with those of machine language but with symbolic names used for operators and operands. It produces machine language as output which are loaded directly in main memory and gets executed. The translation is usually performed in a single pass over the input program text. The resulting machine language program occupies storage locations which are fixed at the time of translation and cannot be changed subsequently. The program can call library subroutines, provided that they occupy other locations than those required by the program. No provision is made for combining separate subprograms translated in this manner.

The load and go assembler forgoes the advantages of modular program development. Among the most of these are

(1) the ability to design code and test different program components in parallel.

(2) change in one particular module does not require scanning the rest of program. Most assemblers are therefore designed to satisfy the desire to create programs in modules. These module assemblers. generally are developed in a two-pass translation. During the first pass the assembler examines the assembler-language program and collects the symbolic names into a table. During the second pass, the assembler generates code which is not quite in machine language. It is rather in a similar form, sometimes called "relocatable code" and here called object code. The program module in object-code form is typically called an object module.

b. Write brief notes on the following given Assembler Directives:-
(i) EQU
(ii) START & END

Answer:

(i) EQU

The EQU statement has the syntax <symbol> EQU <address spec> where <address spec> is an <operand spec> or <constant>. The EQU statement defines the symbol to represent <address spec>. This differs from the DC/DS statement as no LC processing is implied. Thus EQU simply associates the name <symbol> with <address spec>.

(ii) START & END

The START directive indicates that the first word of the target program generated by the assembler should be placed in the memory word with address <*constant*>. START <*constant*> The END directive indicates the end of the source program. The optional <*operand spec*> indicates the address on the instruction where the execution of the program should begin. END [<*operand spec*>]

c. With the help of a diagram give the overview of two pass assembly.

Answer: Refer Page 95 of Text Book

Q.9 a. What are the features that a compiler uses to implement function calls?
Answer:

The compiler uses a set of features to implement function calls. These are described below:

• *Parameter list:* The parameter list contains a descriptor for each actual parameter of the function call. The notation D_p is used to represent the descriptor corresponding to the formal parameter p.

• *Save area:* The called function saves the contents of CPU registers in this area before beginning its execution. The register contents are restored from this area before returning from the function.

• *Calling conventions:* These are execution time assumptions shared by the called function and its caller(s). The conventions include the following:

a) How the parameter list is accessed.
b) How the save area is accessed.
c) How the transfers of control at call and return are implemented.
d) How the function value is returned to the calling program.

Most machine architectures provide special instructions to implement items c) and d).

b. Write short notes on the following:
(i) Local and Global optimization
(ii) Quadruples and Triples
(iii) Dynamic and Static Pointer

Answer:

(i) Local and Global optimization

Local Optimization: The optimizing transformations are applied over small segments of a program consisting of a few segments. Local optimization provides limited benefits at a low cost. The scope of local optimization is a basic block which is an 'essentially sequential' segment in the source program. The cost of local optimization is low because the sequential nature of the basic block simplifies the analysis needed for optimization. The benefits are limited because certain optimizations are beyond the scope of local optimization

Global Optimization: The optimizing transformations are applied over a program unit, i.e. over a function or a procedure. Compared to local optimization, global optimization requires more analysis effort to establish the feasibility of an optimization. Consider global common sub expression elimination. If some expression $x_* y$ occurs in a set of basic blocks SB of program P, its occurrence in a block $b_j \in$ SB can be eliminated if the following two conditions are satisfied for every execution of P:

- o Basic block b_j is executed only after some block $b_k \in$ SB has been executed one or more times.
- o No assignments to x or y have been executed after the last evaluation of $x_* y$ in block b_k.

(ii) Quadruples and Triples

A **triple** is a representation of an elementary operation in the form of a pseudo machine instruction. Each operand of a triple is either a variable or constant or the result of some evaluation represented by another triple.

Operator	Operand1	Operand2

A **quadruple** represents an elementary evaluation in the following format:

Operator	Operand 1	Operand 2	Result name

Here, result name designates the result of the evaluation it can be used as the operand of another quadruple. This is more convenient than using a number to designate a subexpression.

(iii) Dynamic and Static Pointer : Refer **Page No. 170-172 section no. 6.2.2 of Text Book**

TEXT BOOK

I. Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005