Marking Scheme U218-146(T1)

Q 1) a) Six steps of problem solving – Each step carry one mark. [6]

b) To addition of digits. [6]
Flow chart – 3 marks
Algorithm – 3 marks

c) Definition of problem – 2mark
ways to solve problem – 2marks [4]

OR

Q2) a) Different strategies for algorithm design – explanation with example [6]

b) Write output of following functions [6]

Sign(0) 0

Abs (-8) 8

String(-345.88) "-345.88"

Max (5,7,8,9) 9

Mid(s,3,2) ea

Right(s,3) ater

where s=theater

c) Need of function – at least four points each carry 1 mark. [4]

Q3) a) List down major types of module and explain their function with example

List of types of module – 2 marks
Function with example – 4 marks [6]

b) Correct decision table carries four marks [4]

c) Case structure explanation with flowchart 2 marks
example – 2 marks [4]

OR

Q4) a) Explain three decision logic structure with example [6]
Explanation with example.

b) In a multiplex the charges for a movie varies according to the age of the persons. Using the positive logic, develop a solution to print the ticket changes given the age of person. [4]

Solution in the form of flowchart or algorithm

c) call by value 2 marks [4]
call by reference 2 marks

U218-146 (T1)

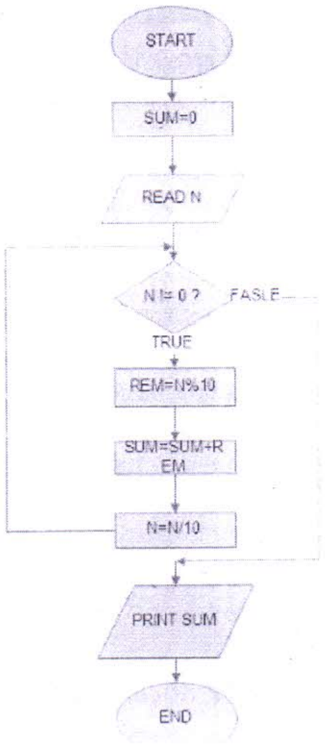
Solution

Q 1) a The six steps of problem solving include the following [6]

1. Identify the problem: What is the problem that needs to be solved?
2. Understand the problem: Are there considerations that need to be taken into account?
3. Identify alternative ways to solve the problem: There may be multiple solutions to the problem. Write down all possible solutions, and at this point, don't rule anything out.
4. Select the best way to solve the problem from the list of alternative solutions: Factors involved in selecting the best solution include efficiency, schedule, cost, available resources, or the need for a long-term solution.
5. List instructions that enable you to solve the problem using the selected solution: When solving a problem with computers, this can include pseudocode, or a combination of English and code.
6. Evaluate the solution: Determine if the solution solves the problem. If not, return to identifying and understanding the problem.

b) Draw a flowchart and write an algorithm to find addition of digits. [6]

Flowchart



Algorithm

```
Step 1: Input N

Step 2: Sum = 0

Step 3: While (N != 0)

Rem = N % 10;

Sum = Sum + Rem;

N = N / 10;
```

Step 4: Print Sum

c) What is meant by problem? What are the ways to solve problem [4]

A problem can present itself in many forms. Examples of problems are as follows:

- Question: What movie should we see?
- Need to do something: Generate the monthly payroll.
- Obstacle to progress: A part is missing that is causing work to halt on a project.
- Status of a process: What is the status of the project?
- What-if simulation: How much money needs to be deducted from each pay check to save for a down payment on a car?
- Evaluation of a solution to another problem: Should traveling be done by train or car?
- Lack of information: How much money is left on the mortgage?
- Undesirable state: A person has gone into debt.
- Lack of control: A company has decided to downsize.

There are many ways in which a problem can be solved. The following are some of the possibilities:

- Algorithm: An algorithm is a series of steps used to arrive at an outcome that represents the best answer to a problem. It is a specification of a behavioral process. An algorithm is a finite set of instructions that govern a behavior step by step, such as the manipulation of data. Problems that require algorithms are often the problems that computers solve best.
- Heuristic solutions: Heuristic solutions involve the use of human ingenuity and reasoning. They are based on learning through experience.

Passing of time: Some problems are solved simply by the passing of time. An undesirable state may change, or it may no longer be a problem.

Gaining resources: The solution to a problem may be more resources—such as more money, time, or people

OR

Q2) a) Explain different strategies for algorithm design [6]

Strategies to solve the problem: brute force, greedy, divide and conquer, and backtracking.

Brute Force

The brute force method of algorithm design is intended to bring into mind hammering a screw into a board of wood. The solution may not be elegant or efficient, but it will work. Generally, this involves considering every aspect of every element of an array, regardless of whether each element is important. It is an exhaustive, programmatic consideration of everything, which takes time. Generally, brute force methods are inefficient but still serviceable in many situations.

Greedy

Greedy algorithms involve considering the largest portion of the problem set first. For example, if the problem considers how to pack a set of items into a bag or a box, the greedy method would put the largest item in first, followed by the second largest, and so forth.

The greedy method will occasionally find an efficient method; however, more often than not, the solution will be similar to a brute force method.

Divide & Conquer

- Suppose that a problem involved shuffling a deck of cards. It could be determined that a deck of cards is shuffled if each half of the deck is shuffled and randomly recombined. Due to the fact each half could be considered a deck, what occurs is a recursive solution. The only component left to define is the base case, which would say that the dividing of decks stops when the deck only consists of one card.

- Afterwards, each subdeck could be randomly combined to reproduce the larger deck. This could be continued until the full deck is shuffled. This method is called divide and conquer, when the algorithm seeks to divide the problem into smaller problems which are, in turn, solved in the same manner.

Back Tracking

A method often called backtracking is one other possibility toward finding an efficient solution by starting with the solution and finding the problem. If an algorithm can be developed that takes an appropriate solution and slowly takes parts of it away until it comes to the problem and the algorithm can be reversed, it is sometimes possible to determine an efficient solution to a problem.

b) Write output of following functions [6]

1. Sign(0) : 0
2. Abs (-8) : 8
3. String(-345.88) : "-345.88"
4. Max (5,7,8,9) : 9
5. Mid(S,3,2) ea
6. Right(s,3) ater

where s=theater

c) What is the need of function? [4]

Modularize a program

Divide and conquer

Manageable program development

Software reusability

Use existing functions as building blocks form new programs

Abstraction-hide internal details (library functions)

Avoid code repetition

Q3) a) List down major types of module and explain their function with example. [6]

- Types of modules
  - The control module: It shows the overall flow of the data through the program. All other modules are subordinate to it.
  - The initialization modules: Processes instructions that are executed only once during the program, and only at the beginning. These instructions include opening files and setting the beginning values of variables used in processing.
  - The Process modules: may be processed only once, or they may be part of a loop, which is processed more than one during the solution. There are several kind of Process modules
    - Calculation modules
    - Print modules
    - Read and data validation modules
  - Wrapup modules: Process all instructions that are executed only once during the program and only at the end. These instructions include closing files and printing totals, among others
  - Modules in an object-oriented program may include event modules such as mouse down, mouse up, key entry, & so on

b) Draw a decision table for the following set of conditions for gross income tax and rate:

1) Gross <= 5,000 tax rate 5% 2) income between 5,000 - 10,000 tax rate 8% 3) income between 10,000 - 15,000 tax rate 10% 4) Gross > 15,000 tax rate 15% [4]

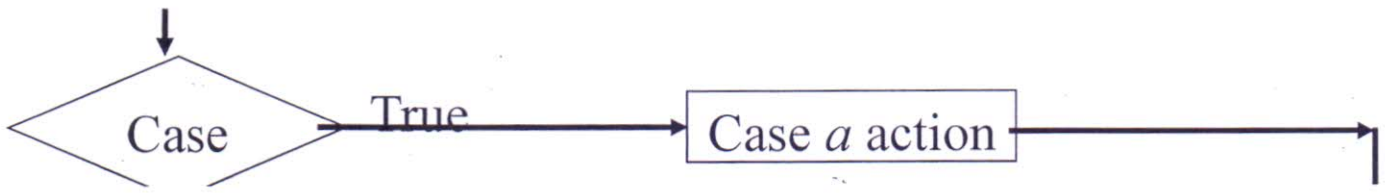
Decision Table

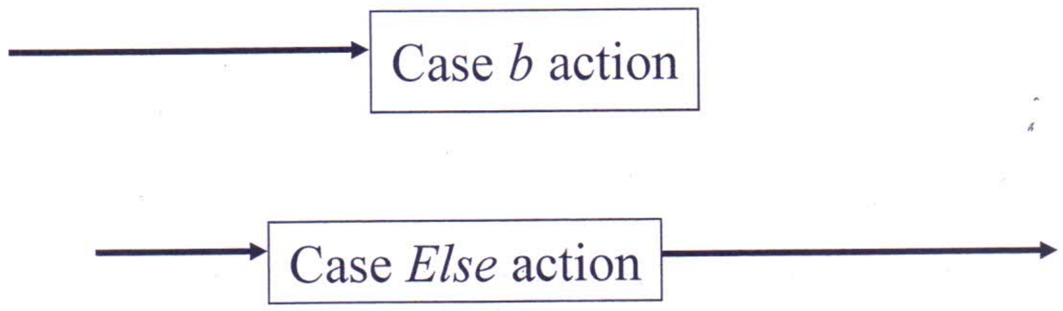
Solution -> Rate Condition	5%	8%	10%	15%
Gross <= 5,000	×			
income between 5,000 - 10,000		×		
income between 10,000 - 15,000			×	
Gross > 15,000				×

c) Explain case structure with flowchart and example [4]

- Is similar to a series of If/Then/Else statements.
- Positive Logic
- Syntax:

```
Select Case testvalue
        Case value1
            statement group 1
        Case value2
            statement group 2
End Select
```





Case Example

```
Select Case Grade
Case 90..100
        LetterGrade = "A"
Case 80..89.9
        LetterGrade = "B"
Case 70..79.9
        LetterGrade = "C"
Case 60..69.9
        LetterGrade = "D"
Else
        LetterGrade = "F"
End Select
```

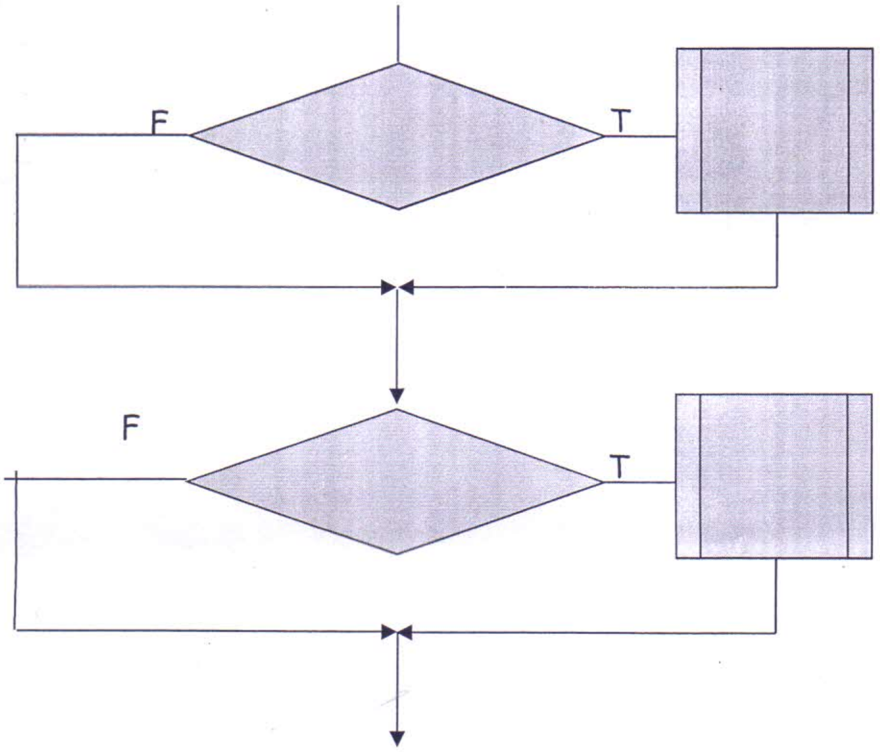
OR

Q4) a) Explain three decision logic structure with example [6]

- Straight-through Logic
  - All decisions are processed sequentially, one after another.
  - Least efficient, but most thorough
- Positive Logic
  - Processing flow continues through the module instead of processing succeeding decisions, once the result is True.
- Negative Logic
  - Flow is based on result being False.
- Nested decisions use Positive or Negative, but not Straight-through.

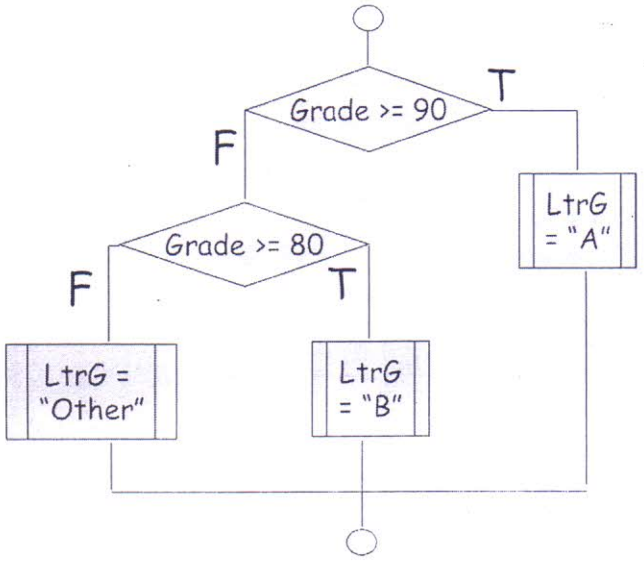
Straight-through Logic

- All conditions are tested.
- Least efficient, but most exhaustive.



Positive Logic:

- Uses If/Then/Else instructions
- Continues processing based on True results

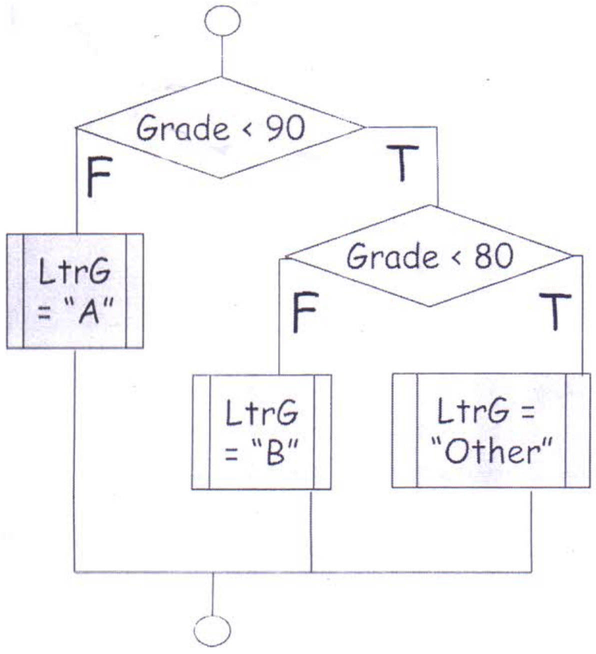


Negative Logic:

- Executes process

  based on

  False
- Processes another decision when the result is True



b) Using positive logic, solve the following set of conditions to calculate hotel bill: [4]
1) Sales of eatables up to 100 Rs., 11% discount
2) Sales of eatables up to 1000 Rs., 22% discount
3) Sales of eatables up to 10000 Rs., 33% discount

As per attached sheet

c) What are the parameter passing techniques? [4]

Call by Value

Calling a function with parameters passed as values
```
int a=10;               void fun(int a)
fun(a);                 {
                                  defn;
                             }
```
Here fun (a) is a call by value.
Any modification done with in the function is local to it and will not be effected outside the function

Example program – Call by value
```
#include<stdio.h>
void main()
{
        int a=10;
        printf("%d",a);             a=10
        fun(a);
        printf("%d",a);             a=10
}
void fun(int x)
{
        printf("%d",x)              x=10
        x++;
        printf("%d",x);             x=11
}
```

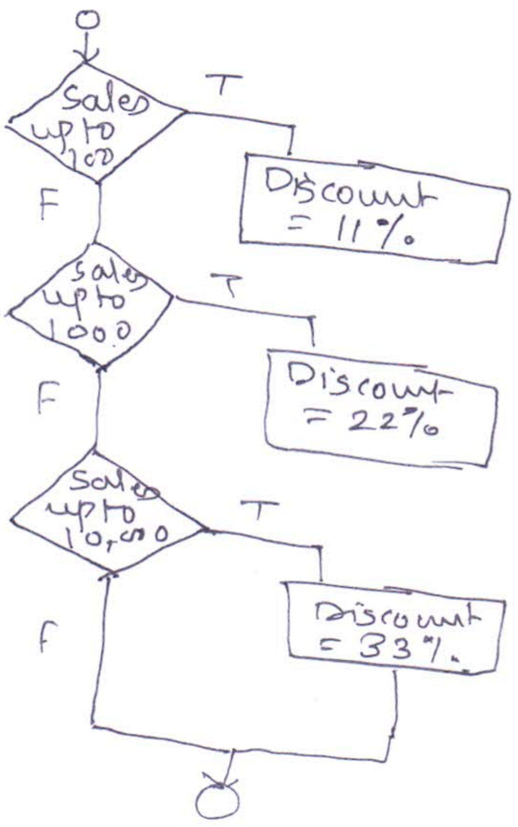
Call by reference
Calling a function by passing pointers as parameters (address of variables is passed instead of variables)
```
int a=1;                void fun(int *x)
fun(&a);                {
                                  defn;
                             }
```
Any modification done to variable a will effect outside the function also

Example Program – Call by reference

```
#include<stdio.h>
void main()
{
        int a=10;
        printf("%d",a);              a=10
        fun(&a);
        printf("%d",a);              a=11
}
void fun(int *x)
{
        printf("%d",x)               x=10
        x++;
        printf("%d",x);              x=11
}
```

b)

Flow chart



Algorithm

```
If
        sales <= 100
            Discount = 11%.
elseif
        sales > 100 or sclles <= 1000
            Discount = 22%
else if
        sales > 1000 or sules <= 10,000
            Disiount = 33%
```