Name :- Manjari ①
USN :- 2GI18CS419

EXPT no :— 1 Date :—

Problem Statement :~ Design & develop a program in a language of your choice to solve the triangle problem. defined as follows : Accept three integers which are supposed to be the tree sides of a triangle & determine if the three sides values represent an equilateral triangle, isosceles triangle, scalene triangle or they do not form a triangle at all. Derive the test cases for your program based on decision-table approach, execute the test cases & discuss the results.

Objectives :-

- To solve the triangle problem
- The test cases should be based on decision table approach.

Theory :-

- The system should accept 3 positive integers numbers (a, b, c) which represents 3 sides of the triangle. Based on the input it should determine if a triangle can be formed or not.
- If the above requirement is satisfied then the system should determine the type of the triangle, which can be
  - Equilateral (all the three sides are equal)
  - Isosceles (two sides are equal)
  - Scalene (All the sides are unequal)

Else the suitable error message should be displayed. Here we assume that the user gives the positive integer numbers as input.

- From the given requirements we can draw the following conditions: C1 : $a < b + c$?

C2 :

$b < a + c$?

C3 :

$c < a + b$?

C4 : $a = b$?

C5 : $a = c$?

C6 : $b = c$?

program code :—

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <process.h>
int main()
{
  int a, b, c;
  clrscr();
  printf("Enter the three sides of a triangle");
  scanf("%d%d%d", &a, &b, &c);
  if (a<b+c)&&(b<a+c)&&(c<a+b))
  {
    if ((a==b)&&(b==c))
    {
      printf("Equilateral triangle");
    }
    else if (a!=b)&&(a!=c)&&(b!=c))
    {
      printf("Scalene triangle");
    }
    else
    {
      printf("Isosceles triangle");
    }
  else
  {
    printf("triangle cannot be formed");
  }
  getch();
  return 0;
}
```

The decision table is given below:

Conditions	Condition entries (Rules)										
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
C1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
C2: b<a+c?	—	F	T	T	T	T	T	T	T	T	T
C3: c<a+b?	—	—	F	T	T	T	T	T	T	T	T
C4: a=b?	—	—	—	F	T	T	T	F	F	F	T
C5: a=c?	—	—	—	T	F	T	F	T	F	F	T
C6: b=c?	—	—	—	T	T	F	F	F	T	F	T
Actions	**Action entries**										
a1: Not a triangle	X	X	X								
a2: Scalene										X	
a3: Isosceles							X	X	X		
a4: Equilateral											X
a5: Impossible				X	X	X					

* Decision table :—

It is used to depict complex logical relationships between input data. Its a method use to build a complete set of test cases without using the internal structure of the program in question.

In the decision table the symbol '-' indicates doesn't care values. The table shows the 6 conditions & 5 actions. All the conditions in the decision table are binary ; hence it is called "Limited entry decision table".

* Test case table :-

- It done using the decision table

TCID	test case	a	b	c	Expected o/p	Actual output	status
1	Test for requirement 1	4	1	2	Not a triangle	Not a triag	TC pass
2	Testing for requirement 1	1	4	2	Not a triangle	Not a triag	TC pass
3	Testing for requirement 1	1	2	4	Not a triangle	Not a triag	TC pass
4	Testing for requirement 2	5	5	5	Equilateral	Equilateral	TC pass
5	Testing for requirement 2	2	2	3	Isosceles	Isosceles	TC pass
6	Testing for requirement 2	2	3	2	Isosceles	Isosceles	TC pass
7	Testing for requirement 2	3	2	2	Isosceles	Isosceles	TC pass
8	Testing for requirement 2	3	4	5	scalene	scalene	TC pass

Name: Manjiri Gunaji
Usn: 2GI18CS419

Enter three sides of the triangle :
4
1
2
Traingle cannot be formed

Enter three sides of the triangle :
1
4
2
Traingle cannot be formed

Enter three sides of the triangle :
1
2
4
Traingle cannot be formed

Enter three sides of the triangle :
5
5
5
Equilateral triangle

Enter three sides of the triangle :
2
2
3
Isosceles triangle

Enter three sides of the triangle :
2
3
2
Isosceles triangle

Enter three sides of the triangle :
3
2
2
Isosceles triangle

Enter three sides of the triangle :
3
4
5

Scalene triangle

Test report :-

No of TC's executed : 08
No of defects Raised : 0
No of TC's passed : 08
No of TC's failed : 0

Conclusion :- the technique is indicate for applications characterized by the if-else logic, logical relationship; The decision table based testing works well for triangle problem because a lot of decision making takes place.

References :-

Paul C jorgensen .3rd edition.

Exp no:-2 Date:-

problem definition:- Design & develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the 3 sides of a triangle & determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive the test cases for your program based on boundary value analysis & execute the test cases & discuss the results.

Exp no:-3

*Objectives :-

- to solve the triangle problem
- use the technique boundary value analysis.

*Theory :-

- Boundary value analysis :- It depends on the output & the constraint on the output. so its least worked on input domain. It yields 4n+1 so inputs are n=3 the 4*3+1 = 12+1=13.

The maximum limit of each sides a, b, c of the triangle is 10 units according to requirement R4, so a, b, c lies between.

$0 \leq a \leq 10$

$0 \leq b \leq 10$

$0 \leq c \leq 10$

*requirements :- 3 positive integers (a, b, c)

- Based on the input the triangle can be formed or not.
- If the above requirement is satisfied then the system should determine the type of the triangle, which can be

equilateral

isosceles

Scalene.

- upper limit for the size of any side is 10.

* program :-

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <process.h>
int main()
{
  int a, b, c;
  clrscr();
  printf("Enter 3 sides of the triangle");
  scanf("%d%d%d", &a, &b, &c);
  if (a>10)||(b>10)||(c>10))
  {
    printf("out of range");
    getch();
    exit(0);
  }
  if (a<b+c)&&(b<a+c)&&(c<a+b))
  {
    if (a==b)&&(b==c))
    {
      printf("Equilateral triangle");
    }
    else if ((a!=b)&&(a!=c)&&(b!=c))
    {
      printf("Scalene triangle");
    }
    Else
      printf("Isosceles triangle");
  }
  else
  {
    printf("Triangle cannot be formed");
  }
  getch();
  return 0;
}
```

* Test case for the triangle problem :-

TC ID	TC description	I/p data a	b	c	expected o/p	Actual o/p	status
1	for a i/p is not given	X	3	6	Not a triangle	-	
2	for b i/p is not given	5	X	4	Not a triangle	-	
3	for c i/p is not given	4	7	X	Not a triangle	-	
4	i/p of c is negative	5	5	-1	Not a triangle	Not a triangle	pass
5	2 sides same 1 side different	5	5	1	Isosceles	Isosceles	pass
6	All sides of i/p equal	5	5	5	equilateral	equilateral	pass
7	2 sides same 1 side different	5	5	9	Isosceles	Isosceles	pass
8	The i/p of c is out of range	5	5	10	Not a triangle	Not a triangle	pass
9	2 sides are same 1 side is given different	5	1	5	Isosceles	Isosceles	pass
10	2 sides same 1 side different	5	2	5	Isosceles	Isosceles	pass
11	2 sides same 1 side is out of range	5	9	5	Isosceles	Isosceles	pass
12	2 sides are same one side is given different input as out of range.	5	10	5	Not a triangle	Not a triangle	pass
13	2 sides are same, sides given different input	1	5	5	Isosceles	Isosceles	pass
14	2 sides are same 1 side is given different input.	2	5	5	Isosceles.	Isosceles	pass
15	2 sides are same 1 side given different input.	9	5	5	Isosceles	Isosceles	pass
16	2 sides are same & other	10	5	5	Not a triangle	Not a triangle	pass

Name: Manjiri Gunaji
Usn: 2GI18CS419
Expt no : 2

Enter three sides of the triangle :
X
3
6
Triangle cannot be formed

Enter three sides of the triangle :
5
X
4
Triangle cannot be formed

Enter three sides of the triangle :
5
7
X
Triangle cannot be formed

Enter three sides of the triangle :
5
5
-1
Triangle cannot be formed

Enter three sides of the triangle :
5
5
1
Isosceles triangle

Enter three sides of the triangle :
5
5
5
Equilateral triangle

Enter three sides of the triangle :
5
5
9
Isosceles triangle

Enter three sides of the triangle :
5
5
10
Triangle cannot be formed

Enter three sides of the triangle :
5
1
5
Isosceles triangle

Enter three sides of the triangle :
5
2
5
Isosceles triangle

Enter three sides of the triangle :
5
9
5
Isosceles triangle

Enter three sides of the triangle :
5
10
5
Triangle cannot be formed

Enter three sides of the triangle :
1
5
5
Isosceles triangle

Enter three sides of the triangle :
2
5
5
Isosceles triangle

Enter three sides of the triangle :
9
5
5
Isosceles triangle

Enter three sides of the triangle :
10
5
5
Isosceles triangle

Test report :-

No of TCs executed :- 16
No of defects raised :- 0
No of TC's pass : 16
No of TC's failed :- 0.

Conclusion :- The triangle problem is been implemented by using the approach of boundary based analysis. & hence the test cases are executed & compared with the expected result.

References :- paul c jorgensen 3rd edition

Exptno :- 3 Date :-

problem definition :- Design & develop a program in a language of your choice to solve the triangle problem defined as follows : Accept three integers which are supposed to be the three sides of a triangle & determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all, Assume that the upper limit for the size of any side is 10 . Derive the test cases for your program based on equivalence class partitioning. Execute the test case & discuss the results.

Objectives :-

- To develop the triangle problem.
- use the technique equivalence class partitioning.

Theory :-

equivalence class partitioning :- This technique is focused on the input domain we can obtain a richer set of test cases. What are some of the possibilities for the three integers, a, b & c ? they can all the equal exactly 1 pair can be equal.

the maximum limit of each side a, b, c of the triangle is 10. so a, b, & c lies between

$0 \leq a \leq 10$

$0 \leq b \leq 10$

$0 \leq c \leq 10$.

requirements :- 3 inputs a, b, c. which represents the 3 sides of the triangle. based on that check whether the triangle can be form or not. the type of the triangle can be

- equilateral (all 3 sides are equal)
- Isosceles (2 sides are equal)
- Scalene (All the 3 sides are unequal)

Program :-

```
#include <stdio.h>
#include <ctype.h>
#include <process.h>
#include <conio.h>
int main()
```

```
{
    int a, b, c;
    clrscr();
    printf("enter 3 sides of the triangle");
    scanf("%d %d %d", &a, &b, &c);
    if((a>10)||(b>10)||(c>10))
    {
        print("out of range");
        getch();
        exit(0);
    }
    if((a<b+c)&&(b<a+c)&&(c<a+b))
    {
        if((a==b)&&(b==c))
        {
            printf("equilateral triangle");
        }
        else if((a!=b)&&(a!=c)&&(b!=c))
        {
            printf("scalene triangle");
        }
        else
            print("isosceles triangle");
    }
    else
    {
        print("triagle cannot be formed");
    }
    getch();
    return 0;
}
```

Test case:

TC id	TC description	I/P data a	b	c	expected o/p	Actual o/p	status
1	WN1	5	5	5	equilateral	equilateral	tc pass
2	WN2	2	2	3	Isosceles	Isosceles	TC pass
3	WN3	3	4	5	Scalene	Scalene	TC pass
4	WN4	4	1	2	Not a triangle	Not a triangle	TC pass
5	WR1	-1	5	5	"a" is not in range of permitted	out of range	pass
6	WR2	5	-1	5	"b" is not in range of permitted	out of range	pass
7	WR3	5	5	-1	"c" is not in range of permitted	out of range	pass
8	WR4	11	5	5	"a" value is exceed the range	out of range	pass
9	WR5	5	11	5	"b" value exceed the range	out of range	pass
10	WR6	5	5	11	"c" value exceed the range		

Name: Manjiri Gunaji
Usn: 2GI18CS419
Expt no : 3

Enter three sides of the triangle :
5
5
5
Equilateral triangle

Enter three sides of the triangle :
2
2
3
Isosceles triangle

Enter three sides of the triangle :
3
4
5
Scalene Triangle

Enter three sides of the triangle :
4
1
2
Triangle cannot be formed

Enter three sides of the triangle :
-1
5
5
Out of range

Enter three sides of the triangle :
5
5
-1
Out of range

```
Enter three sides of the triangle :
5
-1
5
Out of range

Enter three sides of the triangle :
11
5
5
Out of range

Enter three sides of the triangle :
5
11
5
Out of range

Enter three sides of the triangle :
5
5
11
Out of range
```

Test Case Report :-

No of TC executed :- 10
No of defects raised : 0
No of TC pass : 10
No of TC fails : 0.

Conclusion :- Implemented the triangle problem using the equivalence class and also verified the test cases with the expected output.

References : paul C Jorgensen 3rd edition.

Expt no:- 4 Date:- 10-10-20.

problem definition:- Design & develop, code & run the program in any suitable language to solve the commission problem. Analyze it from the perspective of dataflow testing, derive different test cases, Execute & test the results.

Objective :- To solve & analyze the commission problem.

Theory :—

* data flow testing :—

A structural testing techniques it aims to execute subpaths from points where each variable is defined to points where it is referenced. data flow testing is centered on variables, i.e. the sequence of events of given data.

- variable definition :— where a variable is given a new value. Its declaration is not a definition.
- variable uses :- where a variable is not given a new value.
- p-uses :— occur in the predicate portion of a decision statement such as if-then-else, while do etc.
- c-uses :— All others, including variable occurrences in the right hand side of an assignment.
- du-path :- Its a subpath - test case definition based on 4 groups of coverages.
  - All definition
  - All c-uses
  - All p-uses
  - All du-paths

Data flow testing :- Key steps :—

Given a code.

1) No the lines
2) List the variables
3) List occurances
4) Identify du-pairs & their use.
5) define the test cases

fig :- list occurances & assign category to each variable

line	Category		
	definition	c-use	puse.
1			
2			
3			
4			
5			
6			
7			
8			
9	locks, stocks, barrels.		locks, stocks, barrels.
10			
11			
12	flag.		
13			flag.
14			
15			
16			
17			
18			
19			
20	t_sales	lock, stocks, barrels.	t_sales
21			
22			
23	commission	t-sales	
24			
25			t-sales
26			
27	commission	commission, t-sales.	
28			
29			
30			
31			
32	commission		
33	commission	commission.	
34	commission.	commission, t-sale	
35			
36		commission.	
37			
38			
39			

Algorithm :-

Step1 :- define lockprice = 45.0, stockprice = 30.0
barrelprice = 25.0.
Step2 :- input locks
Step3 : while (locks != -1) input device uses -1 to indicate
end of data goto. step12
Step4 : inputs (stocks, barrels)
Step5 :- compute locksales, stocksales, barrelsales & sales
Step6 : output ("total sales")
Step7 : if (sales > 1800) goto step8 else step9
Step8 : commission = 0.10 * 1000.0, commission = commission +
0.15 * 800.0.
Step9 :- if (sales > 1000.0) goto step8 else goto step11
Step10 :- commission = 0.10 * 100.0.
Step11 :- output ("commission")
Step12 :- Exist

program :-

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int locks, stocks, barrels, t_sales, flag = 0;
    float commision;
    printf("Enter the total no of locks");
    scanf("%d", &locks)
    if (locks <= 0) || (locks >= 70))
{
    flag = 1;
}. if
```

fig:- defined test-case

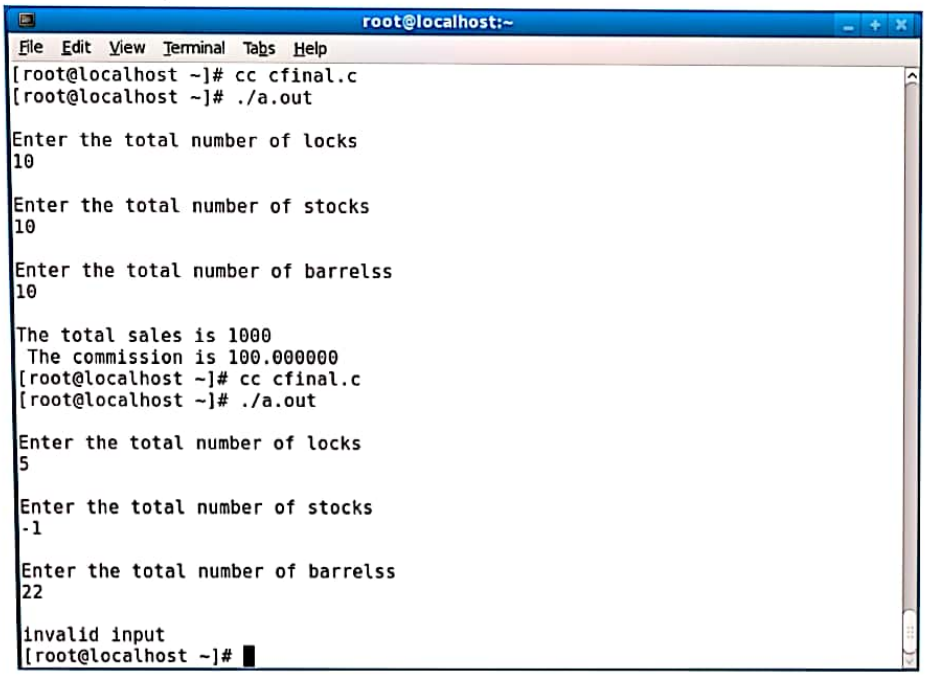
definition-use pair start line - end line	variables () c-use	puse
9 → 10		locks
9 → 10		stocks
9 → 10		barrels
9 → 20	locks	
9 → 20	stocks	
9 → 20	barrels.	
9 → 20.		flag
12 → 24		t-sale
20 → 21		
20 → 23	t-sale	t-sale.
20 → 25		
20 → 28	t-sale	
20 → 34	t-sale	
23 → 36	commission	
27 → 28	commission	
28 → 36	commission	
32 → 33	commission	
33 → 34	commission	
34 → 36	commission	

Test cases based on all definition.

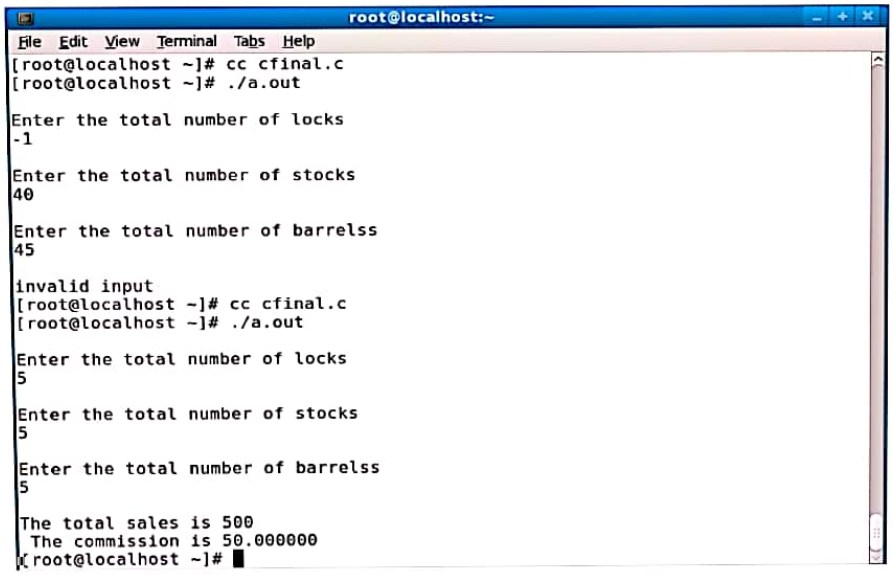
variables	du-pair	subpath	Inputs. locks	stocks	barrels	Expected outputs t-sales	commission
locks, stocks, barrels	9→20	9,10,20	10	10	10	1000	
locks, stocks barrels	9→10	9→10	5	-1	22	invalid input.	
flag	12→24	12→14	-1	40	45	invalid input	
t-sales	20→24	20,21	5	5	5	500	
t-sales	20→25	20,21,25	15	15	15	1500	
commission	23→36	23→36	5	5	5		50
commission	27→36	27,28,36	15	15	15		125
commission	32→36	32,33,34,36	25	25	25		360.

```
printf("Enter the total no.stocks");
scanf("%d", &stocks);
if((stocks<=0)||(stocks>80))
{
  flag=1;
}
printf("Enter the total no of barrels");
scanf("%d", &barrels);
if((barrels<=0)||(barrels>90))
{
  flag=1;
}
if(flag==1)
{
  printf("Invalid input");
  getch();
  exit(0);
}
t_sales=(stocks*45)+(stocks*30)+(barrels*25);
if(t_sales<=1000)
{
  commission=0.10*t_sales;
}
else if(t_sales<1800)
{
  commission=0.10*1000;
  commission=commission+(0.15*(t_sales-1000));
}
else
{
commission=0.10*1000;
commission=commission+(0.15*800);
commission=commission+(0.20*(t_sales-1800));
}
```

1. Snapshot for Total sales and commission when total sales are within 1000 and Invalid input

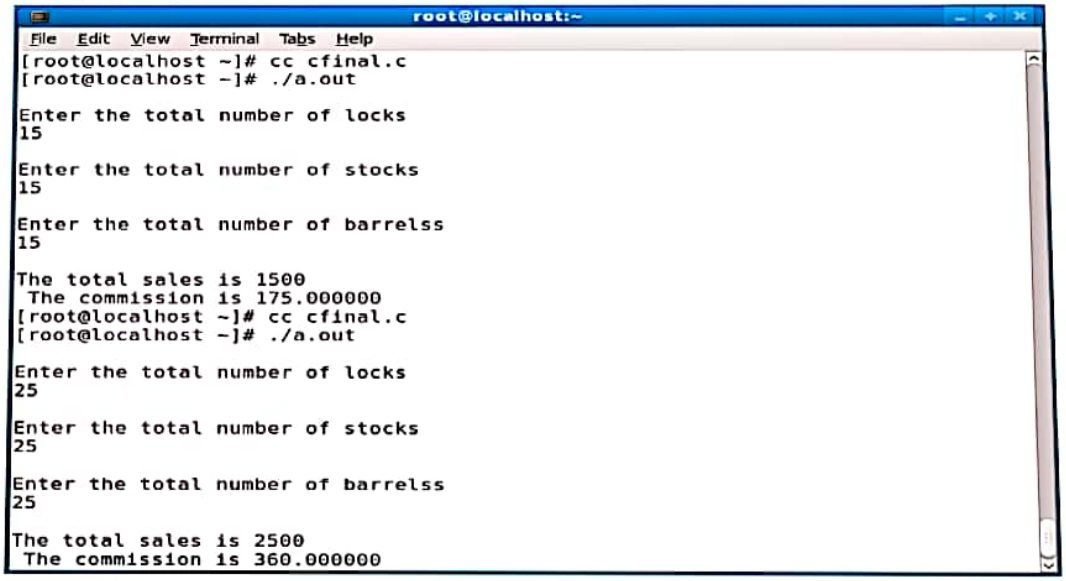


2. Invalid Input and Total sales and commission when total sales are within 1000

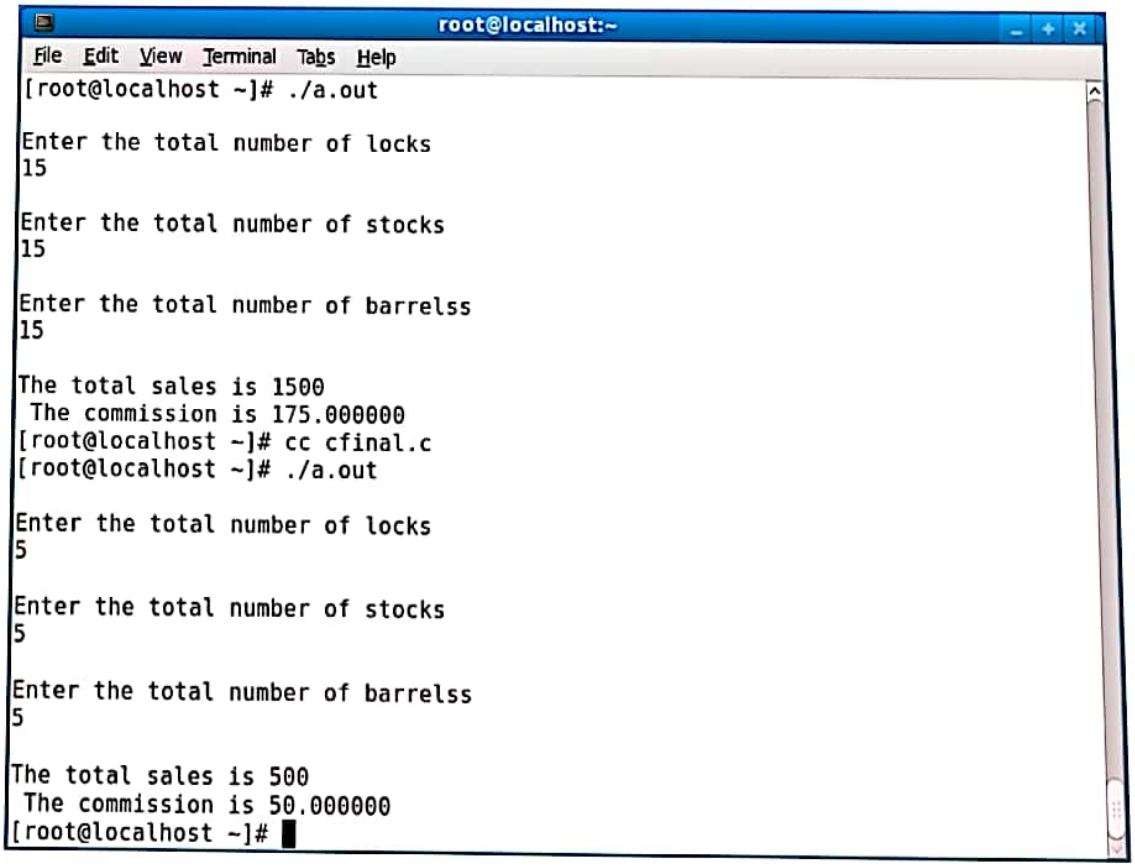


3. Snapshot for for Total sales and commission when total sales are within 1800 and to find out the total commission 360

3. Snapshot for for Total sales and commission when total sales are within 1800 and to find out the total commission 360



4. Snapshot for total sales and commission



```
printf("the total sales is %d & the commision is
%f", t_sales, commission);
    getch();
    return;
}
```

conclusion :- Executed the program & tested the test cases & the tables and compared the expected output with the actual output.

References :- paul c jorenhsen "Software testing".

Expt:- 05 Date:- 20-10-20

problem definition:- Design & develop, code & run the program in any suitable language to solve the commission problem. analyze it from the perspective of decision table based testing, derive different test cases, execute these test cases and discuss the test results.

Objectives :- To implement commission based problem based on decision table based testing

Theory :-
Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behaviour (output) are captured in a tabular format.

Design :-
From the given requirement we can draw the following conditions :
C1 : 1 ≤ locks ≤ 70 ? locks = -1 ?
C2 : 1 ≤ stocks ≤ 80 ?
C3 : 1 ≤ barrels ≤ 90 ?
C4 : sales > 1800 ?
C5 : sales > 1000 ?
C6 : sales ≤ 1000 ?

Algorithm :—
Step 1 :- Input 3 integers as locks, stocks, barrels
Step 2 :- Compute the total sales
(no of locks solds * 45) + (no of stocks sold * 30) + (no of barrels sold * 25)
Step 3 :- If a total sale in dollars is less than or Equal to $1800 then commission = 0.10 * total sales do Step 6
Step 4 :- Else if total sale is less than 1800
then commission = 0.10 * 1000.
do Step 6.

scanf ("%d", &a);

The decision table is given below :-

Condition	condition entries					
C1 : $1 \le$ locks ≤ 70 ?	F	T	T	T	T	T
C2 : $1 \le$ stocks ≤ 80 ?	-	F	T	T	T	T
C3 : $1 \le$ barrels ≤ 90 ?	-	F	F	T	T	T
C4 : Sales > 1800 ?	-	-	-	F	F	F
C5 : Sales > 1800 ?	-	-	-	-	T	F
C6 : Sales ≤ 1000 ?	-	-	-	-	-	T
actions	**action entries.**					
a1 : com = 0.10 * sales						X
a2 : com2 = com1 + 0.15 * (sales - 1000)				X		
a3 : com3 = com2 + 0.20 * (sales - 1800)			X			
a4 : out of Range.		X				

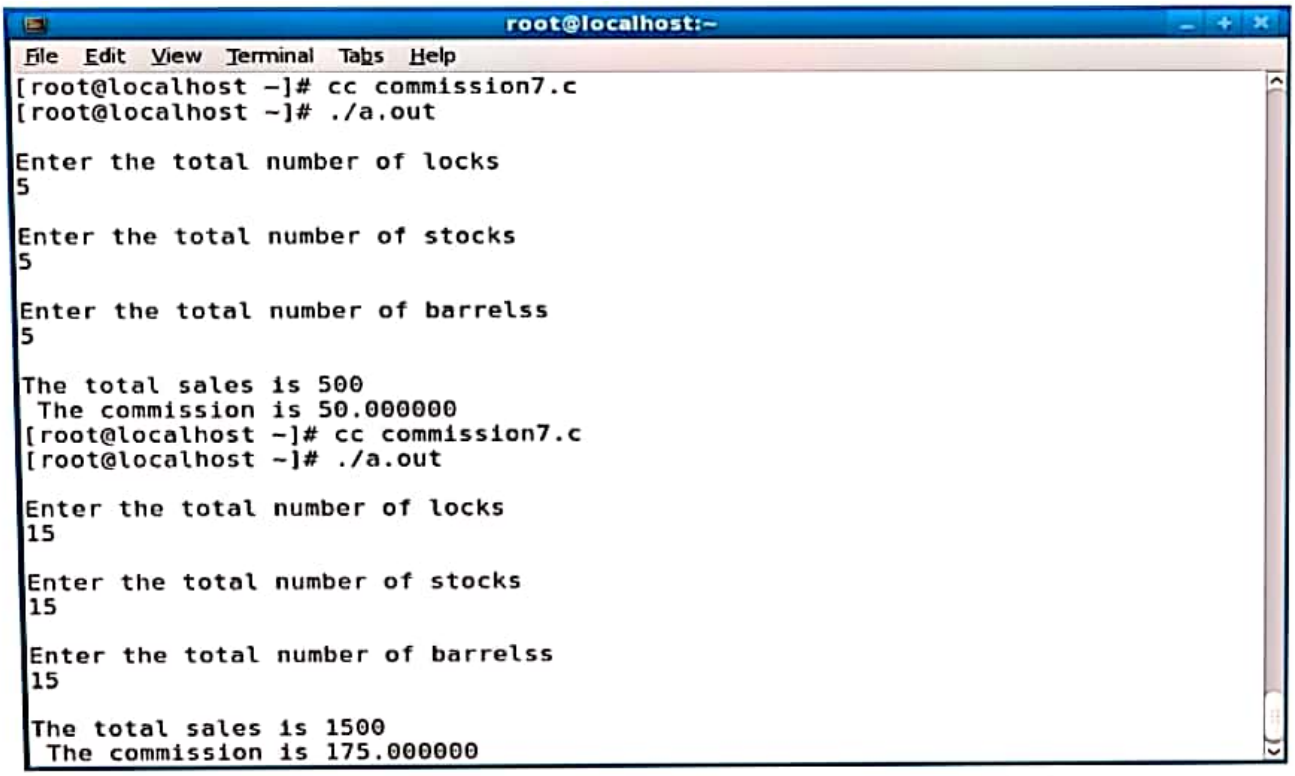
step 5 :- else commission1 = 0.10 * 1000
commission2 = commission1 + (0.15 * 800)
commission = commission2 + (0.20 * (total sales - 1800))
do step 6
step 6 : print commission
step 7 : stop.

program code :-

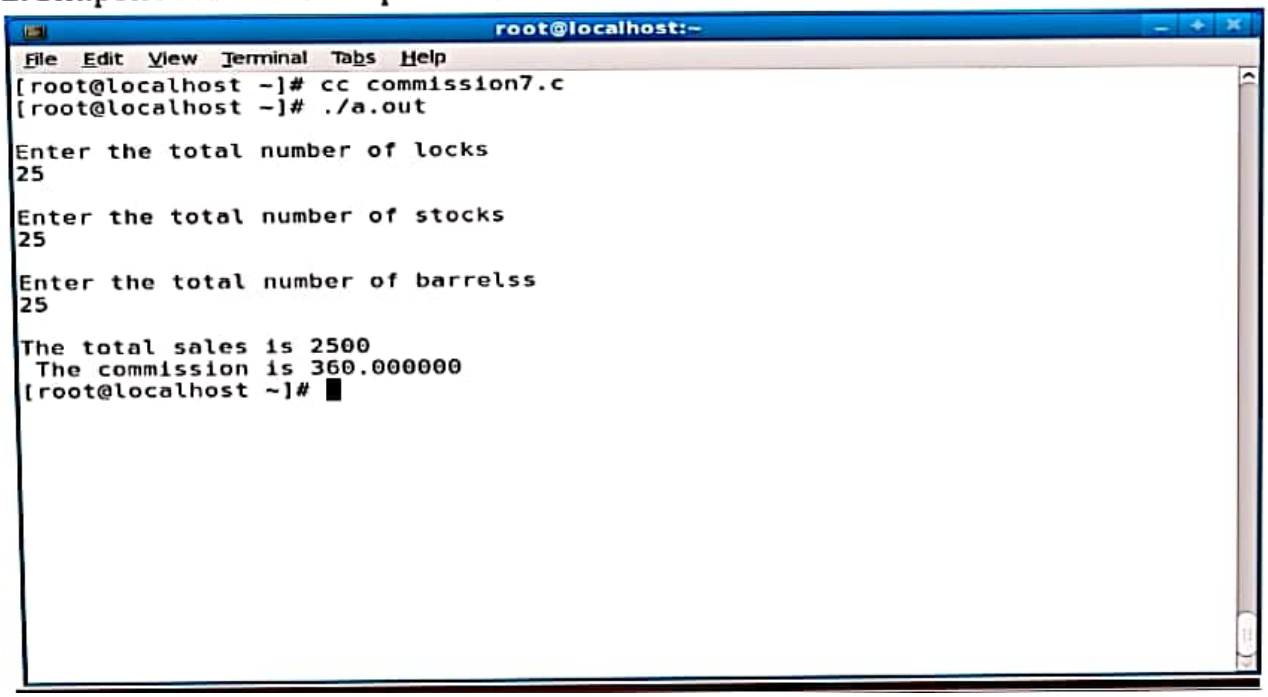
```
#include <stdio.h>
#include <conio.h>
int main()
{
  int locks, stocks, barrels, t_sales, flag = 0;
  float commission;
  printf("enter the total no of locks");
  scanf("%d", &locks);
  if ((locks <= 0) || (locks > 70))
  {
    flag = 1
  }
  printf("enter the total no of stocks");
  scanf("%d", &stocks);
  if ((stocks <= 0) || (stocks > 80))
  {
    flag = 1
  }
  printf("enter the total no of barrels");
  scanf("%d", &barrels);
  if ((barrels <= 0) || (barrels > 90))
  {
    flag = 1;
  }
  if (flag == 1)
  {
    printf("invalid input");
```

Test case using decision table :—

TCID	test case description	locks	stocks	barrels	Expected output	actual	status.
1	Testing for requirement (R) validation (C1).	-2	40	45	out of range	out of range	pass
2	Testing for R1 & C1	90	40	45	out of range	out of range	-
3	Testing for R1 & C2	35	-3	45	out of range	out of range	-
4	Testing for R1 & C2	35	100	45	out of range	out of range	-
5	Testing for R1 & C3	35	40	-10	out of range	out of range	-
6	Testing for R1 & C3.	35	40	150	out of range	out of range	-
7	Testing for requirement 2	5	5	5	500	a1:50	
8	Testing for requirement 2	15	15	15	1500	a2:175	
9	Testing for requirement 2	25	25	25	2500	a3:360	



2. Snapshot when the inputs all are 25.



```
getch();
exit(0);
}
t_sales = (locks * 45) + (stocks * 30) + (barrels * 25);
if (t_sales <= 1000)
{
    commission = 0.10 * t_sales;
}
else if (t_sales < 1800)
{
  commission = 0.10 * 1000;
  commission = commission + (0.15 * (t_sales - 1000));
}
else
{
  commission = 0.10 * 1000;
  commission = commission + (0.15 * 800);
  commission = commission + (0.20 * (t_sales - 1800));
}
printf("The total sales is %d the commission is %f",
t_sales, commission);
getch();
return;
}
```

conclusion :- ——— we Implemented the decision based table and compared with the expected and actual results.

Reference : paul c jorgensen "software testing"

PART B

EXPT NO :- 6

Date :- 10-11-20

problem definition :- Design and develop, code and run the program in any suitable language to implement the Binary Search algorithm. Determine the basis path and using these derive different test cases, executes these test cases and discuss the test results

Objectives: To implement binary search algorithm & determine the basis paths.

Theory :—

* Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you have narrowed down the possible locations to just one

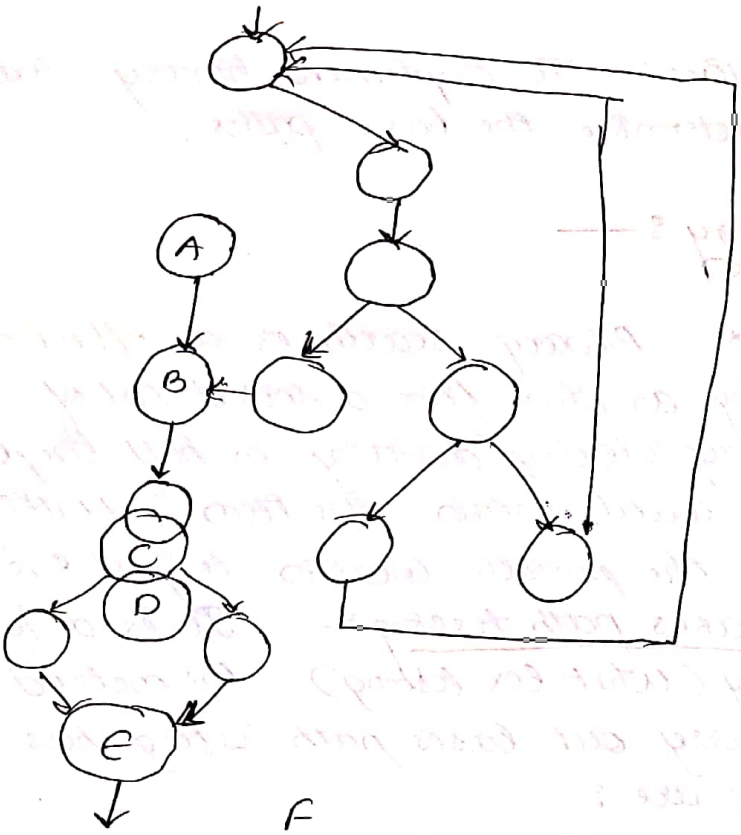
* Basis path testing :- It is a form of structural testing (White box testing). The method devised by mccabe to carry out bases path testing has 4 steps. These are :

1) Compute the program graph
2) Calculate the cyclomatic complexity
3) Select a basis set of paths.
4) Generate the test cases for each of these paths

The cyclomatic complexity of a connected graph is provided by the formula $V(G) = e - n + 2p$ The number of edges is represented by e, the number of nodes by n & the number of connected regions by p. If we apply this formula to the graph given below, the number of linearly independent circuit is

Number of edges = 21
number of nodes = 15

DD path graph for binary search.



Number of connected regions = 1

$25 - 15 + 2(1) = 4$.

* Algorithm :-

Step1 : Enter 'n' integer.
Step2 :- l = 0, h = n-1
Step3 : until(l <= h) do
mid = (l+h)/2
if (a[mid] == key)
then do step5
else if (a[mid] > key)
then do
h = mid - 1
else
l = mid + 1
Step4 :- unsuccessful press do step6
step5 :- successful press.
Step6 :- stop

* program :-

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a[20], n, l, mid, key, i, flag = 0;
    printf("Enter the value for n");
    scanf("%d", &n);
    if (n>0)
    {
        printf("Enter %d Elements in Ascending order");
    for(i=0; i<n; i++)
    {
        scanf("%d", &a[i])
    }
```

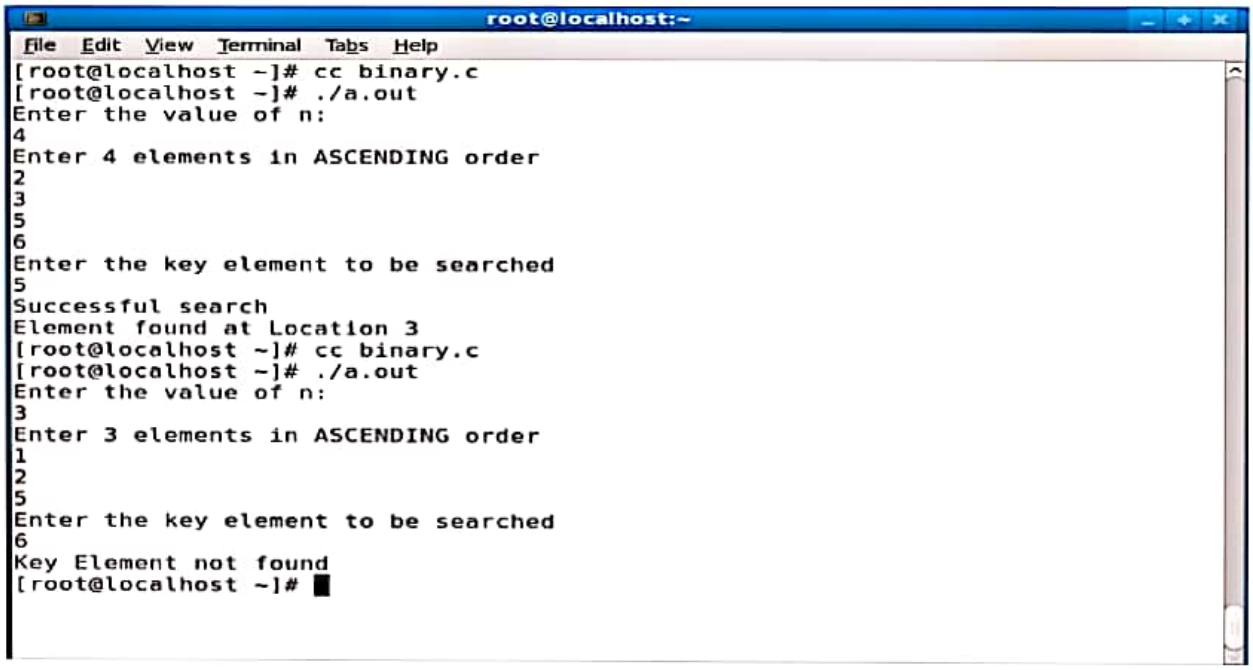
Test cases :-

TC. ID	Test case description	value for 'n'	array elements	Key	Expected output	Actual output	status
1	Testing for requirement path 1	0	- -	5	Key not found	Key not found	pass
2	Testing for R2, path 2	4	2, 3, 5, 6, 7	5	found at position 3	found at position 3	pass
3	Testing for R2, path 3	3	1, 2, 5	6	not found	not found	pass
4	Testing for R2, path 4	3	1, 2, 5	8	not found	not found.	pass
5	testing for R2, P4+P2-1	5	1, 2, 4, 6, 7	2	found at position 2	found at position 2	pass.
6	Testing for R2, path 3, P3-P1	5	4, 5, 7, 8, 9	8	found at position 4	found at position 4	pass

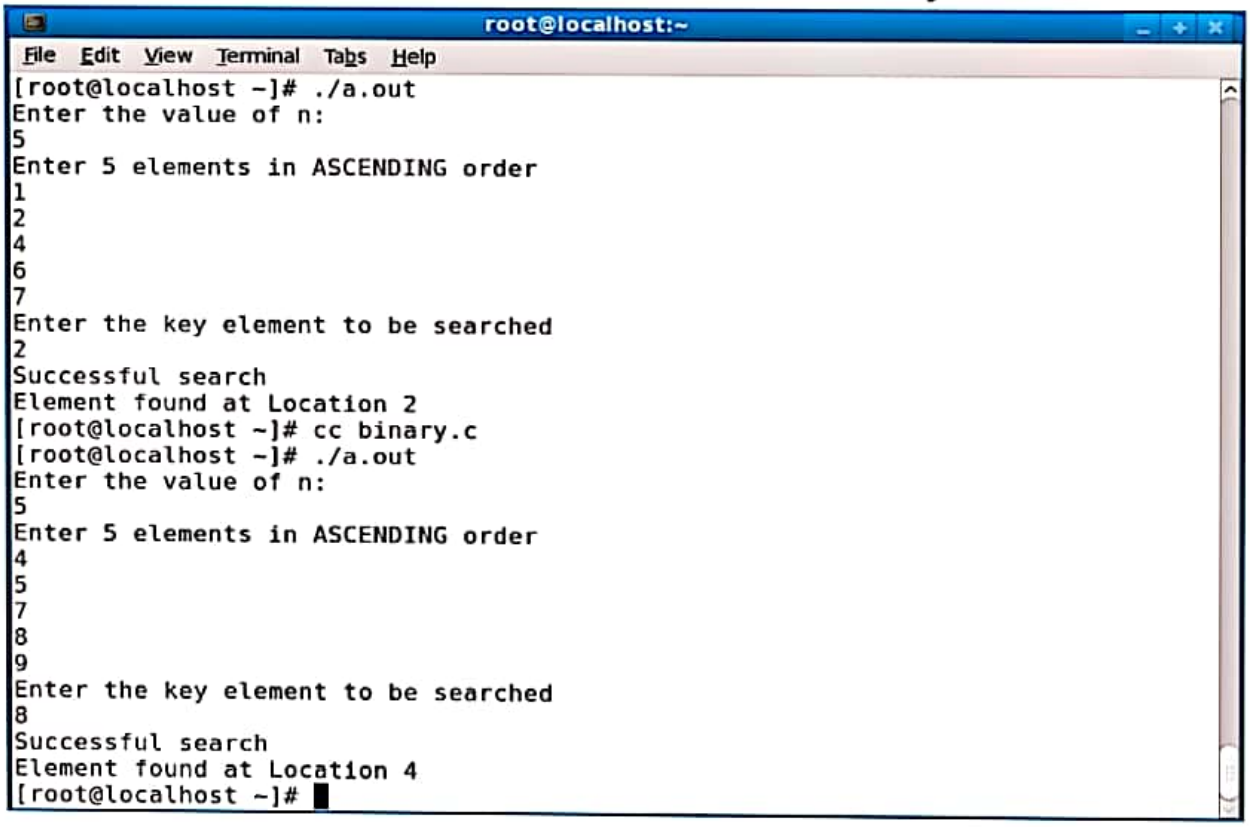
Test report :-

No of TC's executed :- 06
No of defects raised :- 0
No of TC's passed :- 06
no of TC's failed :- 0.

```
printf("Enter the key elements to be searched");
scanf("%d", &key);
l=0; h=n-1;
while(l<=h)
{
    mid = (l+h)/2;
    if(a[mid]==key)
    {
        flag=1;
        break;
    }
    else if(a[mid]<key)
    {
        l = mid+1;
    }
    else
    {
        h = mid-1;
    }
}
if(flag==1)
    printf("Successful search Element found at position
    %d", mid+1);
else
    printf("not found");
}
else
    printf("wrong input");
getch();
return 0;
}
```



3. Snapshot to check successful search and not found key element.



Conclusion:- Implemented the binary search algorithm & determined the basisc paths and also the executed the test cases and compared with the test cases & actual output

References: paul c jorgensen, "software testing" 3rd edition onwards.

Date:- 15-11-20

Expt no:- 7

problem definition :— Design, develop, code & run the program in any suitable language to implement the quicksort algorithm. Determine basic paths and using them derive different test cases. Execute these test cases and discuss the test results.

Objectives:- Implement the quicksort algorithm & determine the basis paths.

Theory :—

quick sort is a divide and conquer algorithm. it works by selecting a "pivot" element from the array and partitioning the other elements into 2 sub arrays. according to whether they are less than or greater than the pivot. The sub arrays are then sorted recursively.

requirement :—

R1 :- The system should accept 'n' number of elements & key element to be searched.

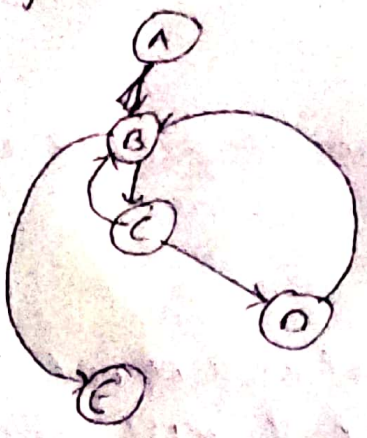
R2 :- If key element is present successful else unsuccessful.

Algorithm :

```
quicksort (arr[], low, high)
{
    if (low < high)
    {
        pivot = partition (arr, low, high);
        quicksort (arr, low, pi-1);
        quicksort (arr, pi+1, high);
    }
}

partition (arr[], l, h)
{
    pivot = arr[h];
    i = (l-1)
    for (j = l; j <= h-1; j++)
    {
        if (arr[j] < pivot)
        {
```

DD path graph:—



cyclomatic complexity:-

no of edges = 6

no of nodes = 5

e - n + 2

6 - 5 + 2 = 3

Test cases:

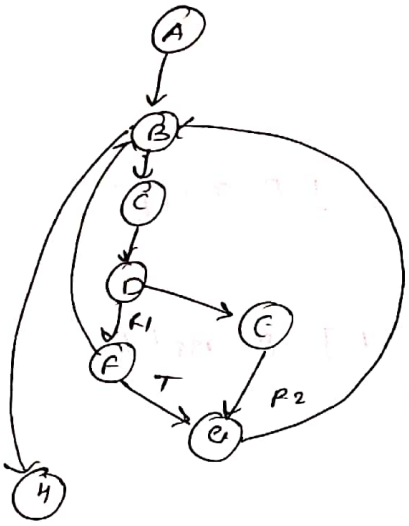
TC id	TC description	Array elements	Expected array	output value	Actual output	status
1	Testing for path p1	5	5	0	0	pass
2	testing for path p2	5, 4, 6, 7, 7	5, 4, 6, 7, 7	4	4	pass
3	Testing for path p3	5, 6, 6, 7, 5	5, 6, 6, 7, 5	0	0	pass

```
        i++;
        swap arr[i] & arr[j]
    }
}
swap arr[i+1] & arr[h]
return (i+1)
}
```

Program :-

```
#include<stdio>
void swap(int *a, int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
int partition(int a[], int l, int h)
{
    int x = a[h];
    int i = (l-1), j;
    for(j=i; j<=h-i; j++)
    {
        if(a[i] <= x)
        {
            i++;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i+1], &a[h]);
    return(i+1);
}
```

DD path graph:-



no of nodes: 8
no of edges: 10
e - n + 2
10 - 8 + 2 = 4.

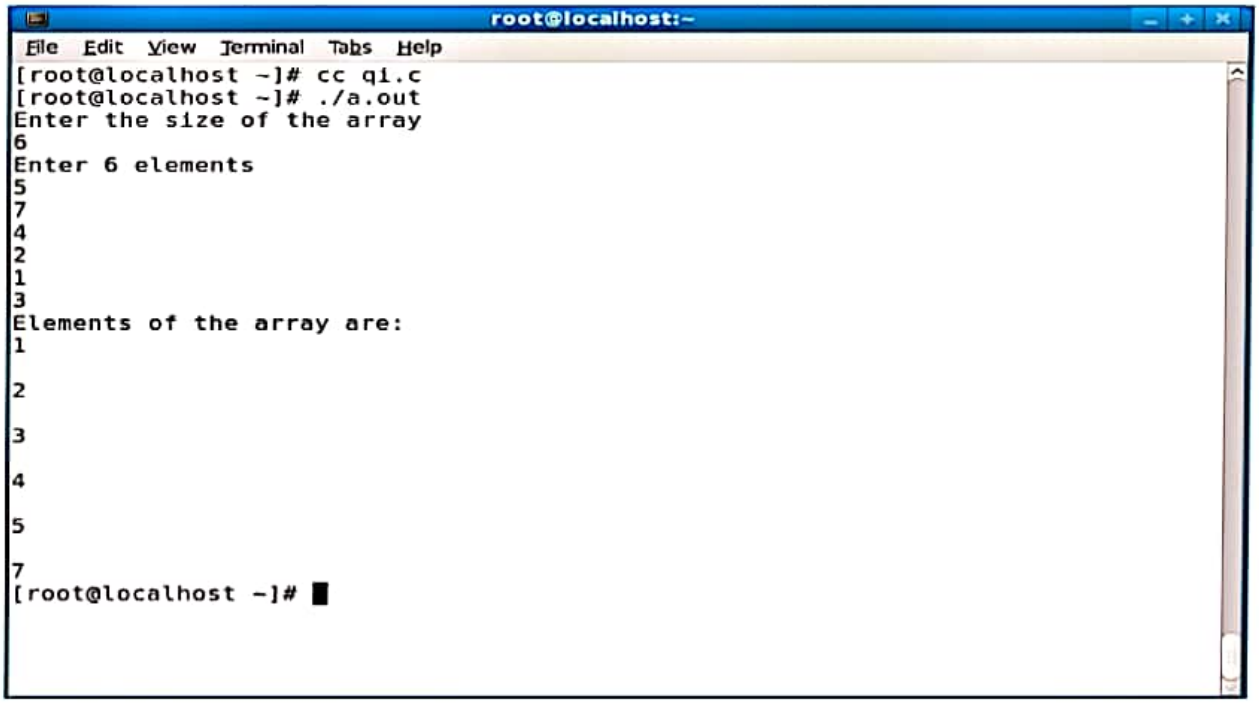
Test cases :-

TC Id	TC description	array	Expected output	Actual output	status
1	testing for path 1	5, 7, 4, 2, 1, 3	2, 1, 3, 5, 7	2, 1, 3, 5, 7	pass
2	Testing for path 2	5, 4, 8, 2, 7	5, 4, 2, 7, 8	—	—
3	Testing for path 3	5, 4, 6, 7, 3	3, 4, 6, 7, 5	3, 4, 6, 7, 5	pass.

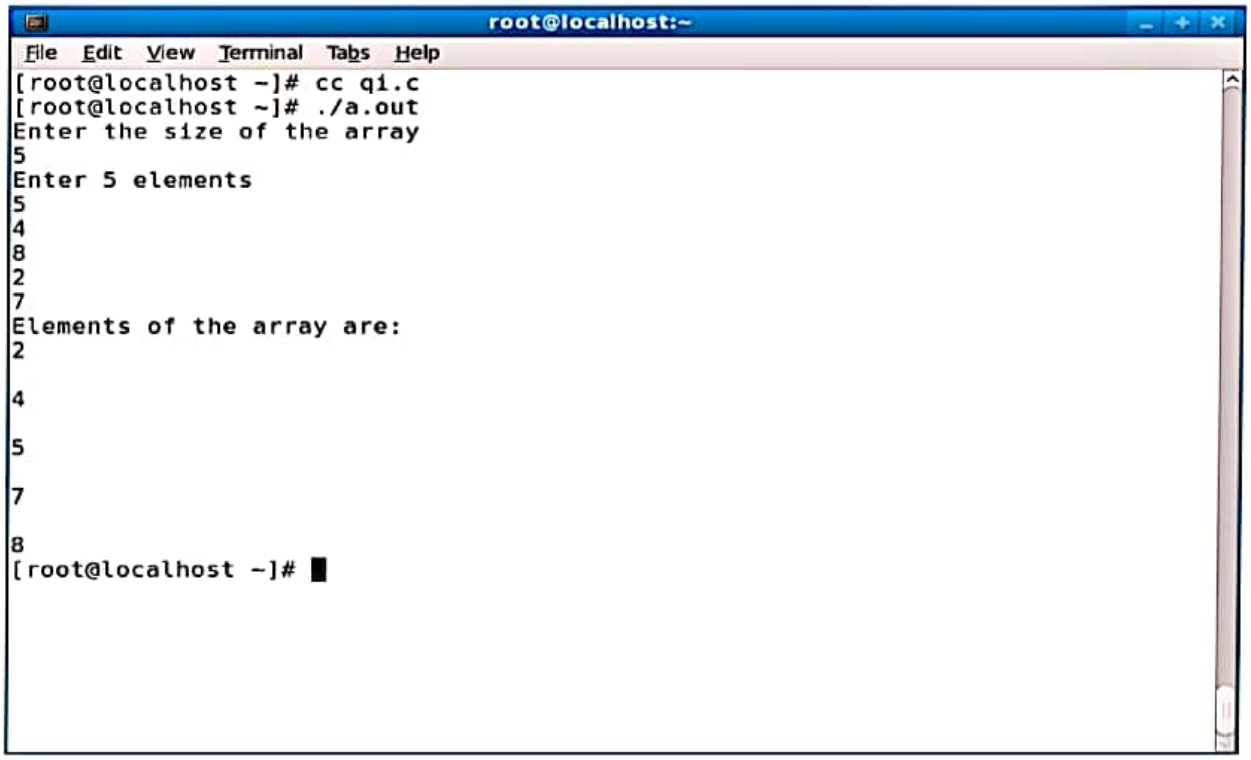
```
void quicksortIterative(int a[], int l, int h)
{
    int stack[10], p;
    int top = -1;
    stack[++top] = l;
    stack[++top] = h;
    while(top >= 0)
    {
        h = stack[top--];
        l = stack[top--];
        p = partition(a, l, h);
        if(p-1 > l)
        {
            stack[++top] = l;
            stack[++top] = p-1;
        }
        if(p+1 < h)
        {
            stack[++top] = p+1;
            stack[++top] = h;
        }
    }
}

int main()
{
    int a[20], n, i;
    printf("Enter the size of the array");
    scanf("%d", &n);
    printf("Enter the elements");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
```

1. Snapshot of quick sort sorted elements are displayed, when the n=6



2. Snapshot of quick sort sorted elements are displayed, when the n=5



```
quicksort&iterate (a, 0, n-1);
printf("Elements of the array are ");
for(i=0; i<n; i++)
    printf("%d", a[i]);
getch();
return 0;
}
```

Conclusion :— Implemented the quick sort algorithm, with respect to basis path and also the expected output is compared with actual output

References :- paul c jorgenson "software testing" 3rd edition.

Expt no :- 8 Date :- 20-11-20

problem definition :— design, develop, code and run the program in any suitable languages to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different test cases, execute test case and discuss the test results.

Objective:- Implement an absolute letter grading procedure making suitable assumption & determine the basis parts

Theory:-

R1: The system should accept marks of 6 subjects, each marks in the range 1 to 100.

i.e for example:- 1<= marks <=100

1 <= kannada <=100

1<= maths <=100 etc.

R2: If R1 is satisfied compute average of marks scored and percentage of the same and depending on percentage display the grade.

algorithm:-

step1:- Enter all the marks

step2:- if (35 &&) >0 this condition makes as a fail. else step 3

step3:- avmar <=40 && avmar >35 make it as grade c

avmar <=50 && avmar >40 make it as grade c+

avmar <=60 && avmar >50 make it as grade B

avmar <=70 && avmar >60 make it as grade B+

avmar <=80 && avmar >70 make it as grade A

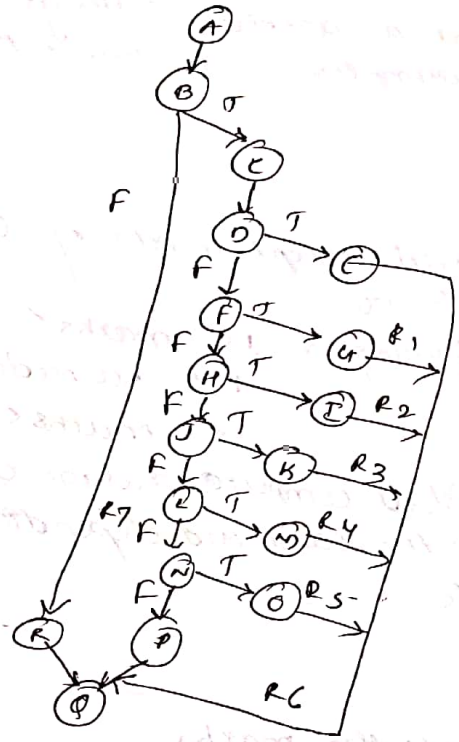
avmar <=100 && avmar >80 make it as grade A+

program:-

```
#include <stdio.h>
main()
```

Cyclomatic complexity :——→

DD path graph



no of edges : 24

no of no nodes = 18.

e − n + 2

24 − 18 + 2 = 8.

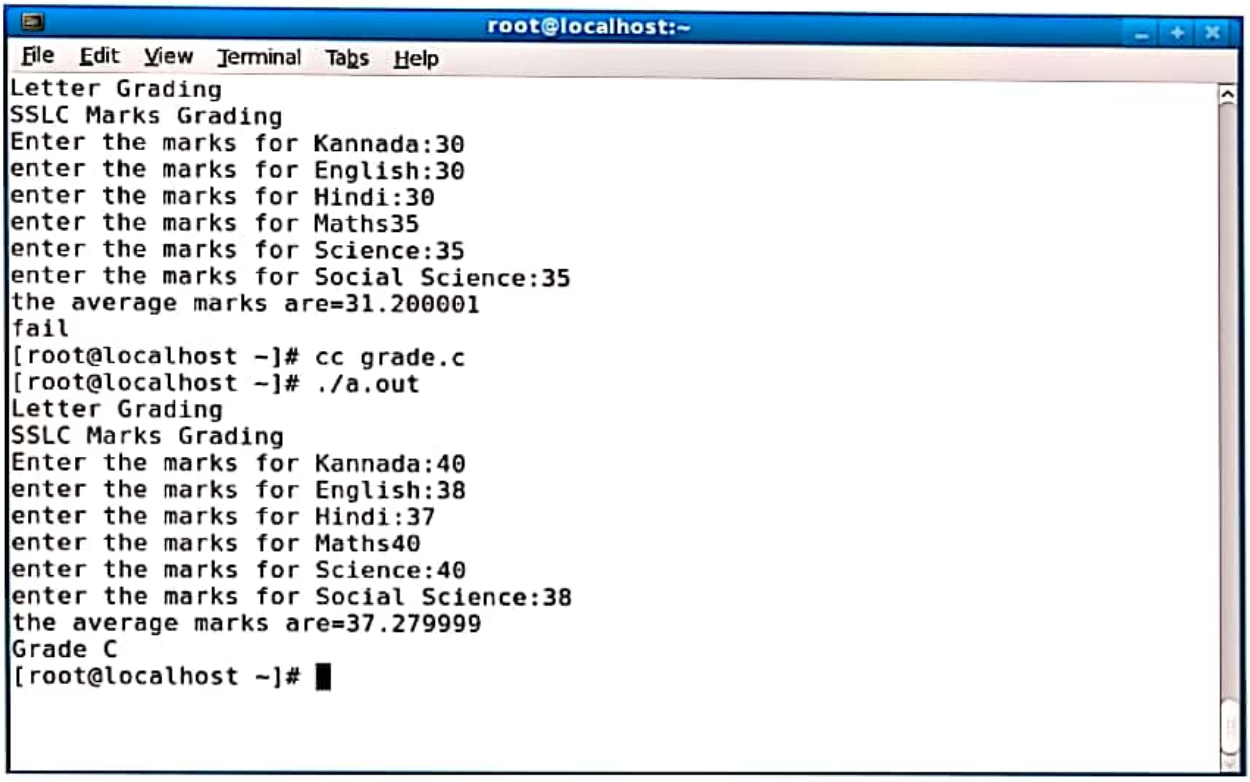
5

```
float kan, eng, hindi, maths, science, sst, avmar;
printf("Letter grading");
printf("SSLC marks grading");
printf("Enter the marks for Kannada");
scanf("%f", &kan);
printf("Enter the marks for English");
scanf("%f", &eng);
printf("Enter the marks for hindi");
scanf("%f", &hindi);
printf("Enter the marks for maths");
scanf("%f", &maths);
printf("Enter the marks for science");
scanf("%f", &science);
printf("Enter the marks for sst");
scanf("%f", &sst);
printf("Enter the. avmar = (kan+eng+hindi+maths+science+
                    sst)/6.25;
printf("The average marks are =%f", avmar);
if(avmar<35)&&(avmas>0))
print("fail");
else if((avmar<=40)&&(avmas>35))
print("Grade C");
else if((avmar<=50)&&(avmar>40))
print("grade C+");
else if((avmar<=60)&&(avmar>50))
print("Grade B");
else if((avmar<=70)&&(avmar>60))
print("grade B+");
```

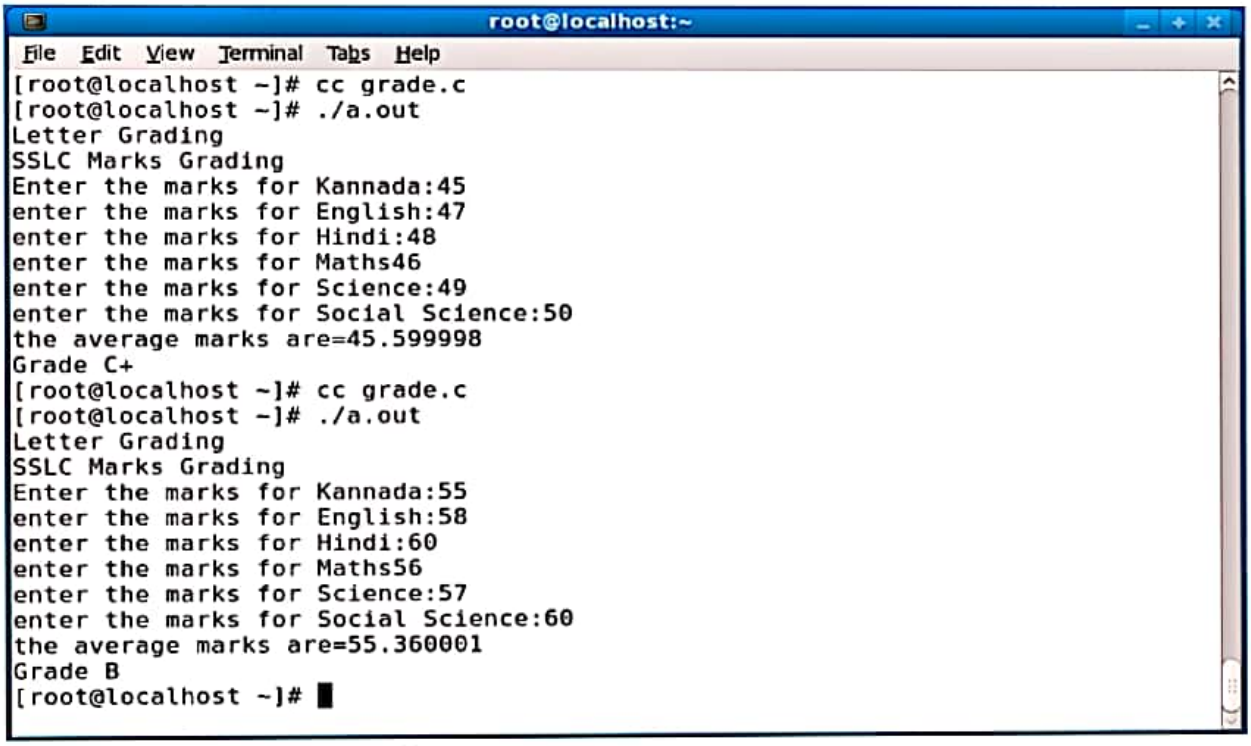
Test cases :—

TCId	test description	Input	expected output	Actual output	status
1)	Testing for path p1	K = 50 E = 50 H = 50 M = 50 S = 50 SST = 150	Invalid Input.	-	—
2)	Testing for path p2	K = 30 E = 30 H = 30 M = 35 S = 35 SST = 35 Avg = 32.5	grade fail	fail	pass
3)	Testing for path p3	K = 40 E = 38 H = 37 M = 40 S = 40 SST = 38 Avg = 38.83	grade C	grade C	pass
4)	Testing for path p4	K = 45 E = 47 H = 48 M = 46 S = 49 SST = 50 avg = 47.5	Grade C+	grade C+	pass
5)	Testing for path p5	K = 55 E = 58 H = 60 M = 56 S = 57 SST = 60 avg = 57.66	Grade B	grade B	pass
6)	Testing for path p6	K = 65 E = 65 H = 65 M = 65 S = 65 SST = 65 avg = 65.0	Grade B+	grade B+	pass
7)	Testing for path p7	K = 75 E = 72 H = 78 M = 75 S = 86 SST = 80 avg = 76.6	Grade A	Grade A	pass
8)	testing for path p8	K = 85 E = 90 H = 80 M = 85 SST = 95 avg = 86.66.	Grade A+	grade A+	pass

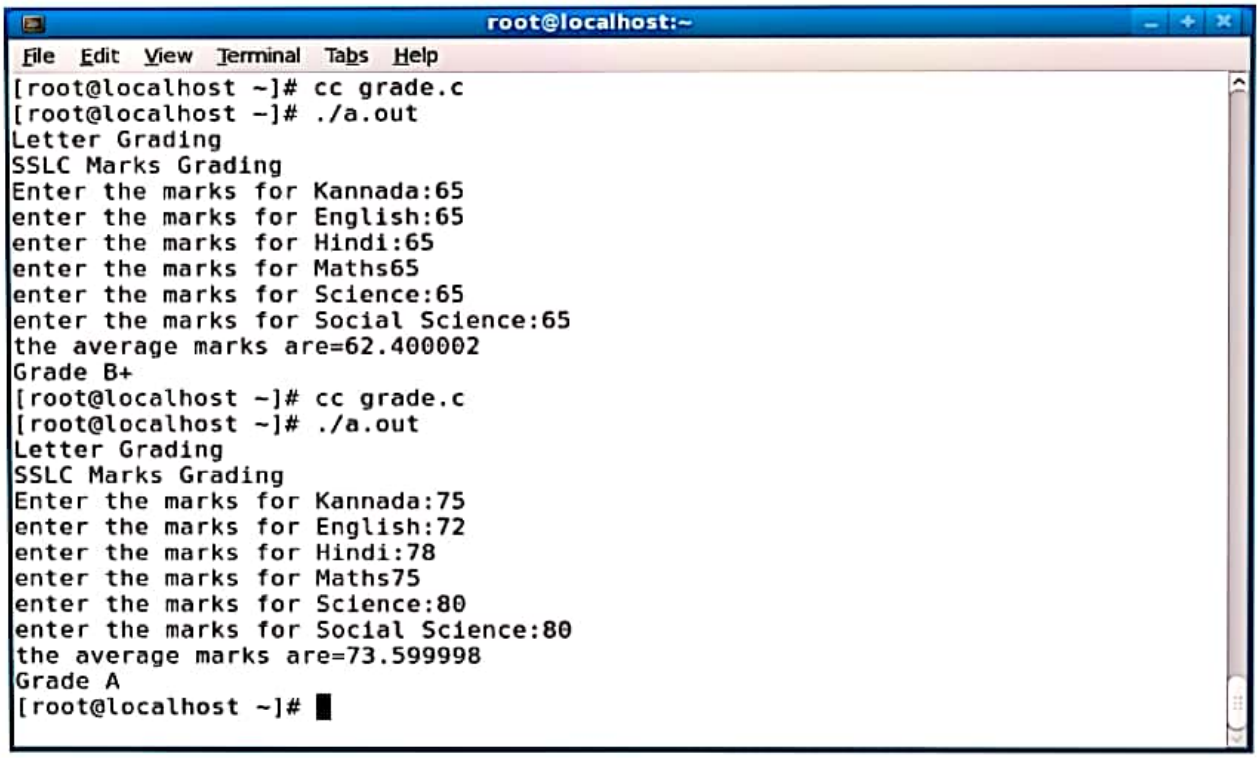
1. Snapshot to Show Fail and Grade C



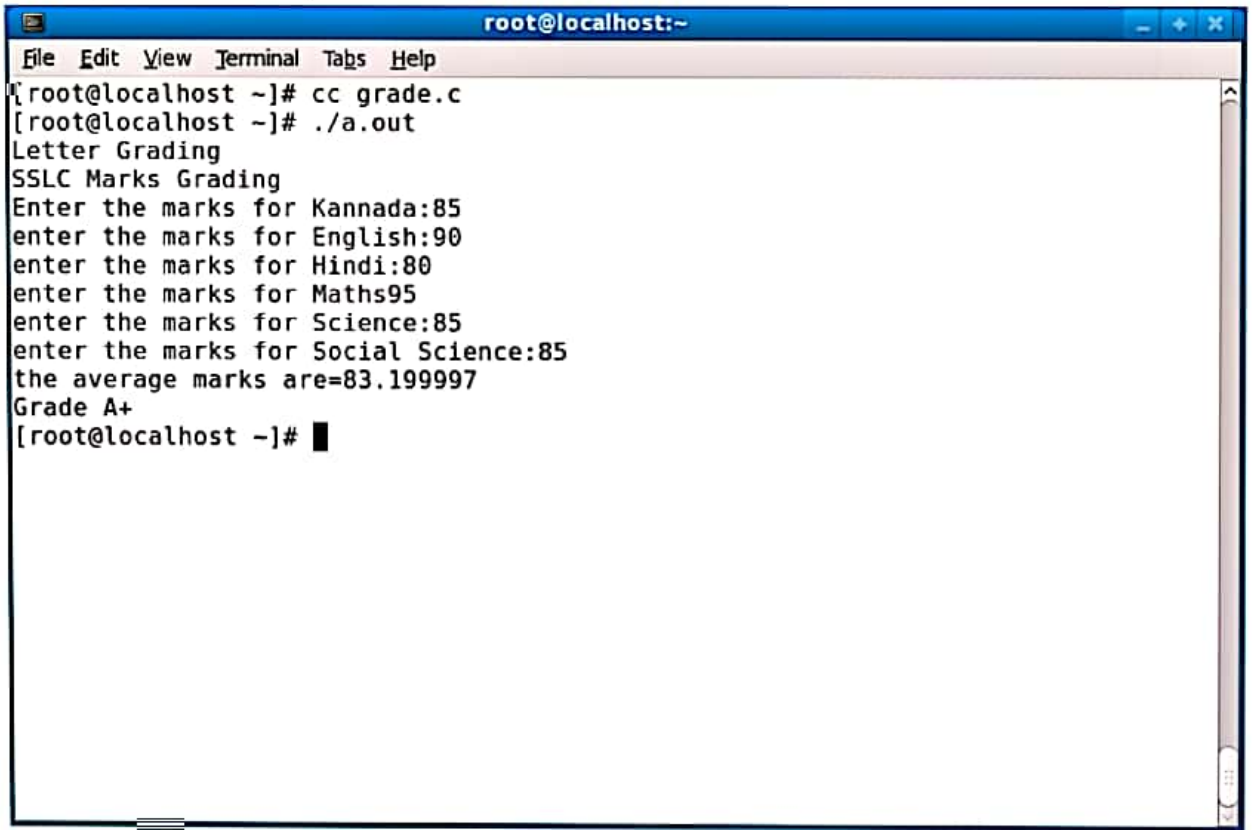
2. Snapshot to show Grade B and Grade C+



4. Snapshot to show the Grade A and Grade B+



4. Snapshot to show the Grade A+



```
else if((avmax <= 80) && (avmax>=70))
printf("grade A");
else if ((avmax <=100) && (avmax>80))
printf("grade A+");
}

getch();
return 0;
}
```

Test report :-

no of TC executed :- 8
no of TC pass :- 8
no of defect :- 0
no of TC fail :- 0.

conclusion :- If letter grading program is Implemented as per the definition and also the program is been verified with the actual result and Expected result

References :- paul c jorchenu "Software testing" 3rd edition onwards.

Expt no 8-9 Date:-28-11-20

problem definition:- design, develop, code and run the program in any suitable language to implement the next date function. Analyze it either from the perspective of equivalence class value testing derive different test cases, execute these test case and discuss the test results.

Objectives :- To Implement the nextdate function and in the form of perspective of equivalence class value testing.

Theory :-

- Nextdate is a function consisting of three variables like month, date & year. It returns the date of next day as output. It reads current date as input date.

The constraints are :-

C1 : $1 \leq month \leq 12$

C2 : $1 \leq day \leq 31$

C3 : $1812 \leq year \leq 2012$

- If any of these condition fails as C1, C2 or C3 the function displays out of range.
- Since many combinations of dates can exist, hence we can simply displays one message for this function "Invalid input date".

* Equivalence class testing :- It allows a user to subdivide the input domain into a relatively small number of domain says n+1.

Algorithm :-

Step1 : Input DD : mm . yy

Step2 : If mm is 01, 03, 05, 07 10, do Step 3 else Step 6.

Step3 : If DD < 31 then do Step 4 else If DD = 31 do Step 5 else o/p

Step4 : tomorrow = 1, tomorrow month = month + 1 goto Step18

Step5 : If mm is 04, 09, 11 do Step 7.

Step 7 : if dd < 30 then do step 4 else if DD = 30 do step 5
Step 8 : if mm is 12
Step 9 : if DD < 31 then step 4 else step 10.
Step 10 :- tomorrow day = 1, tommorow month = 1, tommorow year = yyyy + 1 goto step 18.
Step 11 : if mm is 2
Step 12 : if DD < 28 do step 4
Step 13 : if DD = 28 & yyyy is a leap
Step 14 : tommorowday = 1, tommorowmonth = 18
Step 15 : tommorowday = 1, tommorowmonth = 3 goto step 18
Step 16 : if DD = 29 then do step 15
Step 17 : output ("cannot have feb") step 19
Step 18 : output (tommorow day, tommorowmonth, tommorowyear)
Step 19 : Exit

program :-

```
#include <stdio.h>
#include <conio.h>
main()
{
int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int d, m, y, nd, nm, ny, ndays;
printf("Enter date, month, year");
scanf("%d%d%d", &d, &m, &y);
ndays = month[m-1];
if (y <= 1812 && y > 2012)
{
  printf("Invalid input year");
  exit(0);
}
```

TCid	TC description	Input data mm	dd	yy	Expected output	Actual output	status.
1	Testing for valid d/f changing the day within the month	6	15	1900	6/16/1900	6/16/1900	pass
2	Testing for invalid day	6	-1	1900	Day not in range	Day not in range	pass.
3	Testing for invalid day.	6	32	1900	Day not in range	Day not in range	pass.
4	testing for invalid month	-1	15	1900	month ~~Day~~ not in range	month not in range	pass.
5	Testing for invalid month	13	15	1900	month not in range	month not in range	pass
6	Testing for year.	6	15	1899	year not in range	year not in range	pass.

```
if (d<=0 || d>ndays)
{
printf("Invalid input day");
exit(0);
}
if(m<1 && m>12)
{
printf("printf -Invalid month");
exit(0);
}
if (m==2)
{
if (y%1000==0)
{
if (y%400==0)
ndays=29;
}
else
if(y%4==0)
ndays=29;
}
nd=d+1;
nm=m;
ny=y;
if(nd>days)
{
nd=1;
nm++;
}
if(nm>12)
{
nm=1;
ny++;
}
```

```
File  Edit  View  Terminal  Tabs  Help
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
6
1900

 Given date is 15:6:1900

 Next days date is 16:6:1900
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
-1
6
1900
Invalid Input Day
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
32
6
1900
Invalid Input Day
[root@localhost ~]#
```


2. Invalid Input

```
root@localhost:~
File  Edit  View  Terminal  Tabs  Help
[root@localhost ~]# cc nextdate2.c
[root@localhost ~]# ./a.out
enter the date,month,year
15
-1
1900
Invalid Input Day
[root@localhost ~]#
```

```
printf("given date is %d %d %d", d, m, y);
printf("next day's date is %d %d %d", nd, nm, ny);
getch();
}
```

Conclusion:- Implemented the nextdate function with equivalence class testing. and also verified the testcase with the actual output.

Reference:- paul C Jorgensen "software testing" 3rd edition.