10. Design, develop, code and run the program in any suitable language to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

Requirements

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

i.e for example : 1<=marks<=100
1<=kannada<=100
1<=maths<=100

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

Design

We use total percentage pf marks to grade the student marks.

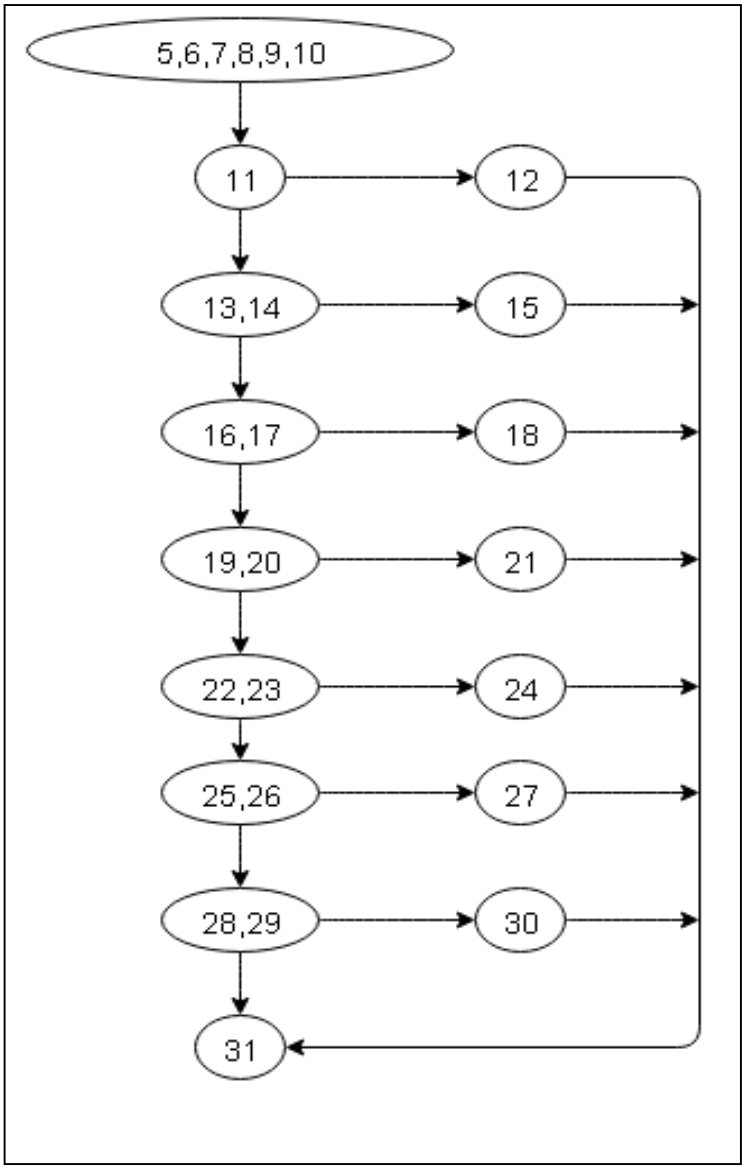
- <35 && >0 of percentage make it as Fail.
- avgmar <=40 && avgmar>35 make it as Grade C
- avgmar <=50 && avgmar>40 make it as Grade C+
- avgmar <=60 && avgmar>50 make it as Grade B
- avgmar <=70 && avgmar>60 make it as Grade B+
- avgmar <=80 && avgmar>70 make it as Grade A
- avgmar <=100 && avgmar>80 make it as Grade A+

Program Code:

```
1 #include<stdio.h>
2 void main()
3 {
4       float kan,eng,hindi,math,science,social,avgmar;
5       printf("Letter Grading\n");
6       printf("SSLC marks grading\n");
7       printf("enter the marks for all subject");
8       scanf("%f%f%f%f%f%f",&kan,&eng,&hindi,&math,&science,&social);
9       avgmar=(kan+eng+hindi+math+science+social)/6.25;
10      printf("The average marks are %f\n",avgmar);
11      if((avgmar<35)&&(avgmar>0))
12      printf("fail");
13      else
14      if((avgmar<=40)&&(avgmar>35))
15      printf("Grade C");
16      else
17      if((avgmar<=50)&&(avgmar>40))
18      printf("Grade C+");
19      else
20      if((avgmar<=60)&&(avgmar>50))
21      printf("Grade B");
22      else
23      if((avgmar<=70)&&(avgmar>60))
24      printf("Grade B+");
```

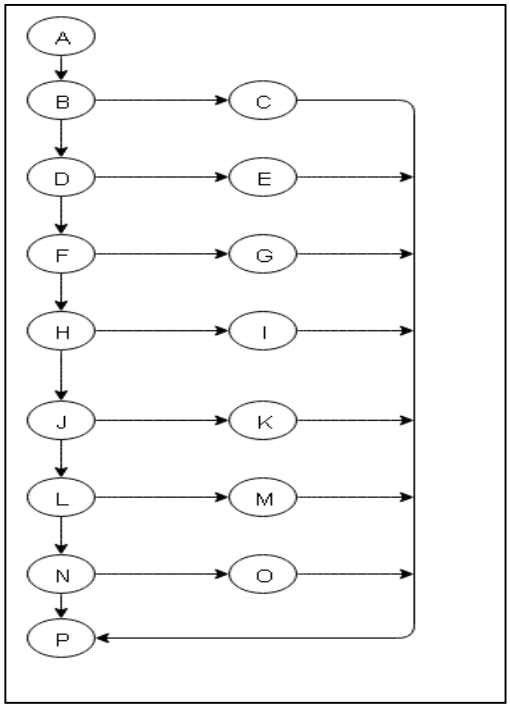
```
25      else
26      if((avgmar<=80)&&(avgmar>70))
27      printf("Grade A");
28      else
29      if((avgmar<=100)&&(avgmar>80))
30      printf("Grade A+");
31 }
```

Program Graph :



DD-Path graph:-

DD path name	Program graph nodes
A	5,6,7,8,9,10
B	11
C	12
D	13,14
E	15
F	16,17
G	18
H	19,20
I	21
J	22,23
K	24
L	25,26
M	27
N	28,29
O	30
P	31



Cyclomatic Complexity V(G)=e-n+p

Edges =23, Nodes=16, Regions=1

V(G)=23-16+1=8

According to cyclomatic complexity 8 feasible basis path exists:

Path1: A,B,D,F,H,J,L,N,P (Infeasible path)

Path2: A,B,C,P (fail)

Path3: A,B,D,E,P (Grade C)

Path4: A,B,D,F,G,P (Grade C+)

Path5: A,B,D,F,H,I,P (Grade B)

Path6: A,B,D,F,H,J,K,P (Grade B+)

Path7: A,B,D,F,H,J,L,M,P (Grade A)

Path8: A,B,D,F,H,J,L,N,O,P (Grade A+)

Test Cases:

TC ID	Description	Input	Expected output	Actual output	Status

1	Testing for path1	K=50,E=150,H=50, M=50,SC=60,SS=50	Invalid Input		
2	Testing for path2	K=30,E=30,H=30, M=30,SC=30,SS=30	Fail		
3	Testing for path3	K=40,E=38,H=37, M=40,SC=40,SS=38	Grade C		
4	Testing for path4	K=45,E=46,H=47, M=46,SC=49,SS=50	Grade C+		
5	Testing for path5	K=55,E=58,H=57, M=56,SC=59,SS=60	Grade B		
6	Testing for path6	K=65,E=65,H=65, M=65,SC=65,SS=65	Grade B+		
7	Testing for path7	K=72,E=75,H=75, M=78,SC=80,SS=80	Grade A		
8	Testing for path8	K=85,E=80,H=90, M=85,SC=95,SS=90	Grade A+		

Test Report :

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed

9. Design, develop, code and run the program in any suitable language to implement the quicksort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

```
#include<stdio.h>
void quicksort(int x[10],int first,int last)
{
        int
        temp,pivot,i,j;
        if(first<last)
        {
                pivot=first;
                i=first;
                j=last;
                while(i<j)
                {
                        while(x[i]<=x[pivot] &&
                        i<last) i++;
                        while(x[j]>x[pivot])
                        j--;
                        if(i<j)
                        {
                                temp=x[i];
                                x[i]=x[j];
                                x[j]=temp;
                        }
                }
                temp=x[pivot];
                x[pivot]=x[j];
                x[j]=temp;
                quicksort(x,first,j-
                1);
                quicksort(x,j+1,last)
                ;
        }
}

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

```
                scanf("%d",&a[i])
                ; quicksort(a,0,n-
                1);
                printf("the elements in the sorted array is:\n");
                for(i=0;i<n;i++)
                printf("%d\t",a[i]);
        }
        else
        {
                printf("size of array is  invalid\n");
        }
}
```

Quick sort function with line number

```
void quicksort(int x[10],int first,int last)
{
1       int temp,pivot,i,j;
2       if(first<last)
        {
3               pivot=first;
4               i=first;
5               j=last;
6               while(i<j)
                {
7                       while(x[i]<=x[pivot] && i<last)
8                       i++;
9                       while(x[j]>x[pivot])
10                      j--;
11                      if(i<j)
                        {
12                              temp=x[i];
13                              x[i]=x[j];
14                              x[j]=temp;
                        }
                }
15              temp=x[pivot];
16              x[pivot]=x[j];
17              x[j]=temp;
18              quicksort(x,first,j-1);
19              quicksort(x,j+1,last);
        }
20}
```

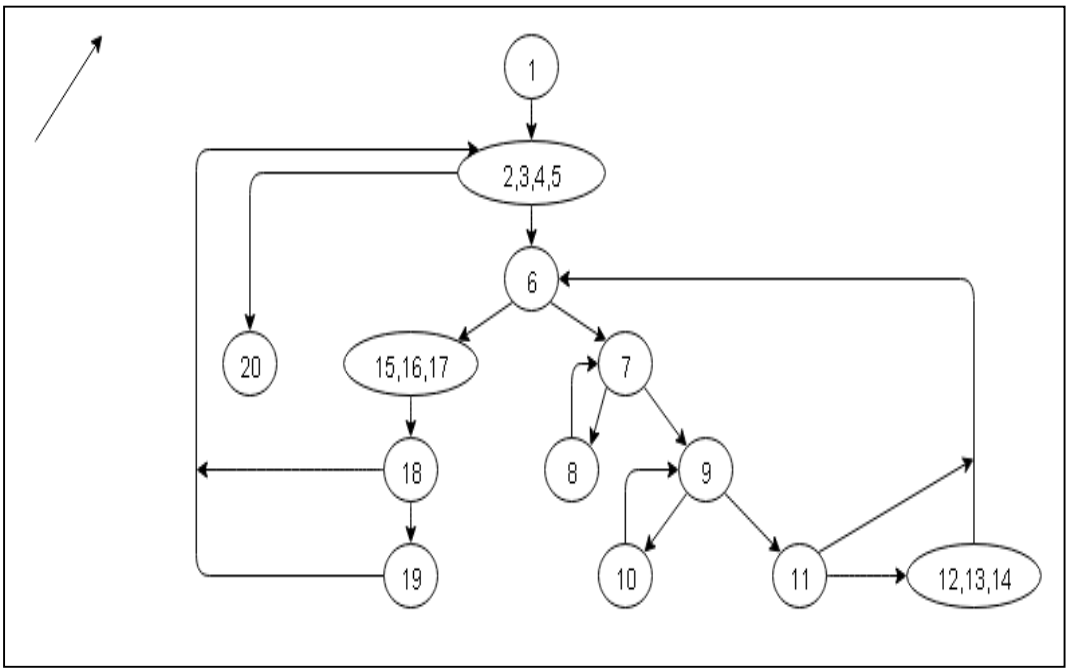
Testing

Technique used: Basis Path Testing

Basis path testing is a form of structural testing (white box testing). The method devised by McCabe to carry out basis path testing has four steps: these are

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

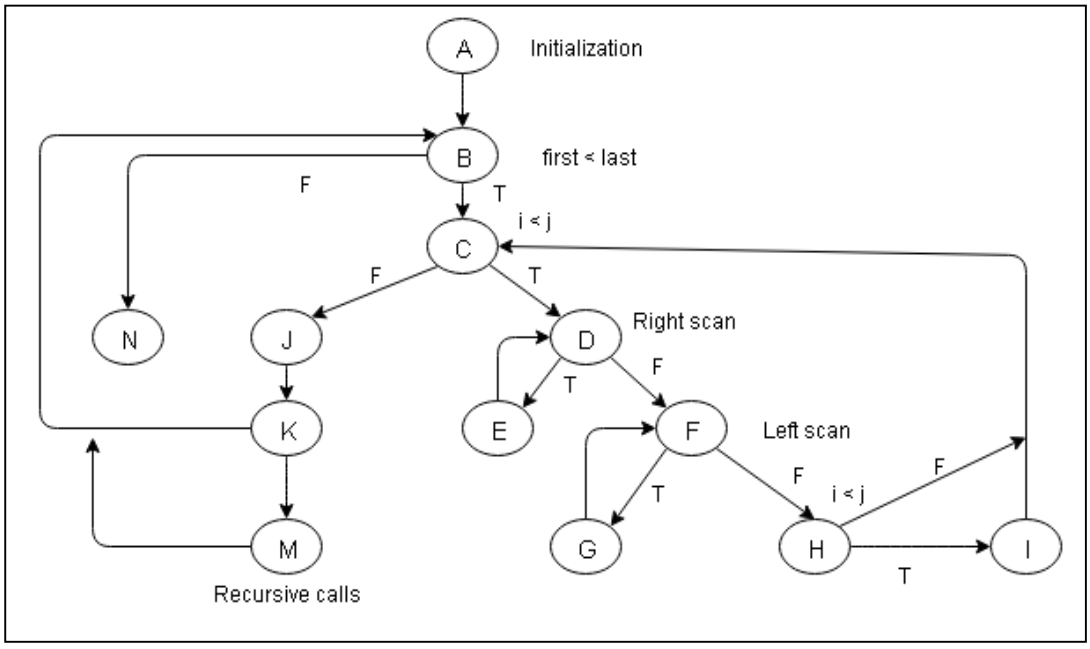
Step 1: Program graph



Using the program graph we derive (Decision to Decision) DD path graph for binary search program.

DD path nodes	Program graph nodes
A	1
B	2,3,4,5
C	6
D	7
E	8
F	9
G	10
H	11
I	12,13,14

J	15,16,17
K	18
M	19
N	20



DD - Path graph for Quick sort function

Cyclomatic complexity V(G) = e-n+2p

Edges = 18, Nodes = 13, Regions = 1

V(G) = 18-13+2 = 7

According to cyclomatic complexity 7 feasible basis path exists :

P1 : A,B,N

P2 : A,B,C,J,K,B

P3 : A,B,C,J,K,M,B

P4 : A,B,C,D,F,H,C

P5 : A,B,C,D,F,H,I,C

P6 : A,B,C,D,E,D,F,H

P7 : A,B,C,D,F,G,F,H

Test cases

TC ID	Description	Inputs		Expected output	Remarks	Status
		X[]	First, Last			
1	Test for path P1	5	1,1	Sorted	Only one element	
2	Test for path P2	5,4	1,2	Repeated &sorted	Two elements	
3	Test for path P3	1,2,3 or 3,1,2	1,3	Repeated &sorted	Three elements	
4	Test for path P4	1,2,3,4,5	1,5	Repeated &sorted	ASC sequence	
5	Test for path P5	5,4,3,2,1	1,5	Repeated &sorted	DSC sequence	
6	Test for path P6	1,4,3,2,5	1,5	Repeated &sorted	Pivot is MIN	
7	Test for path P7	5,2,3,1,4	1,5	Repeated &sorted	Pivot is MAX	

Test Report:

1. Number of TC Executed :
2. Number of Defects raised :
3. Number of TC's passed :
4. Number of TC's failed :