Experiment No: 8 Date

# Implementation of Quick Sort Algorithm and Path Testing

## problem Statement

Design, develop and execute a program to implement the quick sort algorithm. Using the program's control flow, identify basis path and derive independent test case using path testing.

Theory:

Path Testing

Path Testing is a white-box-testing technique used to ensure that every independent path in the program's control flow has been tested.

i. Draw the control Flow Graph (CFG)

Nodes = Statements

Edges = flow of control

ii. Calculate Cyclomatic Complexity (V(G))

Using formula:

$V(G) = E - N + 2$

where

E : number of edges

N : number of nodes

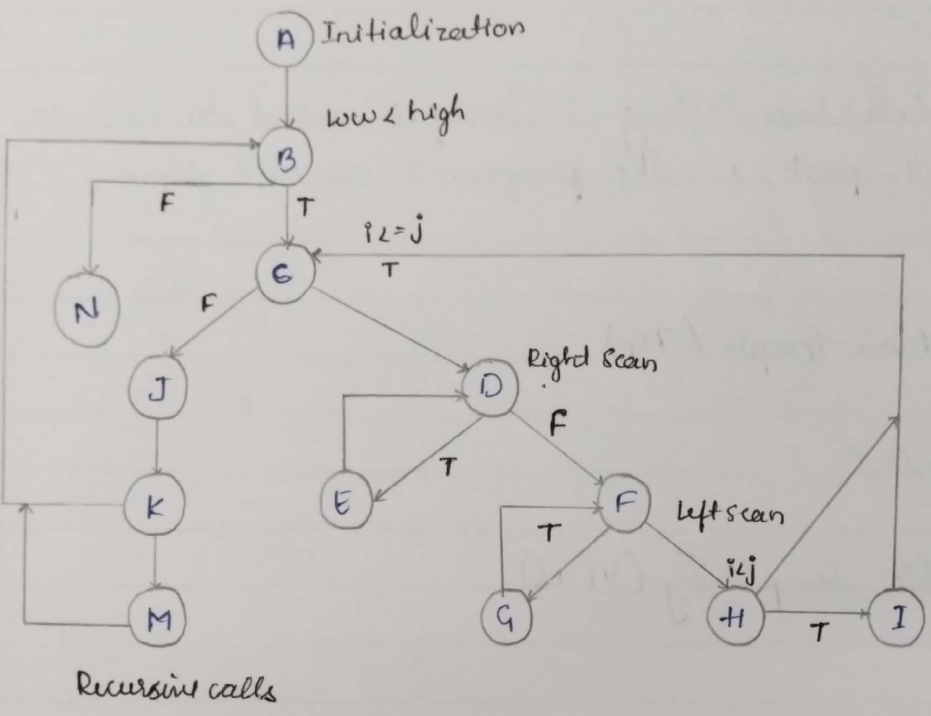
iii. Identify Bases Paths

Each independent path = one unique test scenario

iv. Design Test cases to cover each path.

- All decision are tested.
- Logic errors are minimized.
- Program reliability increases.

## Program Graph




## Independent paths

Edges = 18 Nodes = 13 P = 1

$$V(G) = E - N + 2P$$
$$= 18 - 13 + 2$$
$$= 7$$

## Independent Path

$P_1$: A-B-N

$P_2$: A-B-C-J-K-B

$P_3$: A-B-C-J-K-M-B

$P_4$: A-B-C-D-F-H-C

$P_5$: A-B-C-D-F-H-I-C

$P_6$: A-B-C-D-E-D-F-H

$P_7$: A-B-C-D-F-G-F-H

| Paths | Inputs | | Expected Output | Remarks |
|---|---|---|---|---|
| | X[] | First, last | | |
| P1: A-B-N | 5 | 1,1 | Sorted | only one Element |
| P2: A-B-C-J-K-B | 5,4 | 1,2 | Repeated & Sorted | Two elements |
| P3: A-B-C-J-K-M-B | 1,2,3 OR 3,2,1 | 1,3 | Repeated & Sorted | Three elements |
| P4: A-B-C-D-F-H-C | 1,2,3,4,5 | 1,5 | Repeated & Sorted | ASC Sequence |
| P5: A-B-C-D-F-H-I-C | 5,4,3,2,1 | 1,5 | Repeated & Sorted | DSC Sequence |
| P6: A-B-C-D-E-D-F-H | 1,4,3,2,5 OR 2,2,2,2,2 | 1,5 | Repeated & Sorted | Pivot in MID |
| P7: A-B-C-D-F-G-F-H | 5,2,3,1,4 | 1,5 | Repeated & Sorted | Pivot is MAX |

Source Code:

```
#include <stdio.h>
void swap (int *a, int *b){
    int t = *a
        *a = *b
        *b = t;
}
int partition (int arr[], int low, int high){
    int pivot = arr[high], i = low-1;
    for (int j= low ; j< high ; j++){
        if (arr[j] <= pivot ){
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i+1], &arr[high]);
    return i+1;
}
void quickSort (int arr[], int low, int high){
    if (low < high){
        int pi = partition (arr, low, high);
        quickSort (arr, low, pi-1);
        quickSort (arr, pi+1, high );
    }
}
int main (){
 int arr[100], n;
 printf(" Enter n :"); scanf("%d",&n);
 for (int i=0; i<n; i++){ scanf(" %d", &arr[i]);}
 quickSort (arr, 0, n-1);
 for (int i=0; i<n ; i++){ printf(" %d", arr[i]);}
 return 0;
}
```

Conclusion:

The Quick Sort algorithm was successfully implemented and tested using basis path testing. By analyzing the control flow and desiring cyclomatic complexity, four independent paths were identified.

References:


* Roger. S Pressman, Software Engineering : A Practitioner's Approach
* Ian Sommerville, SOFTWARE ENGINEERING, Addison - Wesley 9th Edition

Experiment No: 9 Date:

Next Date Function - Boundary Value Analysis Testing

Problem Statement

Design, develop and execute program to compute the next calendar date for a given valid input date. Analyze the program using Boundary Value Analysis (BVA) to derive test cases, execute them.

# Theory

## Boundary Value Analysis (BVA)

Boundary Value Analysis is a black-box testing technique based on the observation that most bugs occur at the edges of input ranges, not in the middle.

## Boundaries for the Next Date Program

i) Day Boundaries
- Lower : 1, 2
- Upper : 30, 31
- February Boundaries : 28, 29

ii) Month Boundaries
- Lower : 1, 2
- Upper : 11, 12

iii) Year Boundaries
- Lower : 1900 , 1901
- Upper : 2099 , 2100

## Next date Output Boundary Value Analysis Cases

| Case ID | Description | Input Data | | | Expected Output | | | Actual Output | | | Status | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | month | day | year | month | day | year | month | day | year | | |
| 1 | Enter the min value month, day and year | 1 | 1 | 1812 | 1 | 2 | 1812 | 1 | 2 | 1812 | pass | |
| 2 | Enter the min+1 value of for year and min for month and day | 1 | 1 | 1813 | 1 | 2 | 1813 | 1 | 2 | 1813 | pass | |
| 3 | Enter the normal value for year and min for month and day | 1 | 1 | 1912 | 1 | 2 | 1912 | 1 | 2 | 1912 | pass | |
| 4 | Enter the max-1 value for year and min for month and day | 1 | 1 | 2012 | 1 | 2 | 2012 | 1 | 2 | 2012 | pass | |
| 5 | Enter the max value of for year and min for month and day | 1 | 1 | 2013 | 1 | 2 | 2013 | 1 | 2 | 2013 | pass | |
| 6 | Enter the min+1 value of day and min for month and year | 1 | 2 | 1812 | 1 | 3 | 1812 | 1 | 3 | 1812 | pass | |
| 7 | Enter the min+1 value for day and year and min for month | 1 | 2 | 1813 | 1 | 3 | 1813 | 1 | 3 | 1813 | pass | |
| 8 | Enter the min+1 value for day, normal value for year and min value for month | 1 | 2 | 1912 | 1 | 3 | 1912 | 1 | 3 | 1912 | pass | |
| 9 | Enter the min+1 value for day, max-1 value for year and min value for month | 1 | 2 | 2012 | 1 | 3 | 2012 | 1 | 3 | 2012 | pass | |
| 10 | Enter the min+1 value for day, max value for month | 1 | 2 | 2013 | 1 | 3 | 2013 | 1 | 3 | 2013 | pass | |
| 11 | Enter the normal value of day and min for year and month | 1 | 15 | 1812 | 1 | 16 | 1812 | 1 | 16 | 1812 | pass | |
| 12 | Enter the normal value for day and min+1 for year and min for month | 1 | 15 | 1813 | 1 | 16 | 1813 | 1 | 16 | 1813 | pass | |
| 13 | Enter the normal value for day normal value for year and min value for month | 1 | 15 | 1912 | 1 | 16 | 1912 | 1 | 16 | 1912 | pass | |
| 14 | Enter the normal value for day, max-1 value for year and min value for month | 1 | 15 | 2012 | 1 | 16 | 2012 | 1 | 16 | 2012 | pass | |
| 15 | Enter the normal value for day, max value for year and min value for month | 1 | 15 | 2013 | 1 | 16 | 2013 | 1 | 16 | 2013 | pass | |
| 16 | Enter the max-1 value of day and min for day and year. | 1 | 30 | 1812 | 1 | 31 | 1812 | 1 | 31 | 1812 | pass | |
| 17 | Enter the max-1 value for day and min for month and min+1 for year | 1 | 30 | 1813 | 1 | 31 | 1813 | 1 | 31 | 1813 | pass | |
| 18 | Enter the max-1 value for day, normal value for year and min value for month. | 1 | 30 | 1912 | 1 | 31 | 1912 | 1 | 31 | 1912 | pass | |
| 19 | Enter the max-1 value for day, max-1 value for year and min value for month | 1 | 30 | 2012 | 1 | 31 | 2012 | 1 | 31 | 2012 | pass | |
| 20 | Enter the max-1 value for day, max value for year and min value for month | 1 | 30 | 2013 | 1 | 31 | 2013 | 1 | 31 | 2013 | pass | |
| 21 | Enter the max value of day and min for year and month | 1 | 31 | 1812 | 2 | 1 | 1812 | 2 | 1 | 1812 | pass | |
| 22 | Enter the max value for day and min for month and min+1 for year | 1 | 31 | 1813 | 2 | 1 | 1813 | 2 | 1 | 1813 | pass | |
| 23 | Enter the max value for day, normal value for year and min value for month | 1 | 31 | 1912 | 2 | 1 | 1912 | 2 | 1 | 1912 | pass | |
| 24 | Enter the max value for day, max-1, value for year and min value for month | 1 | 31 | 2012 | 2 | 1 | 2012 | 2 | 1 | 2012 | pass | |
| 25 | Enter the max value for day, max value for year and min value for month | 1 | 31 | 2013 | 2 | 1 | 2013 | 2 | 1 | 2013 | pass | |

Source code

```
#include <stdio.h>
int isLeap(int y){return (y%400==0)||(y%4==0&&y%100!=0);}
int daysInMonth(int m, int y){
    if(m==2) return isLeap(y)?29:28;
    if(m==4||m==6||m==9||m==11) return 30;
    return 31;
}
int main(){
    int d,m,y;
    printf("enter day month year"); scanf("%d %d %d",&d,&m,&y);
    if(m<1||m>12||y<1900||y>2100){
        printf("Invalid date"); return 0;
    }
    int dim = daysInMonth(m,y);
    if(d<1||d>dim){
        printf("Invalid date"); return 0;
    }
    if(d<dim) d++;
    else{
        d=1;
        if(m==12){m=1;y++;}
        else m++;
    }
    printf("%02d-%02d-%04d",d,m,y);
    return 0;
}
```

## Conclusion

Boundary Value Analysis helped identify important edge conditions such as month transitions, year transitions and leap year scenarios. Testing these boundary values ensured that the Next Date program correctly handles all critical cases and is robust near extreme input conditions.

## References:


* Roger S. Pressman, Software Engineering : A Practitioner's Approach.
* Ian Sommerville, SOFTWARE ENGINEERING, Addison-Wesley, 9th Edition.

Experiment No : 10 Date :

# Next Date Program Using Equivalence Class Analysis

## Problem Statement

Implement a program that calculates the next date for a given day, month and year. Using Equivalence Class Analysis (ECA), divide the input domain into valid and invalid classes, derive test cases, execute them and discuss the results.

Theory

Equivalence Class Analysis (ECA)

Equivalence class Analysis is a black-box testing technique where the input domain is divided into classes of inputs that behave similarly.

From each class, only one representative value needs to be tested.

purpose:

- Reduce total number of test cases
- Maintain complete coverage
- Identify both valid and invalid inputs

## Weak and strong Normal Equivalence Class

| Case ID | Description | Input Data | | | Expected Output | | | Actual Output | | | Status | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | month | day | year | month | day | year | month | day | year | | |
| WN1, SN1 | Enter M1, D1 and Y1 valid cases | 6 | 15 | 1912 | 6 | 16 | 1912 | 6 | 16 | 1912 | pass | |

## Weak Robustness Equivalence Class

| Case Id | Description | Input Data | | | Expected Output | | | Actual Output | | | Status | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | month | day | year | month | day | year | month | day | year | | |
| WR1 | Enter M1, D1 & Y1 | 6 | 15 | 1912 | 6 | 16 | 1912 | 6 | 16 | 192 | pass | |
| WR2 | Enter M2, D1 & Y1 | -1 | 15 | 1912 | should display message value of month not in range | | | month not in range | | | pass | |
| WR3 | Enter M3, D1 and Y1 | 13 | 15 | 1912 | should display message vale of month not in range | | | month not in range | | | pass | |
| WR4 | Enter M1, D2 and Y1 | 6 | -1 | 1912 | should display message value of day not in range | | | day not in range | | | pass | |
| WR5 | Enter M1, D3 and Y1 | 6 | 32 | 1912 | should display message value of day not in range | | | day not in range | | | pass | |
| WR6 | Enter M1, D1 and Y2 | 6 | 15 | 1811 | should display message value of year not in range | | | year not in range | | | pass | |
| WR7 | Enter M1, D1 and Y3 | 6 | 15 | 2014 | should display message value of year not in range | | | year not in range | | | pass | |

## Strong Robustness Equivalence Class

| Case Id | Description | Input Data | | | Expected Output | Actual Output | Status | Comment |
|---|---|---|---|---|---|---|---|---|
| | | month | day | year | | | | |
| SR1 | Enter M1, D1 and Y1 | -1 | 15 | 1912 | display message value of month not in range | month not in range | pass | |
| SR2 | Enter M1, D2 and Y1 | 6 | -1 | 1912 | display message value of day not in range | day not in range | pass | |
| SR3 | Enter M1, D1 and Y2 | -6 | 15 | 1817 | display message value of year not in range | year not in range | pass | |
| SR4 | Enter M2, D2 and Y1 | -9 | -1 | 1912 | display message value of month not in range | month not in range | pass | |
| | | | | | display message value of day not in range | day not in range | pass | |
| SR5 | Enter M1, D2 and Y2 | -6 | -1 | 1811 | display message value of day not in range | day not in range | Pass | |
| | | | | | display message value of year not in range | year not in range | Pass | |
| SR6 | Enter M2, D1 and Y2 | -1 | 15 | 1811 | display message value of month not in range | month not in range | pass | |
| | | | | | display message value of year not in range | year not in range | pass | |
| SR7 | Enter M2, D2 and Y2 | -1 | -1 | 1811 | display message value of month not in range | month not in range | pass | |
| | | | | | display message value of day not in range | day not in range | pass | |
| | | | | | display message value of year not in range | year not in range | pass | |

Source Code

```
#include <stdio.h>
int isLeap(int y){return (y%400==0) || (y%4==0 && y%100!=0);}
int daysInMonth(int m, int y){
    if(m==2) return isLeap(y)?29:28;
    if(m==4 || m==6 || m==9 || m==11) return 30;
    return 31;
}

int main(){
  int d, m, y;
  printf("Enter date"); scanf("%d %d %d", &d, &m, &y);
  if(m<1 || m>12 || y<1811 || y>2012){
    printf("Invalid date"); return 0;
  }

  int dim = daysInMonth(m,y);
  if(d<1 || d>dim){
    printf("Invalid date"); return 0;
  }

  if(d<dim) d++;
  else{
    d=1;
    if(m==12){m=1; y++;}
    else m++;
  }


  printf("%02d-%02d-%04d", d, m, y);
  return 0;

}
```

## Conclusion

Equivalence Class Analysis allowed us to divide the input space of the Next Date function into well-defined valid and invalid classes. By selecting one representative value from each class, we significantly reduced the number of test ~~cases~~ cases while still maintaining complete functional Coverage.

## References


* Roger S. Pressman, Software Engineering : A Practitioner's Approach.
* Ian Sommerville , SOFTWARE ENGINEERING, Addison-Wesly, 9th Edition