VEMU INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

15A05509- Object Oriented Analysis and Design & Software Testing Laboratory

Regulation – R15

Year / Semester: III / I

EXPERIMENT 1

1. Write a c program to demonstrate the working of the fallowing constructs: i) do…while

ii) while…do

iii) if …else

iv)switch

v) for Loops in C language

i) AIM: To demonstrate the working of do...while construct

Objective

To understand the working of do while with different range of values and test cases

```
#include <stdio.h>
void main ()
{
        int i, n=5,j=0;
        clrscr();
        printf(“enter a no”);
        scanf(“%d”,&i);
        do
        {
                if(i%2==0)
                {
                        printf("%d", i);
                        printf("is a even no.");
                        i++;
                        j++;
                }
                else
                {
                        printf("%d", i);
                        printf("is a odd no.\n");

                        i++;
                        j++;
                }
        }
        while(i>0&&j<n);
        getch();
}
```

Input	Actual output
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	Expected output	Actual output	Remarks
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no: 2

Test case name: Negative values within a range

Input = -2	Expected output	Actual output	Remarks
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
123456789122222222	123456789122222	13 234567891222222215	fail

ii) AIM: To demonstrate the working of while construct Objective

To understand the working of while with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>
void main ()
{
        int i, n=5,j=1;
        clrscr();
        printf(“enter a no”);
        scanf(“%d”,&i);
        while (i>0 && j<n)
        {
                if(i%2==0)
                {
                        printf(“%d”,i);
                        printf(“is a even number”);
                        i++;
                        j++;
                }
                else
                {
                        printf(“%d”,i);
                        printf(“is a odd number”);
                         i++;
                         j++;
                 }
        }
        getch();
}
```

Input	**Actual output**
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:

Test case no:1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no:2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
123456789122222222	1234567891222222222	13 2345678912222222215	fail

iii) AIM: To demonstrate the working of if else construct

Objective To understand the working of if else with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>
void main ()
{
        int i;
        clrscr();
        printf(“enter a number”);
        scanf(“%d”,&i);
        if(i%2==0)
        {
                printf(“%d”,i);
                printf(“is a even number”);
        }
        else
        {
                printf(“%d”,i);
                printf(“is a odd number”);
        }
        getch();
}
```

Input	**Actual output**
2	2 is even number
	3 is odd number
	4 is even number

```
5 is odd number
6 is even number
```

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no:2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
1234567891222222222	1234567891222222222	13234567891222222215	fail

iv) AIM: To demonstrate the working of switch construct

Objective To understand the working of switch with different range of values and test cases

```
#include<stdio.h>
void main()
{
        int a,b,c;
        clrscr();
        printf("1.Add/n 2.Sub /n 3.Mul /n 4.Div /n Enter Your choice");
        scanf("%d", &i);
        printf("Enter a,b values");
        scanf("%d%d",&a,&b);
        switch(i)
        {
                case 1: c=a+b;
                printf(“The sum of a & b is: %d” ,c);
                break;
                case 2: c=a-b;
                printf(“The Diff of a & b is: %d” ,c);
                break;
                case 3: c=a*b;
                printf(“The Mul of a & b is: %d” ,c);
                break;
                case 4: c=a/b;
                printf(“The Div of a & b is: %d” ,c);
                break;
                default: printf(“Enter your choice”);
                break;
        }
        getch();
}
```

Output:

Input	**Output**
Enter Ur choice: 1	
Enter a, b Values: 3, 2	The sum of a & b is:5
Enter Ur choice: 2	
Enter a, b Values: 3, 2	The diff of a & b is: 1
Enter Ur choice: 3	
Enter a, b Values: 3, 2	The Mul of a & b is: 6
Enter Ur choice: 4	
Enter a, b Values: 3, 2	The Div of a & b is: 1

Test cases:

Test case no: 1

Test case name: Positive values within range

Input	**Expected output**	**Actual output**	**Remarks**
Enter Ur choice: 1			
Enter a, b Values: 3, 2	The sum of a & b is:5	5	
Enter Ur choice: 2			
Enter a, b Values: 3, 2	The diff of a & b is: 1	1	Success
Enter Ur choice: 3			
Enter a, b Values: 3, 2	The Mul of a & b is: 6	6	
Enter Ur choice: 4			

Enter a, b Values: 3, 2	The Div of a & b is: 1	1	

Test case no:2

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
Option: 1			
a= 22222222222222			
b=22222222222222	44444444444444	-2	fail

Test case no: 3

Test case name: Divide by zero

Input	Expected output	Actual output	Remarks
Option: 4			
a= 10 & b=0	error		fail

v)AIM: To demonstrate working of for construct

Objective To understand the working of for with different range of values and test cases

```
#include <stdio.h>
#include <conio.h>
void main ()
{
        int i;
        clrscr();
        printf(“enter a no”);
        scanf(“%d”,&i);
        for(i=1;i<=5;i++)
        {
                if(i%2==0)
                {
                        printf(“%d”, i);
                        printf(”is a even no”);
                        i++;
                }
                else
                {



                        printf(“%d”, i);
```

```
            printf(“is a odd no”);
            i++;
        }
    }
    getch();
}
```

Output:

```
Enter a no: 5
0 is a even no
1 is a odd no
2 is a even no
3 is a odd no
4 is a even no
5 is a odd no
```

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	0 is even number	0 is even number	
	1 is odd number	1 is odd number	success
	2 is even number	2 is even number	

Test case no: 2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	0 is even number	0 is an even number	
	-1 is odd number	-1 is even no	fail
	-2 is even number	-2 is odd no	

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
1234567891222222222	1234567891222222222	13 234567891222222215	fail

EXPERIMENT 2

AIM: A program written in c language for matrix multiplication fails —Introspect the causes for its failure and write down the possible reasons for its failure.

Objective: Understand the failures of matrix multiplication

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[3][3],b[3][3],c[3][3],i,j,k,m,n,p,q;
        clrscr();
        printf(“Enter 1st matrix no.of rows & cols”);
        scanf(“%d%d”,&m,&n);
        printf(“Enter 2nd matrix no.of rows & cols”);
        scanf(“%d%d”,&p,&q);
        printf("\n enter the matrix elements");
        for(i=0;i<m;i++);
        {
                for(j=0;j<n;j++);
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("\n a matrix is\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
```

```
            {
                scanf("%d\t",&b[i][j]);
            }
        }
        printf("\n b matrix is\n");
        for(i=0;i<p;i++)
        {
            for(j=0;j<q;j++)
            {
                printf("%d\t",b[i][j]);
            }
            printf("\n");
        }
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                c[i][j]=0;
                for(k=0;k<n;k++)
                {
                    c[i][j]=c[i][j]+a[i][k]*b[k][j];
                }
            }
        }
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                printf("%d\t",c[i][j]);
            }
            printf("\n");
        }
        getch();
    }
```

Output:

```
Enter Matrix1:      1 1 1
                    1 1 1
                    1 1 1

Enter Matrix2:      1 1 1
                    1 1 1
                    1 1 1

Actual Output:      3 3 3
                    3 3 3
                    3 3 3
```

Test cases:

Test case no: 1

Test case name: Equal no.of rows & cols

Input	Expected output	Actual output	Remarks
Matrix1 rows & cols= 3 3			
Matrix2 rows & cols= 3 3			
Matrix1: 1 1 1			
1 1 1	3 3 3	3 3 3	
1 1 1	3 3 3	3 3 3	Success
	3 3 3	3 3 3	
Matrix2: 1 1 1			
1 1 1			
1 1 1			

Test case no:2

Test case name: Cols of 1st matrix not equal to rows of 2nd matrix

Input	Expected output	Actual output	Remarks
Matrix1 rows & cols= 2 2	Operation Can_t be Performed		fail
Matrix2 rows & cols= 3 2			

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks

Matrix1 rows & cols= 2 2

Matrix2 rows & cols= 2 2

1234567891	2222222222	fail
2234567891	2222222221	
234567891	22222221533	
213242424	56456475457	

EXPERIMENT 3

AIM: Take any system (e.g. ATM system) and study its system specifications and report the various bugs.

1. Machine is accepting ATM card.
2. Machine is rejecting expired card.
3. Successful entry of PIN number.
4. Unsuccessful operation due to enter wrong PIN number 3 times.
5. Successful selection of language.
6. Successful selection of account type.
7. Unsuccessful operation due to invalid account type.
8. Successful selection of amount to be withdrawn.
9. Successful withdrawal.
10. Expected message due to amount is greater than day limit.
11. Unsuccessful withdraw operation due to lack of money in ATM.
12. Expected message due to amount to withdraw is greater than possible balance.
13. Unsuccessful withdraw operation due to click cancel after insert card.

EXPERIMENT 4

AIM:Write the test cases for any known application (e.g. Banking application)

1. Checking mandatory input parameters
2. Checking optional input parameters
3. Check whether able to create account entity.
4. Check whether you are able to deposit an amount in the newly created account (and thus updating the balance)
5. Check whether you are able to withdraw an amount in the newly created account (after deposit) (and thus updating the balance)
6. Check whether company name and its pan number and other details are provided in case of salary account
7. Check whether primary account number is provided in case of secondary account
8. Check whether company details are provided in cases of company's current account
9. Check whether proofs for joint account is provided in case of joint account
10. Check whether you are able deposit an account in the name of either of the person in an joint account.
11. Check whether you are able withdraw an account in the name of either of the person in an joint account.
12. Check whether you are able to maintain zero balance in salary account
13. Check whether you are not able to maintain zero balance (or mini balance) in non-salary account.

EXPERIMENT 5

AIM: Create a test plan document for any application (e.g. Library Management System)

The Library Management System is an online application for assisting a librarian managing book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification**.** This test report is the result for testing in the LMS. It mainly focuses on two problems

1. What we will test

2. How we will test.

3. GUI test

Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties

4. Database test

Pass criteria: Results of all basic and advanced operations are normal (refer to section 4)

5. Basic function test Add a student

Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and Phone number.

The retrieved customer information by viewing customer detail should contain the four attributes.

6. Update/delete student

1. The record would be selected using the student ID.
2. Updates can be made on full. Items only: Name, Address, Phone number The record can be deleted if there are no books issued by user. The updated values would be reflected if the same customer's ID/SSN is called for.

7. Check-in book

Librarians can check in a book using its call number

1. The check-in can be initiated from a previous search operation where user has selected a set

   of books.
2. The return date would automatically reflect the current system date.
3. Any late fees would be computed as difference between due date and return date at rate of 10 cents a day.

EXPERIMENT 6

AIM: Study of Any Testing Tool (Win Runner)

Win Runner is a program that is responsible for the automated testing of software. Win Runner is a Mercury Interactive enterprise functional testing tool for Microsoft windows applications.

Importance of Automated Testing:

1. Reduced testing time
2. Consistent test procedures – ensure process repeatability and resource independence. Eliminates errors of manual testing
3. Reduces QA cost – Upfront cost of automated testing is easily recovered over the life-time of the product
4. Improved testing productivity – test suites can be run earlier and more often
5. Proof of adequate testing
6. For doing Tedious work – test team members can focus on quality areas.

Win Runner Uses:

1. With Win Runner sophisticated automated tests can be created and run on an application.
2. A series of wizards will be provided to the user, and these wizards can create tests in an automated manner.
3. Another impressive aspect of Win Runner is the ability to record various interactions, and transform them into scripts. Win Runner is designed for testing graphical user interfaces.
4. When the user makes an interaction with the GUI, this interaction can be recorded. Recording the interactions allows to determine various bugs that need to be fixed.
5. When the test is completed, Win Runner will provide with detailed information regarding the results. It will show the errors that were found, and it will also give important information about them. The good news about these tests is that they can be reused many times.
6. Win Runner will test the computer program in a way that is very similar to normal user interactions. This is important, because it ensures a high level of accuracy and realism. Even if an engineer is not physically present, the Recover manager will troubleshoot any problems that may occur, and this will allow the tests to be completed without errors.
7. The Recover Manager is a powerful tool that can assist users with various scenarios. This is important, especially when important data needs to be recovered.

The goal of Win Runner is to make sure business processes are properly carried out. Win Runner uses TSL, or Test Script Language.

Win Runner Testing Modes

Context Sensitive

Context Sensitive mode records your actions on the application being tested in terms of the GUI objects you select (such as windows, lists, and buttons), while ignoring the physical lo-cation of the object on the screen. Every time you perform an operation on the application being tested, a TSL statement describing the object selected and the action performed is generated in the test script. As you record, Win Runner writes a unique description of each selected object to a GUI map.

The GUI map consists of files maintained separately from your test scripts. If the user interface of your application changes, you have to update only the GUI map, instead of hundreds of tests. This allows you to easily reuse your Context Sensitive test scripts on future versions of your application.

To run a test, you simply play back the test script. Win Runner emulates a user by mov-ing the mouse pointer over your application, selecting objects, and entering keyboard input. Win Runner reads the object descriptions in the GUI map and then searches in the application being tested for objects matching these descriptions. It can locate objects in a window even if their placement has changed.

Analog

Analog mode records mouse clicks, keyboard input, and the exact x- and y-coordinates traveled by the mouse. When the test is run, Win Runner retraces the mouse tracks. Use Analog mode when exact mouse coordinates are important to your test, such as when testing a drawing application.

The Win Runner Testing Process

Testing with *Win Runner* involves six main stages:

1. Create the GUI Map

The first stage is to create the GUI map so Win Runner can recognize the GUI objects in the application being tested. Use the Rapidest Script wizard to review the user interface of your application and systematically add descriptions of every GUI object to the GUI map. Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.

2. Create Tests

Next is creation of test scripts by recording, programming, or a combination of both. While recording tests, insert checkpoints where we want to check the response of the application being tested. We can insert checkpoints that check GUI objects, bitmaps, and databases. During this process, Win Runner captures data and saves it as *expected results*—the expected response of the application being tested.

3. Debug Tests

Run tests in Debug mode to make sure they run smoothly. One can set breakpoints, mon-itor variables, and control how tests are run to identify and isolate defects. Test results are saved in the debug folder, which can be discarded once debugging is finished. When Win Runner runs a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in **If**, **While**, **Switch**, and **For** statements. We can use the **Syntax Check** options (**Tools >Syn-tax Check**) to check for these types of syntax errors before running your test.

4. Run Tests

Tests can be run in Verify mode to test the application. Each time Win Runner encounters a checkpoint in the test script, it compares the current data of the application being tested to the expected data captured earlier. If any mismatches are found, Win Runner captures them as *actual results*.

5. View Results

Following each test run, Win Runner displays the results in a report. The report details all the major events that occurred during the run, such as checkpoints, error messages, system