IV Year – I SEMESTER

T	P	C
0	3	2

Software Testing Lab

Lab Assignments

Problem Statement 01

Consider an automated banking application. The user can dial the bank from a personal computer, provide a six-digit password, and follow with a series of keyword commands that activate the banking function. The software for the application accepts data in the following form:

Area Code	Blank or three-digit number
Prefix	Three-digit number, not beginning with 0 or 1
Suffix	Four-digit number
Password	Six-character alphanumeric
Commands	"Check status", "Deposit", "Withdrawal"

Design adhoc test cases to test the system

Problem Statement 02

Consider an automated banking application. The user can dial the bank from a personal computer, provide a six-digit password, and follow with a series of keyword commands that activate the banking function. The software for the application accepts data in the following form:

Area Code	Blank or three-digit number
Prefix	Three-digit number, not beginning with 0 or 1
Suffix	Four-digit number
Password	Six-character alphanumeric
Commands	"Check status", "Deposit", "Withdrawal"

Design the test cases to test the system using following Black Box testing technique:

BVA, Worst BVA, Robust BVA, Robust Worst BVA

Equivalence class testing (Input/Output domain)

Problem Statement 03

Consider an application that is required to validate a number according to the following simple rules:

A number can start with an optional sign.

The optional sign can be followed by any number of digits.

The digits can be optionally followed by a decimal point, represented by a period.

If there is a decimal point, then there should be two digits after the decimal.

Any number-whether or not it has a decimal point, should be terminated a blank.

A number can start with an optional sign.

The optional sign can be followed by any number of digits.

The digits can be optionally followed by a decimal point, represented by a period.

If there is a decimal point, then there should be two digits after the decimal.

Any number-whether or not it has a decimal point, should be terminated a blank. Generate test cases to test valid and invalid numbers.

(HINT) Use Decision table and cause-effect graph to generate test cases.

Problem Statement 04

Generate test cases using Black box testing technique to Calculate Standard Deduction on Taxable Income. The standard deduction is higher for tax payers who are 65 or older or blind. Use the method given below to calculate tax.

The first factor that determines the standard deduction is the filing status. The basic standard deduction for the various filing status are:

Single	$4,750
Married, filing a joint return	$9,500
Married, filing a separate return	$7,000

If a married couple is filing separate returns and one spouse is not taking standard Deduction, the other spouse also is not eligible for standard deduction.

An additional $1,000 is allowed as standard deduction, if either the filer is 65 yrs or the spouse is 65 yrs or older (the latter case applicable when the filing status is "Married" and filing "joint").

An additional $1,000 is allowed as standard deduction, if either the filer is blind or the spouse is blind (the latter case applicable when the filing status is "married" and filing "joint").

(HINT):

From the above description, it is clear that the calculation of standard deduction depends on the following 3 factors:

Status of filing of the filer
Age of the filer
Whether the filer is blind or not

In addition, in certain cases, the following additional factors also come into play in calculating the standard deduction.

Whether spouse has claimed standard deduction
Whether spouse is blind
Whether the spouse is more than 65 years old

Problem Statement 05

Consider the following program segment:

```
int max (int i, int j, int k)
{
int max;
if (i>j) then
if (i>k) then max=i;
else max=k;
else if (j > k) max=j
else max=k
return (max);
 }
```

Draw the control flow graph for this program segment
Determine the cyclomatic complexity for this program
Determine the independent paths

Problem Statement 06

Source code of simple insertion sort implementation using array in ascending order in c programming language

```
#include<stdio.h>
int main(){
int i,j,s,temp,a[20];
```

```
Printf ("Enter total elements: "); Scanf ("%d",&s);
printf("Enter %d elements: ",s); for(i=0;i<s;i++) scanf("%d",&a[i]);
for(i=1;i<s;i++){ temp=a[i]; j=i-1; while((temp<a[j])&&(j>=0)){ a[j+1]=a[j]; j=j-1;

}
a[j+1]=temp;
}
printf("After sorting: ");
for(i=0;i<s;i++)
printf(" %d",a[i]);
return 0;
}
```

HINT: for loop is represented as while loop

Draw the program graph for given program segment b) Determine the DD path graph
Determine the independent paths
Generate the test cases for each independent path

Problem Statement 07

Consider a system having an FSM for a stack having the following states and transitions:

States
Initial: Before creation
Empty: Number of elements = 0
Holding: Number of elements > 0, but less than the maximum capacity
Full: Number elements = maximum
Final: After destruction
Initial to Empty: Create
Empty to Holding, Empty to Full, Holding to Holding, Holding to Full: Add
Empty to Final, Full to Final, Holding to Final: Destroy
Holding to Empty, Full to Holding, Full to Empty: Delete

Design test cases for this FSM using state table-based testing.

Problem Statement 08

Given the following fragment of code, how many tests are required for 100% decision coverage? Give the test cases.

```
if width > length
then biggest_dimension = width if height > width
then biggest dimension = height end_if
else if biggest dimension = length then if height > length
then biggest_dimension = height end_if
end_if end_if
```

Hint 04 test cases

Problem Statement 09

Given the following code, how much minimum number of test cases is required for full statement and branch coverage?

```
read p read q
if p+q> 100
then print "Large" endif
if p > 50
then print "p Large" endif
```

Hint 1 test for statement coverage, 2 for branch coverage

Problem Statement 10

Consider a program to input two numbers and print them in ascending order given below. Find all du paths and identify those du-paths that are not feasible. Also find all dc paths and generate the test cases for all paths (dc paths and non dc paths).

```
#include<stdio.h>
#include<conio.h>
   void main ()
   {
3 int a, b, t;
   Clrscr ();
   Printf ("Enter first number");
   scanf ("%d",&a);
   printf("Enter second number");
   scanf("%d",&b);
   if (a<b){
     t=a;
11a=b;
    b=t;
13}
     printf ("%d %d", a, b);
    getch ();
}
```

Problem Statement 11

Consider the above program and generate possible program slices for all variables. Design at least one test case from every slice.

Problem Statement 12

Consider the code to arrange the nos. in ascending order. Generate the test cases for relational coverage, loop coverage and path testing. Check the adequacy of the test cases through mutation testing and also compute the mutation score for each.

```
i = 0;
n=4; //N-Number of nodes present in the graph
While (i<n-1) do j = i + 1;
While (j<n) do
if A[i]<A[j] then swap (A[i], A[j]); end do;
i=i+1;
end do
```