AIM: Study of Control Statements

THEORY:

1). if ... else ... fi statement

It is a form of control statement that allows Shell to execute statements in a controlled way and make the right choice

SYNTAX:
```
if [expression]
then
    Statements
else
    Statements
fi
```

2). case ... esac statement

- This statement allows to give an expression which can be used to evaluate and to execute several different statements based on the value of the expression.
- The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

SYNTAX:
```
case word in
    pattern 1)
        Statements
    ;;
```

```
pattern 2).
        Statement
        ;;
        *).
        ;;
   esac.
```

3). LOGICAL OPERATORS:

i). Logical OR:

- The logical OR "-o" operator will give true if any one of the operand is true.
- If both operands are false then it will return false.

ii). Logical AND:

- The logical AND "-a" operator will give true if both the operands are true.
- Otherwise, false.

Eg i) if ['expr $a % 2' == 0 -a $a -gt 10]

ii). if ['expr $a % 2' == 0 -o $a -lt 10]

4). test command:

- test is used as part of the conditional execution of shell commands.
- test exits with the status determined by EXPRESSION.

- Placing the EXPRESSION between square brackets ([and]) is the same as testing the EXPRESSION with test.
  Eg. test 100 -lt 99 && echo "Yes."

5). WHILE LOOP:

- The loop keeps on executing the lines of code while an expression is true.

```
SYNTAX: while [ <some test> ]
          do
            <commands>
          done
```

6). UNTIL LOOP:

- The until loop is fairly similar to the while loop.
- The difference is that it will execute the commands within it until the test becomes true.

```
SYNTAX: until [ <some test> ]
          do
            <commands>
          done
```

7). FOR LOOP:

- For loop perform the given set of commands for each of the items in a given list.
- The for loop will take each item in the list (in order one after the other), assign that

item as the value of variable var, execute the commands between do and done then go back to the top, grab the next item in the list and repeat over.

SYNTAX :

```
for var in <list>
do
  <commands>
done
```