BCSE102L- Structured and object-oriented programming

Module 1

Dr. P.Keerthika
Associate Professor
School of Computer Science & Engineering
VIT, Vellore.

BCSE102L- Structured and object-oriented programming

Module:1	**C Programming Fundamentals**	**2 hours**
Variables - Reserved words – Data Types – Operators – Operator Precedence - Expressions - Type Conversions - I/O statements - Branching and Looping: if, if-else, nested if, if-else ladder, switch statement, goto statement - Loops: for, while and do…while – break and continue statements.		
Module:2	**Arrays and Functions**	**4 hours**
Arrays: One Dimensional array - Two-Dimensional Array – Strings and its operations. User Defined Functions: Declaration – Definition – call by value and call by reference - Types of Functions - Recursive functions - Storage Classes - Scope, Visibility and Lifetime of Variables.		
Module:3	**Pointers**	**4 hours**
Declaration and Access of Pointer Variables, Pointer arithmetic – Dynamic memory allocation – Pointers and arrays - Pointers and functions.		
Module:4	**Structure and Union**	**2 hours**
Declaration, Initialization, Access of Structure Variables - Arrays of Structure - Arrays within Structure - Structure within Structures - Structures and Functions – Pointers to Structure -		
Module:5	**Overview of Object-Oriented Programming**	**5 hours**
Features of OOP - Classes and Objects - "this" pointer - Constructors and Destructors - Static Data Members, Static Member Functions and Objects - Inline Functions – Call by reference - Functions with default Arguments - Functions with Objects as Arguments - Friend Functions and Friend Classes.		
Module:6	**Inheritance**	**5 hours**
Inheritance - Types of Inheritance: Single inheritance, Multiple Inheritance, Multi-level Inheritance, Hierarchical Inheritance - Multipath Inheritance - Inheritance and constructors.		
Module:7	**Polymorphism**	**4 hours**
Function Overloading - Operator Overloading – Dynamic Polymorphism - Virtual Functions - Pure virtual Functions - Abstract Classes.		
Module:8	**Generic Programming**	**4 hours**
Function templates and class templates, Standard Template Library.		
	Total Lecture hours:	**30 hours**

BCSE102L- Structured and object-oriented programming – Text Books and Reference Books

Text Book(s)	
1.	Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill Education, 2017
2.	Herbert Schildt, C++: The Complete Reference, 4th Edition, McGraw Hill Education, 2017.
Reference Books	
1.	Yashavant Kanetkar, Let Us C: 17th Edition, BPB Publicaitons, 2020.
2.	Stanley Lippman and Josee Lajoie, C++ Primer, 5th Edition, Addison-Wesley publishers, 2012.

BCSE102P- Structured and object-oriented programming Laboratory

Indicative Experiments	
1.	Programs using basic control structures, branching and looping
2.	Experiment the use of 1-D, 2-D arrays and strings and Functions
3.	Demonstrate the application of pointers
4.	Experiment structures and unions
5.	Programs on basic Object-Oriented Programming constructs.
6.	Demonstrate various categories of inheritance
7.	Program to apply kinds of polymorphism.
8.	Develop generic templates and Standard Template Libraries.

Text Book(s)	
1.	Robert C. Seacord, Effective C: An Introduction to Professional C Programming, 1st Edition, No Starch Press, 2020.
Reference Book(s)	
1.	Vardan Grigoryan and Shunguang Wu, Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20's latest features, 1st Edition, Packt Publishing Limited, 2020.

BCSE102L- Structured and Object-Oriented Programming

- **Module-1:C Program Fundamentals - Introduction**
  - **Variables**
  - **Reserved Words**
  - **Data types**
  - **Operators- Operator Precedence**
  - **Expressions**
  - **Type Conversions**
  - **I/O Statements**

BCSE102L- Structured and Object-Oriented Programming

- **Module-1: C Program Fundamentals – Branching and Looping**
  - if, if-else, nested if, else-if ladder
  - Switch Statement
  - Goto Statement
  - Looping
    - For
    - While and do while
  - Break Statement
  - Continue Statement

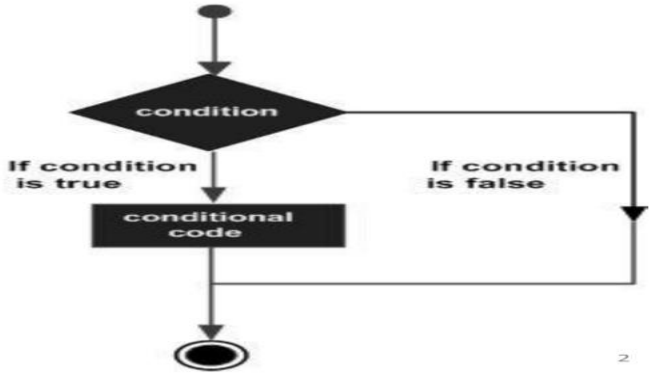
BRANCHING STATEMENTS

- **Conditional Branching**
  - Called as selection or decision control statements.
  - It contains the statements that validate the validity of an expression before running the code.
    - Simple if
    - If-else
    - If – else if
    - Nested if else
    - Switch
- **UnConditional Branching**
  - Pass to statements unconditionally from one point to another in the program.
    - Goto
    - Return
    - Break
    - Continue

Simple if

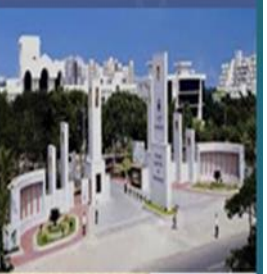
Syntax:

```
if (test expression)
{
        statement block;
}
```




```
#include < stdio.h>
void main( )
{
        int age;
        scanf("%d",&age);
        if (age>= 18 )
        {
        printf("Eligible to vote");
        }
        printf("Not Eligible to Vote");
}
```

```
#include < stdio.h>
void main( )
{
        ------------------
        ------------------
        if (student_quota == sports)
        {
        ent_marks= ent_marks+ 10;
        }
        printf("%f", ent_marks);
}
```



Simple if - Examples

//count number of boys in class whose weight is less than or equal to 60 and height greater than or equal to 160.

```
#include < stdio.h>
void main( )
{
        float weight, height; int count=0,
        printf(" weight & height of Boy 1:");
        scanf("%f%f", &weight, &height);
        if(weight<=60)&&(height>=160)
                count= count+1;
        printf(" weight & height of Boy 2:");
        scanf("%f%f", &weight, &height);
        if(weight<=60)&&(height>=160)
                count= count+1;

        ------------------------
        ------------------------
        ------------------------

        printf("Number of boys whose weight <=60 and >=160 is: %d",count);
        }
}
```

Need of LOOPING?????

Class strength= 60 means??

Simple if - Examples

//count number of boys in class whose weight is less than or equal to 60 and height greater than or equal to 160.

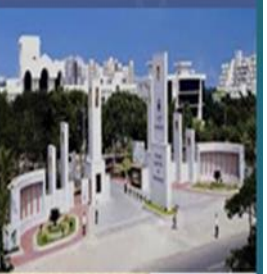
```
#include < stdio.h>
void main( )
{
        float weight, height; int count=0, i;
        printf(" weight & height of  60 Boys: \n");

        for(i=1;i<=60;i++)
        {
                scanf("%f%f", &weight, &height);
                if(weight<=60)&&(height>=160)
                count= count+1;
        }

        printf("Number of boys whose weight <=60 and >=160 is: %d",count);

}
```

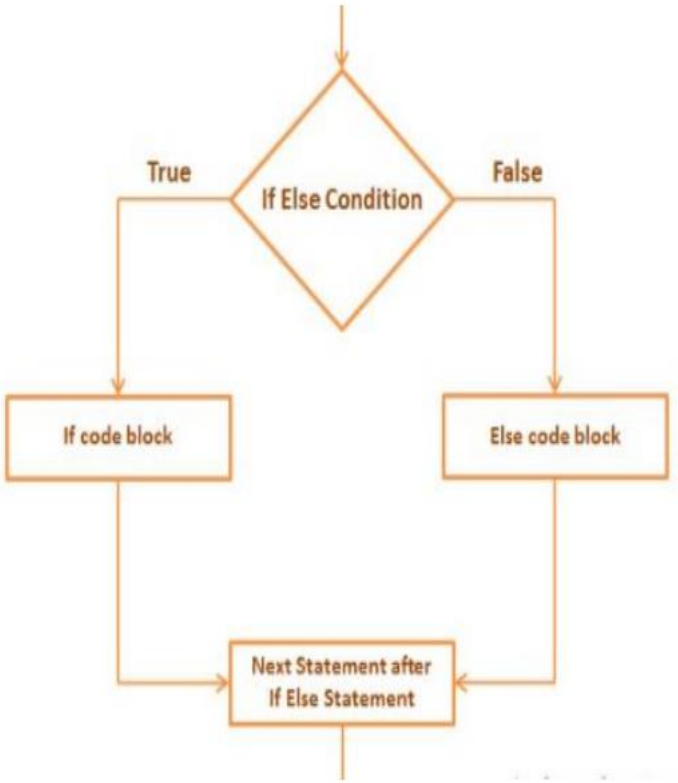
Execution of repeated set of statements

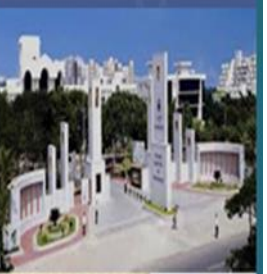


If-Else

Syntax:

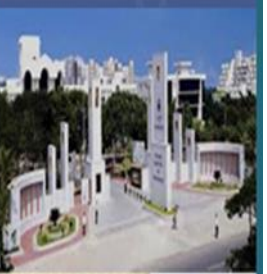
```
if (expression)
{
        statement block1;
}
else
{
         statement block2;
}
```





If-Else – Example 1

```
Void main( ) // Biggest of 2 numbers.
{
        int x,y;
        scanf("%d %d ",&x, &y);// Getting input from user
        if (x>y )    // test expression
        {
                printf("x is bigger");// True Block
        }
        else
        {
                printf("b is bigger"); // False Block
        }
}
```



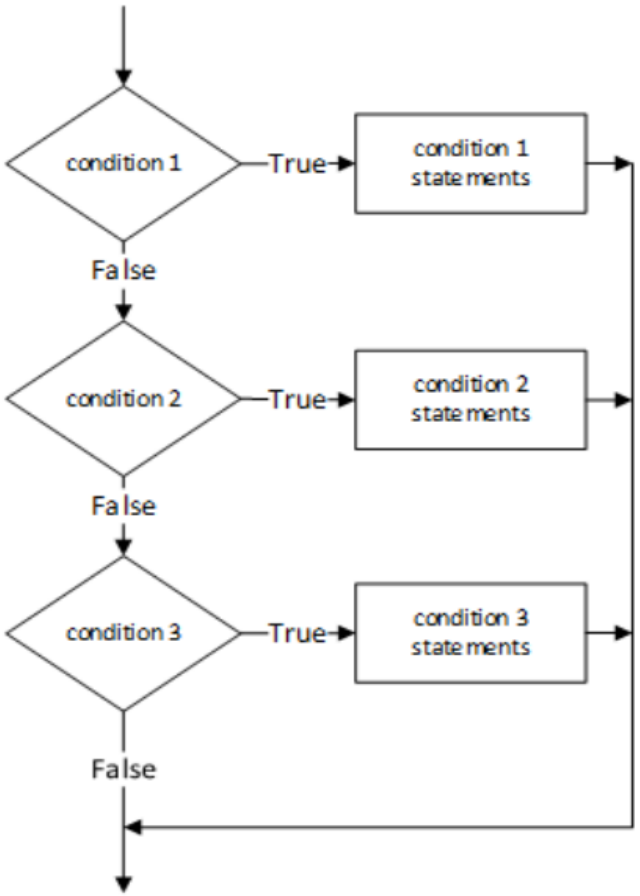
If-Else – Example 2

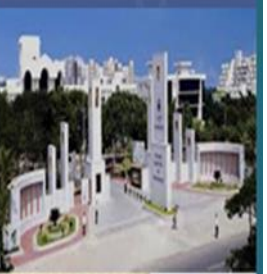
```
int main()
{
        float basic_sal, gross_sal, da, hra ;
        printf ( "Enter basic salary " ) ;
        scanf ( "%f", &basic_sal ) ;
        if (basic_sal < =50000)
        {                                          // TRUE BLOCK
                hra = basic_sal * 10/100 ;
                da = basic_sal * 50 / 100 ;
        }
        else
        {                                          // FALSE BLOCK
                hra = basic_sal * 15/100 ;
                da = basic_sal * 60/ 100 ;
        }
        gross_sal = basic_sal + hra + da ;
        printf ("Gross salary = Rs. %f\n", gross_sal) ;
return 0 ;
}
```

If-Else if statement

Syntax:

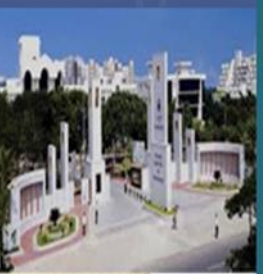
```
if (expression)
{
        statement block1;
}
else if( expression)
{
         statement block2;
}
else
        statements
```





If-Else if Example - I

```
int main()  // Biggest of three numbers
{
        int a,b,c; ;
        printf ("Enter the value for a, b and c:");
        Scanf("%d %d %d", &a , &b, &c);
        if (a>b))   // if Part
        {
                if(a>c)
                        printf("a is bigger");
                else
                        printf("c is bigger ");
        }
        else if(b>c)    // else if part
                printf("b is bigger");
        else                    // else part
                printf("c is bigger");
}
```



If-Elseif – Example 2

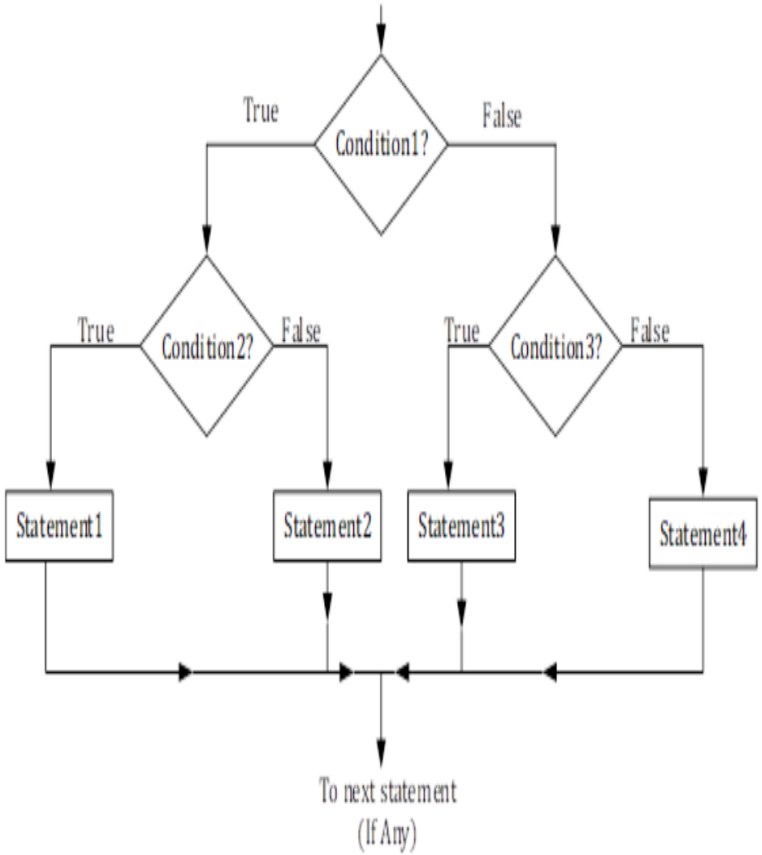
```
int main()
{
 int year;
 scanf("%d", &year);
 if((year % 400==0) // divisible by 400
            printf(" Given %d year is a leap Year", year);
else if(year%100==0) // divisible by 100 but not by 400
            printf(" Given %d year is not a leap Year", year);
else if(year%4==0)) not divisible by 100 but divisible by 4
            printf(" Given %d year is a leap year", year);
else
             printf(" Given %d year is not a leap year", year);
return 0;
}
```

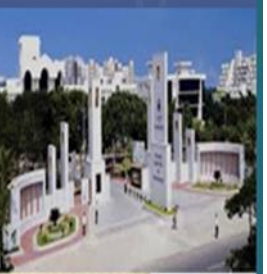
Nested if else statement

- When a series of decisions are involved , we may have to use more than one if..else statement in a nested form.

Syntax:

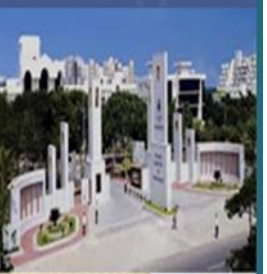
```
if (expression)
{
        if(expression 1)
        {
        statement block1;
        }
        else
        {
         statement block2;
        }
else
{
        statement block3;
}
```





Nested If-Else Example

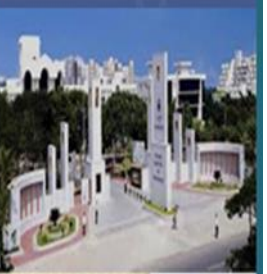
```
int main()  // Biggest of three numbers
{
        int a,b,c; ;
        printf ("Enter the value for a, b and c:");
        Scanf("%d %d %d", &a , &b, &c);
        if (a>b))
        {
                if(a>c)
                        printf("a is bigger");
                else
                        printf("c is bigger ");
        }
        else
        {
                if(b>c)
                        printf("b is bigger");
                else
                        printf("c is bigger");
        }
}
```



Else if Ladder statement

Syntax:

```
if (expression1)
        statement 1;
else if (expression2)
        statement 2;
else if (expression3)
        statement 3;
.....
.....
else if (expression n)
        statement n;
else
        default statement ;
Statement x;
```

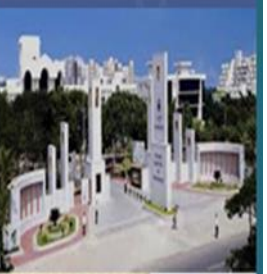


Else if Ladder statement

Syntax:

```
if (grade >=9)
        category = “Outstanding”;
else if (grade <9 && >=8))
        category = “Excellent”;
else if (grade <8 && >=7))
        category = “Very good”;
else if (grade <7 && >=6))
        category = “Good”;
else if (grade <6 && >=5))
        category = “Average”;
else
        category = “Fail”;

Printf(“%s\n”, category);
```

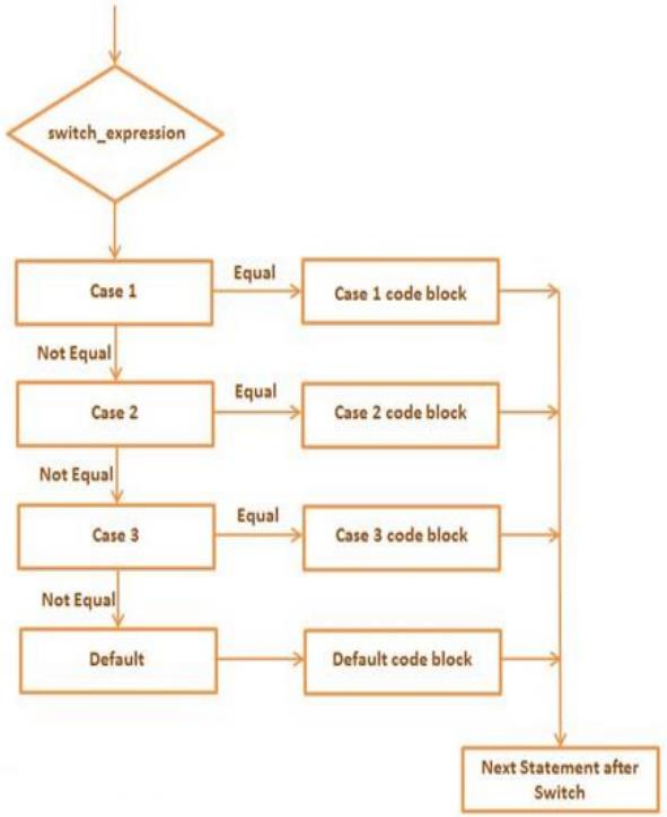


Switch statement

- When the number of alternatives increases, using if statement will increase the complexity. Switch is a built-in multiway decision statement. It test the value of a given variable against a list of case values and when a match is found, the block of statement associated with the case will be executed.

Syntax:

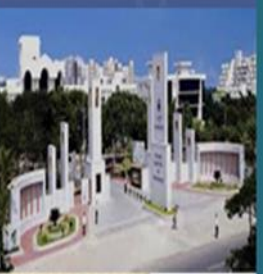
```
switch (expression)
{
        Case Constant1 :
                statement;
                break;
        Case Constant2 :
                statement;
                break;
        Case Constant3 :
                statement;
                break;
        default: /*Optional*/
                statement;
}
Statement x;
```



Switch statement - Example

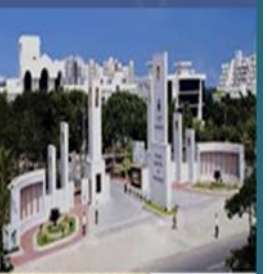
```
int main()
{
int day = 4;
switch (day)
{
  case 1:
    printf("Monday");
    break;
  case 2:
    printf("Tuesday");
    break;
  case 3:
    printf("Wednesday");
    break;
  case 4:
    printf("Thursday");
    break;
  case 5:
    printf("Friday");
    break;
  case 6:
    printf("Saturday");
    break;
  case 7:
    printf("Sunday");
    break;
  default:
     Printf(“ Wrong Choice”);
}
```

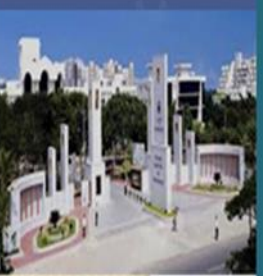

```
int main()
{
int a,b,operation;
printf(" 1.Addition\n 2.Subtraction\n 3.Multiplication\n 4.Division\n");
printf("Enter the values of a & b: ");
scanf("%d %d",&a,&b);
printf("Enter your Choice : ");
scanf("%d",&operation);
switch(operation)
        {
              case 1 :
              printf("Sum of %d and %d is : %d",a,b,a+b);
              break;
              case 2 :
              printf("Difference of %d and %d is : %d",a,b,a-b);
              break;
              case 3 :
              printf("Multiplication of %d and %d is : %d",a,b,a*b);
              break;
              case 4 :
              printf("Division of Two Numbers is %d : ",a/b);
              break;
              default :
              printf(" Wrong choice!!!!! Enter Your Correct Choice.");
              break;
          }
return 0;
 }
```



Switch statement - Example

```
int main()
{
        char character;
        scanf("%c",&character)
        printf("TRAVEL INFORMATION:");
        switch (character)
        {
                case 'A':
                Airways-information(); // Function call
                break;
                case 'B':
                Bus-information();
                break;
                case 'C':
                Train-information();
                break;
                case 'D':
                Hotel-information();
                break;
                default:
                Printf("Wrong Choice");
        }
}
```





Goto statement

- Previous cases- Flow of execution is based on certain specified conditions.
- **Goto-** branch unconditionally from one point to another.
- Requirement: It needs a **label** in order to identify the place where the branch is made.
- Label is any valid variable name, must be followed by **":"**
- **Where it is placed?**
- It is placed immediately before the statement where the control is transferred.

General forms of Goto statement


```
int main()
{
        float x,y;
        read:
           scanf("%f",&x);
        if(x<0)
                goto read;
        y=sqrt(x);
        printf("%f %f\n", x,y);
}
```

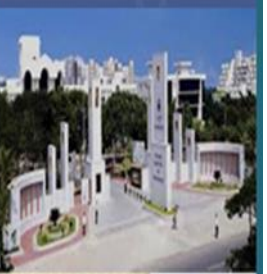
```
int main()
{
    int x=1;
    label:
      printf("%d\n",x+10);
    x=x+1;
    if(x<10)
       goto label;
  printf("Exit because x= %d\n", x);
}
```

return

- Returning control to the caller function puts an end to the execution of a function.
- The calling function may also get a value from a return statement.
- The respective function will terminate and return a value to its caller when you use a return statement.
- **Syntax:**

  return;

  return expression;

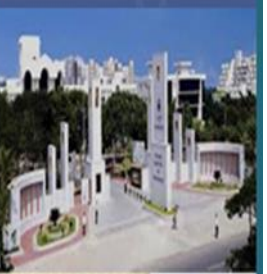


break

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- Break statement terminates the execution of the nearest enclosing **do, for, switch, or while** statement in which it appears.
- Control passes to the statement that follows the terminated statement.

- **Syntax:**

  break;

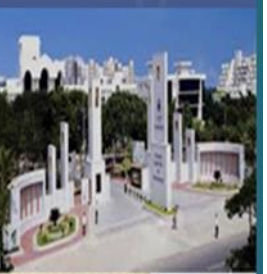


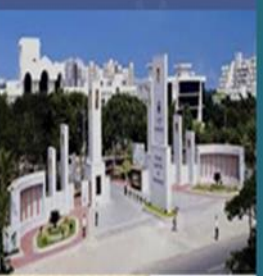
Break – how it works

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```





continue

- Continue statement skips any code in between and executes the next iteration of the loop.
- Continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

- **Syntax:**

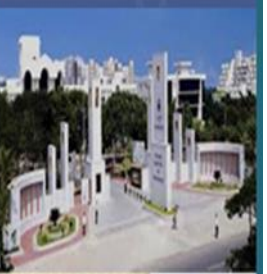
```
continue;
```

Continue – how it works

```
while (test expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
```

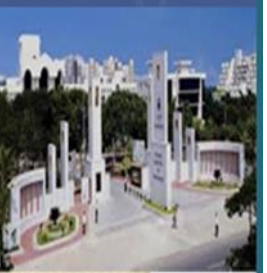
```
do {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statements/
}
```



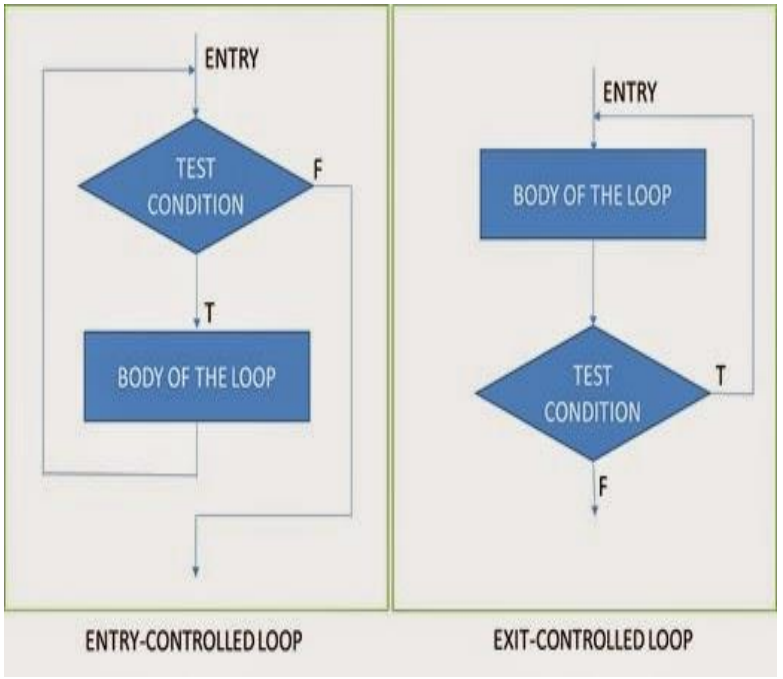
Difference between break and Continue

Break	Continue
Break is used to **terminate** the execution of the loop.	Continue is not used to terminate the execution of loop.
It **breaks** the iteration.	It **skips** the iteration.
When this statement is executed, control will come out from the loop and executes the statement immediate after loop.	When this statement is executed, it will not come out of the loop but moves/jumps to the next iteration of loop.
Break is used with loops as well as switch case.	Continue is only used in loops, it is not used in switch case.



Looping Statements

- A looping statement allows us to execute a statement or group of statements multiple times.
- The program loop consists of two segments
  - Control statement
  - Body of the loop
- Three Categories
  - **For**
  - **While**
  - **Do while**



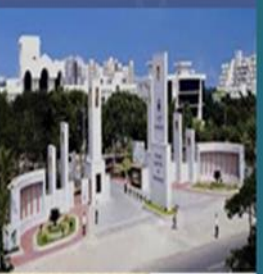
while Loop

- Condition is checked at the beginning of the loop, *i.e. Entry Check*
- Syntax:

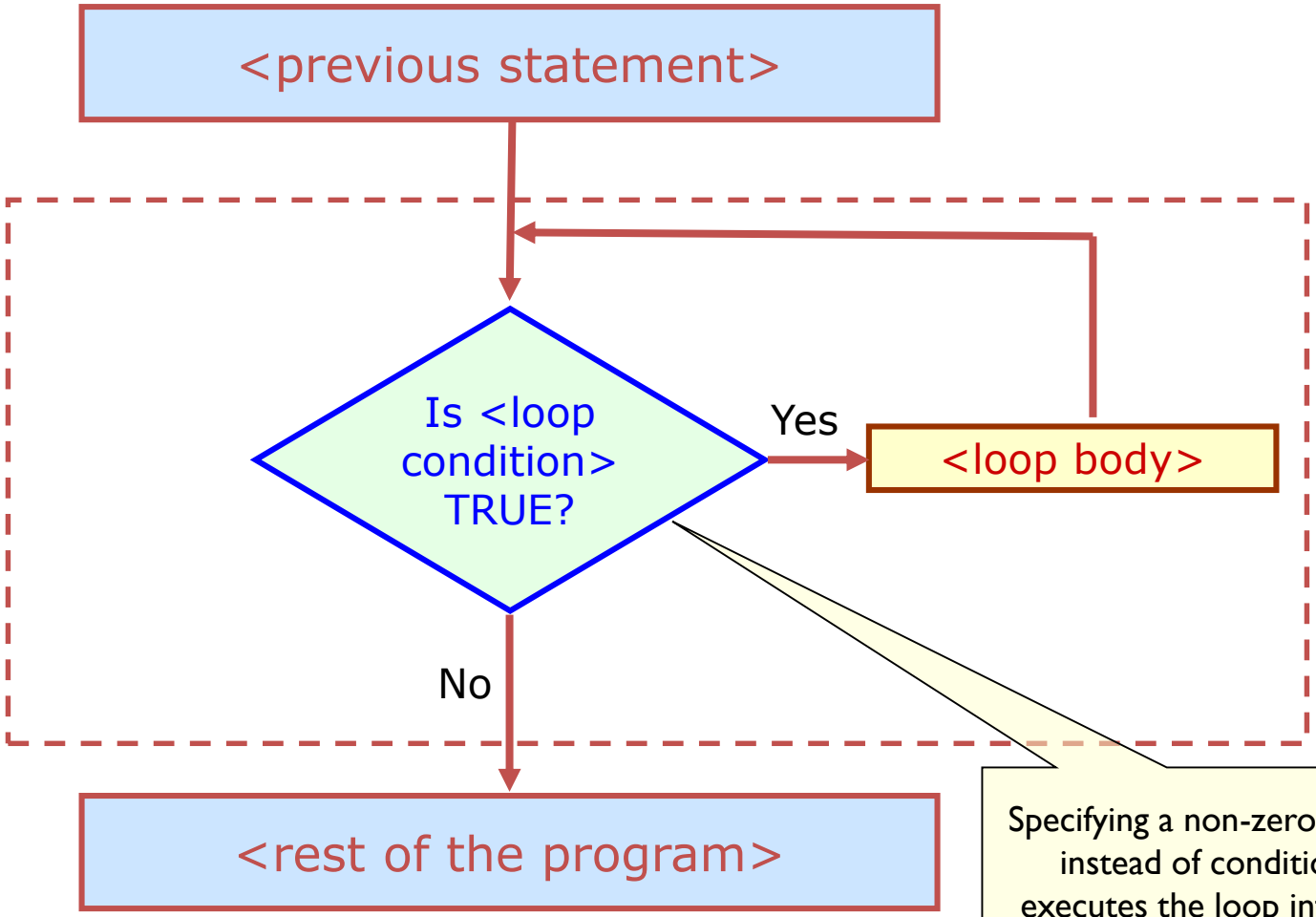
```
while(<condition>)
{
        <statement>;
        <statement>;
        …
}
```

Do not give a semicolon(;) here

- The body of while loop is executed till the condition becomes false
- The body of while loop is executed zero or more times, depending on the condition

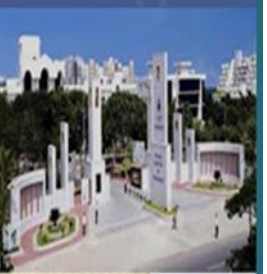


while Loop



while Loop - Example

```
int main()
{
        int Count = 0; /* Variable Declaration */
        while (Count < 10) /* Working of while */
        {
                printf("Value of Count = %d \n",Count);
                Count++;
        }
    return 0;
}
```



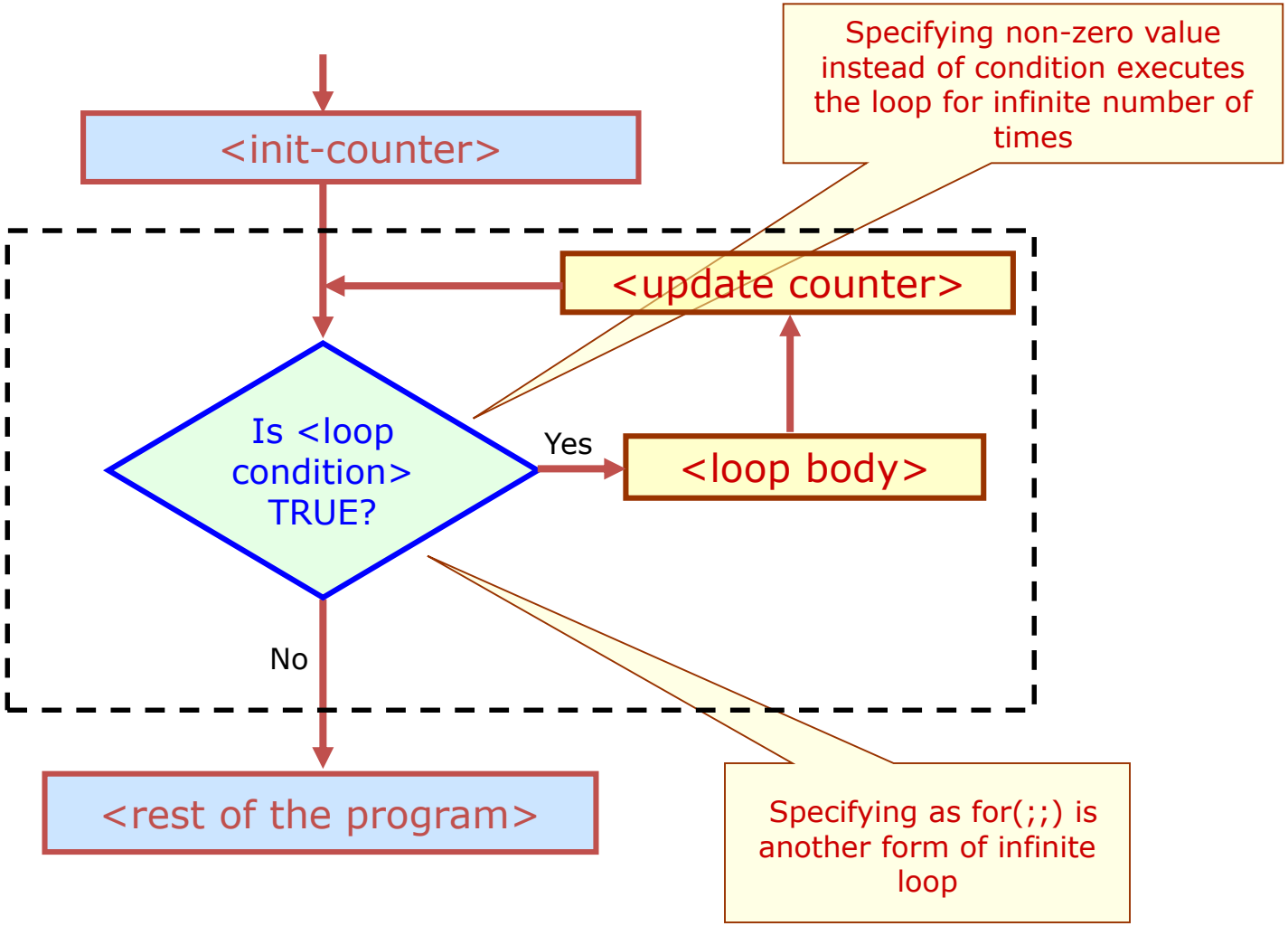
for Loop

- Provides a compact syntax for iteration
- Allows you to specify the following, all on one line:
  1. Initialization statement(s)
  2. Condition i.e Boolean expression for continuing loop
  3. Statements to be executed after loop
- Condition is checked at the starting of loop, *i.e. Entry Check*
- Syntax:

```
for(<initialization>;<condition>;<action statements>)
{
        <statement>;
        <statement>;
        …
}
```

Specifying (;) is mandatory

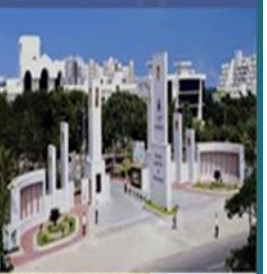
for Loop


<init-counter>
<update counter>
Is <loop condition> TRUE?
Yes
<loop body>
No
<rest of the program>
Specifying non-zero value instead of condition executes the loop for infinite number of times
Specifying as for(;;) is another form of infinite loop

For Loop - Example

```
int main()
{
        int count; /*Variable Declaration */
        for(count=0, count<10; count++) /*Working of For*/
        {
                printf("Value of Count = %d \n",count);
        }
    return 0;

}
```



do-while Loop

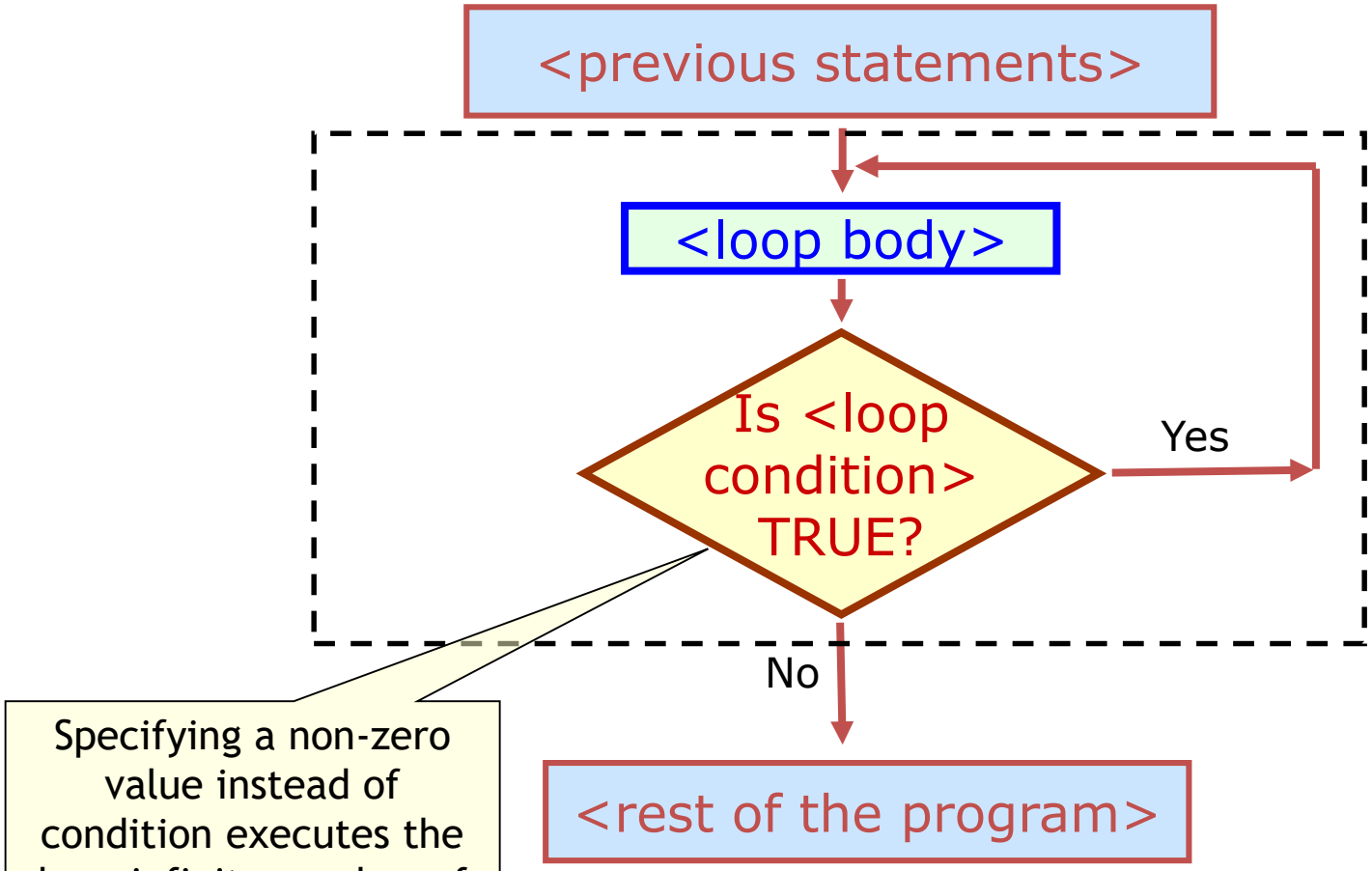
- Condition is checked at the end of loop, *i.e. Exit Check*
- Syntax:

```
do
{
        <statement>;
        <statement>;
        ……..……;
} while(<condition>);
```

Do not forget to give semicolon(;) here

- The body of ***do-while* loop** is executed continuously, till the condition becomes false.
- ***do-while* loop** is executed *at least once,* irrespective of condition being true or false, since the condition is checked at the end of loop

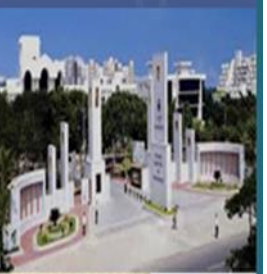
do-while Loop



Specifying a non-zero value instead of condition executes the loop infinite number of times

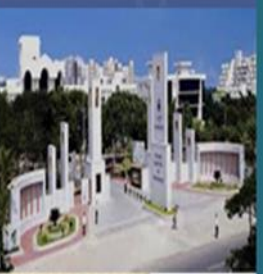
do-while Loop - Example

```
int main()
{
        int Count = 0; /*Variable Declaration */
        do
        {
                printf("Value of Count = %d \n",Count);
                Count++;
        } while (Count < 10) ;/*Working of  do-while */
    return 0;
}
```



do-while Loop - Example

```
int main()
{
        int Count = 0; /*Variable Declaration */
        do
        {
                printf("Value of Count = %d \n",Count);
                Count++;
        } while (Count < 10) ;/*Working of  do-while */
    return 0;
}
```

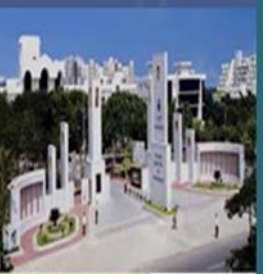


Infinite Loop - Example

```
do
{
    ..............
} while (1);

while(-1)
{
    ..............
}

for ( ; ;)
{
    ..............
    ..............
}
```



Other forms of Looping - Example

- while (i <= 10)
- while (i >= 10 && j <= 15) // using relational / logical
- while (j > 10 && (b < 15 || c < 20))

- for (x = 1 ; x <= 3 ; x = x+ 1) // Normal representation
- for (i = 10 ; i ; i --)
  printf ("%d ", i) ;
- for (scanf("%d", &i) ; i <= 10 ; i++)
  printf ("%d", i) ;
- for (i = 1 ; i <= 10 ;)
  printf ("%d\n", i) ;
- for (; i <= 10 ;)
  printf ("%d\n", i) ;

Additional Features of For loop

- More than one variable can be initialized at a time in the for statement.

```
x=1;
for(i=0;i<=20; i++)
```

→

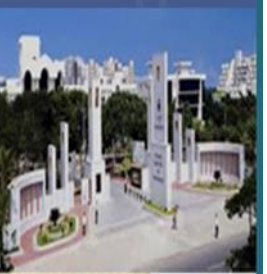
```
for( x=1, i=0;i<=20,i++)
```

- Like initialization section, increment section also more than one part.

```
for( x=1,y=50; x<=y; x++, y--)
```

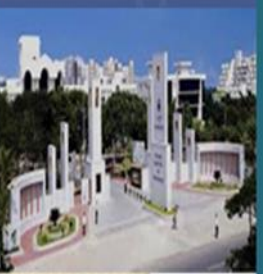
- Test condition may also have compound relation.

```
for( x=1;  x<=10&& sum<100;  x++)
```



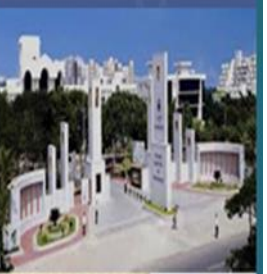
Nested Loops

- It is often necessary for one loop to include another loop to solve a problem
- Both loops need separate initializations, boolean expressions & increments.
- If the outer loop executes N times and the inner loop executes M times, the inner loop body is executed N x M times.
  - for loop within another for loop
  - two while loops can also be nested.
  - a for loop can occur within a while loop, or a while within a for.



Nested Loops

```
int main()
{
    int i,j;          // variable declaration
    for (i=0; i< 20; i++)  // Start of Outer loop
    {
        for (j=0; j < 30; j++) // Start of inner loop
        {
            printf("*");
        }                          // End of Inner For Loop
     }                             // End of Outer For Loop
     return 0;
}                                  // End of Function main()
```

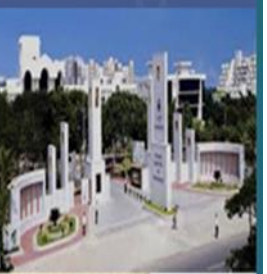


Do's & Don'ts for looping statements

- Extra semicolon after *for (..)* or *while (..)* causes a zero line infinite loop.
- Counters (and other variables) not properly initialized cause infinite loop and other errors.
- Never update a *for* loop counter variable inside the *for* loop
- Never use the same counter variable for nested loops
- Don't use the *goto* statement as it abruptly changes the flow of the program

Practice Exercises

1. Find the roots of quadratic equation.
2. Calculate sum of all numbers greater than 100 and less than 200 that are divisible by 8.
3. Calculate sum of n natural numbers. Get the value of n from user
4. To make as simple calculator using Switch statements.
5. Generation of Multiplication table.
6. To check whether the given number is Armstrong number or not.
7. To check whether the given number is prime or not
8. Printing of Fibonacci series.
9. Printing sum of the given digits.
10. Find the factorial of given number.



VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Thank You