CL1002
Programming Fundamentals

Lab 05
Nested Decision Structures

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2025

AIMS AND OBJECTIVES

Aims:

Manage Complex Decision Making:

- To handle complex decision processes where multiple conditions must be checked in a specific order, often depending on earlier decisions.

Increase Program Flexibility:

- To allow more nuanced and flexible control flows in a program, enabling the program to respond differently based on combinations of multiple conditions.

Create Modular Logic:

- To build programs that can handle decisions in a modular way, separating different logical branches for clearer and more organized code.

Achieve Precise Outcomes:

- To ensure that the program can execute precise operations by evaluating a series of conditions and making the right decisions based on the context.

Objectives:

Efficiently Handle Multiple Scenarios:

- To allow the program to handle multiple scenarios or conditions effectively without requiring excessive repetition of code.

Simplify Complex Condition Checking:

- To break down complicated decisions into smaller, more manageable conditions, making the code easier to understand and maintain.

Support Hierarchical Decision Logic:

- To support decisions that depend on previous outcomes, where a specific condition can only be evaluated if another has already been satisfied or failed.

Optimize Code Readability and Maintenance:

- Structure code in a way that is easier to read and maintain, with clearly defined decision-making paths.

Minimize Redundancy:

- To reduce code duplication by nesting decisions, allowing for more efficient logic where common conditions can be reused in different decision branches.

INTRODUCTION

In this lab we will be covering the following topics

1. Introduction to Nested Decision Structures
2. Nested If-Else Statements
3. Nested Switch-Case Statements
4. Operators in C
5. Introduction to Math Library Functions

OBJECTIVE: 1 INTRODUCTION TO NESTED DECISION STRUCTURES

In programming, decision-making structures are fundamental for controlling the flow of execution based on certain conditions. Nested decision structures refer to a form of decision-making where one decision statement is placed

inside another. This allows a program to make more complex decisions by evaluating multiple conditions in a hierarchical or dependent manner.

When a program requires more than a simple yes-or-no decision, nested decisions come into play. For instance, certain actions may only be taken if an initial condition is satisfied, and within that, further decisions may need to be made. These structures are essential when dealing with real-world problems, where multiple factors often influence the outcome.

OBJECTIVE: 2 NESTED IF-ELSE STATEMENTS

Nested decision structures are frequently used in various fields such as data processing, game development, business applications, and AI, where multiple layers of conditions need to be evaluated to arrive at the correct decision. By using these structures, programmers can write more efficient and adaptable code, which can handle complex logic seamlessly.

The use of nested decision structures introduces clarity and organization in complex decision-making processes, making the program not only functional but also easier to maintain and expand.

Placing the block of if else statement inside an existing if or else block statement is called nested If else statement. Each block of nested if else, logically perform same as simple if else statements. Whenever a user wants to check more than one condition at a time, the appropriate way is to use nested if-else statements.

Following is the structure of nested if else statement in an algorithm.

```
1     IF (logical-expression) THEN
2        statements
3        IF (logical-expression) THEN
4           statements
5        ELSE
6           statements
7        END IF
8        statements
9     ELSE
10       statements
11       IF (logical-expression) THEN
12          statements
13       END IF
14       statementsObjective: 3 Opening a pull request
```

EXAMPLE 1

Let’s consider the problem of finding the largest value if three variables are given from the user.

```
1   Input X, Y,Z
2   If(X>Y) then
3       If(X>Z) then
4               Max= X [X>Y, X>Z]
5       Else
6               Max= Z [Z>X>Y]
7       Endif
8   Else
9       If(Y>Z) then
10              Max = Y [Y>X, Y>Z]]
11      Else
12              Max = Z [Z>Y>X]
13      Endif
14  Endif
15  Print "The largest number is", Max
```

Flowchart:

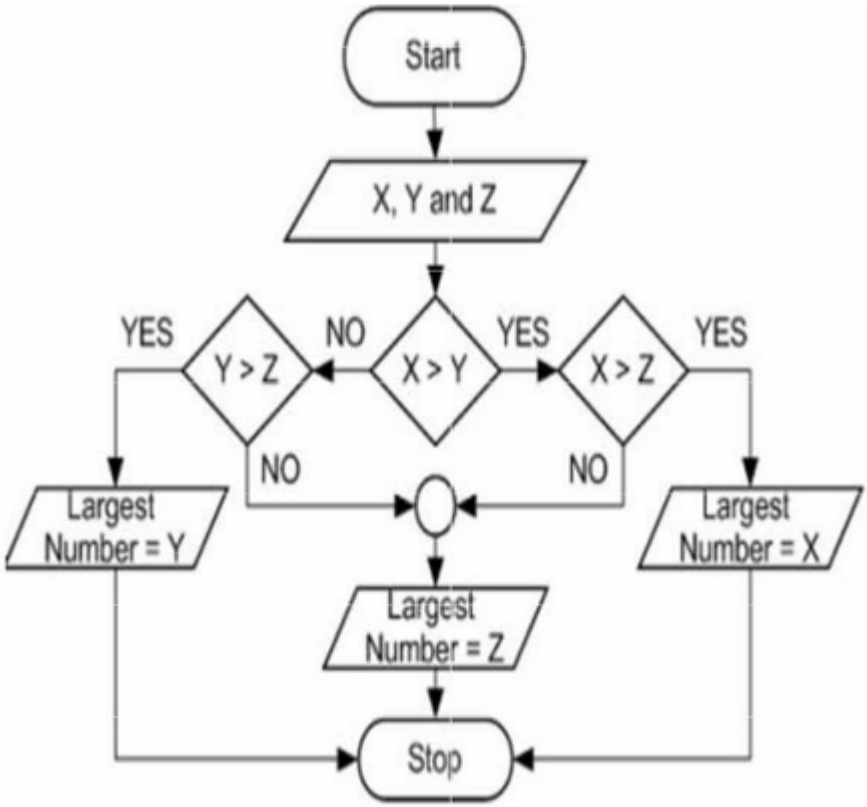

Figure: 2 Flowchart of Nested If-Else Statement

C-Implementation:

```c
#include<stdio.h>

int main(){
    int x,y,z;
    printf("Enter value of X");
    scanf("%d",&x);
    printf("Enter value of Y");
    scanf("%d",&y);
    printf("Enter value of Z");
    scanf("%d",&z);
    if(x>y){
            if(x>z){
                    printf("The largest value is of x = %d",x);
            }
            else{
                    printf("The largest value is of z =%d", z);
            }
    }
    else{
            if(y>z){
                    printf("The largest value is of y= %d",y);
            }
            else{
                    printf("The largest value is of z= %d",z);
            }
    }
  return 0;
}
```

OBJECTIVE: 3 NESTED SWITCH-CASE STATEMENTS

Placing the simple switch case statements inside an existing case statement is called nested switch-case statement. Each block of nested switch case statement logically performs the same as simple switch case statement. Following is the structure of nested switch-case statement in an algorithm.

```c
Switch(controlling expression){
    Label set 1:
            Statement 1;
            Break;
    Label set 2:
            Statement 2;
            Switch(controlling expression){
                    Label set 1:
                            Statement 1;
                            Break;
                    Label set 2:
                            Statement 2;
                            Break;
                    Default:
                            Statement d;
            }
```

```
        Break;
    Default:
        Statement d;
}
```

EXAMPLE 2

Problem: Ayesha is interested in knowing the names of different countries. She wants a list of countries by just giving a starting and ending letter.

C-Implementation

```
#include <stdio.h>
int main()
{
    char start,e;
    printf("Please say starting letter of country");
    scanf("%c",&start);
    switch(start)
    {
        case 'A':
        case 'a':
            printf("Please say ending letter\n");
            scanf("\n%c",&e);
            switch(e)
            {
                case 'A':
                case 'a':
                    printf("\n Alaska \n Albania \n Algeria");
                    break;
                default:
                    printf("\n No such country");
            }
            break;

        case 'B':
        case 'b':
            printf("Please say ending letter\n");
    scanf("\n%c",&e);
            switch(e)
            {
                case 'A':
                case 'a':
                    printf("\n Bulgeria \n Bolivia \n Botswana");
                    break;
                default:
                    printf(" No such country");


            }
                    break;

```

```
97              default:
98                      printf("Please type correct letter");
99              }
100 }
```

OBJECTIVE: 4 TERNARY OPERATORS IN C

ARITHMETIC OPERATORS

There are many operators in C for manipulating data which include arithmetic Operators, Relational Operators, Logical operators and many more which will be discussed accordingly. Some of the fundamental operators are:

Operator	Description	Example
+	Adds two operands.	A + B = 30
–	Subtracts second operand from the first.	A – B = -10
*	Multiplies both operands.	A * B = 200
/	Divides 1st operand by 2nd operand.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

RELATIONAL OPERATORS

A relational operator checks the relationship between two operands. If the relation is true, it returns 1 (true); if the relation is false, it returns value 0 (false). Relational operators are used in decision making and loops

Operator	Meaning of Operator	Example
==	Is Equal to	5 == 3 returns 0
>	Is Greater than	5 > 3 returns 1
<	Is Less than	5 < 3 returns 0
!=	Is Not equal to	5 != 3 returns 1

>=	Is Greater than or equal to	5 >= 3 returns 1
<=	Is Less than or equal to	5 <= 3 returns 0

LOGICAL OPERATORS

An expression containing a logical operator returns either 0 (false) or 1 (true) depending upon whether the expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0 (false)	If c = 5 then, expression !(c==5) equals to 0.

BITWISE OPERATORS

The computer runs on binary, thus on the lowest level all operations are performed on bits. In C language, you can perform these operations using bitwise operators.

Operator	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

TERNARY OPERATORS

A ternary operator is a shorthand way of writing an if-else conditional statement in a single line of code. It is often used to simplify the code, making it more concise and readable in situations where you want to assign a value based on a condition.

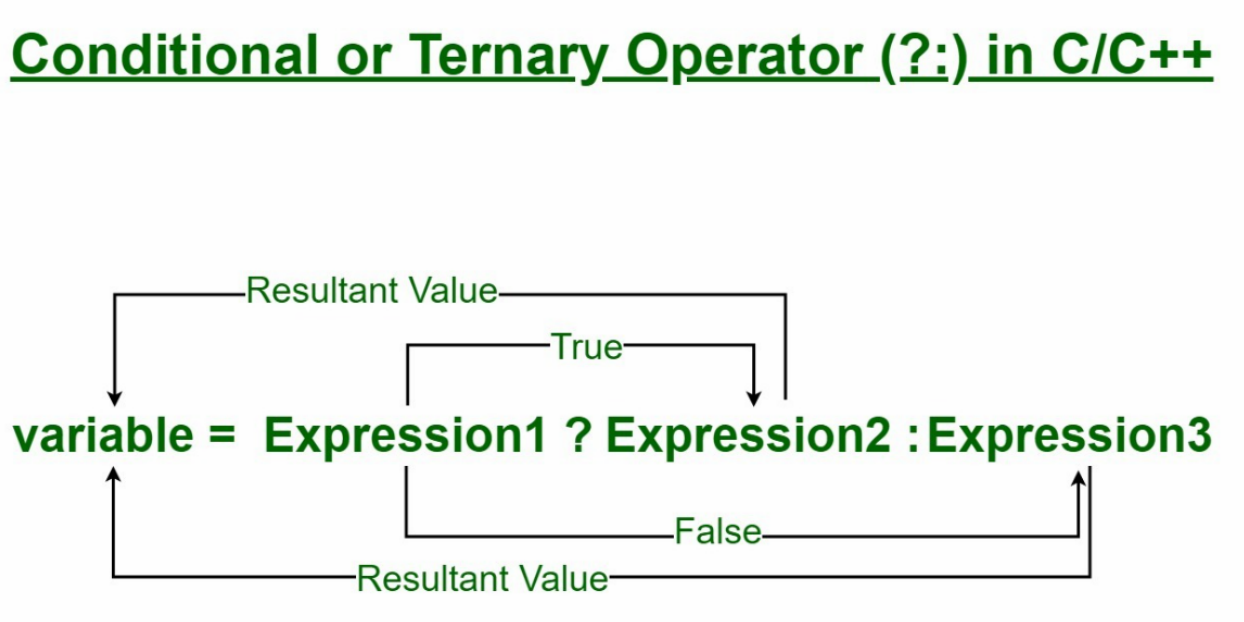

Figure: 4(a) Syntax of Ternary Operators in C

There can be variations of the ternary operators in C. The conditional operator is of the form

```
1   variable = Expression1 ? Expression2 : Expression3
```

Or the syntax will also be in this form

```
101 variable = (condition) ? Expression2 : Expression3
```

Or syntax will also be in this form

```
102 (condition) ? (variable = Expression2) : (variable = Expression3)
```

The conditional operator is kind of like the if-else statement as it does follow the same algorithm as of if-else statement, but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

Flow Chart of a ternary operator operations in C.

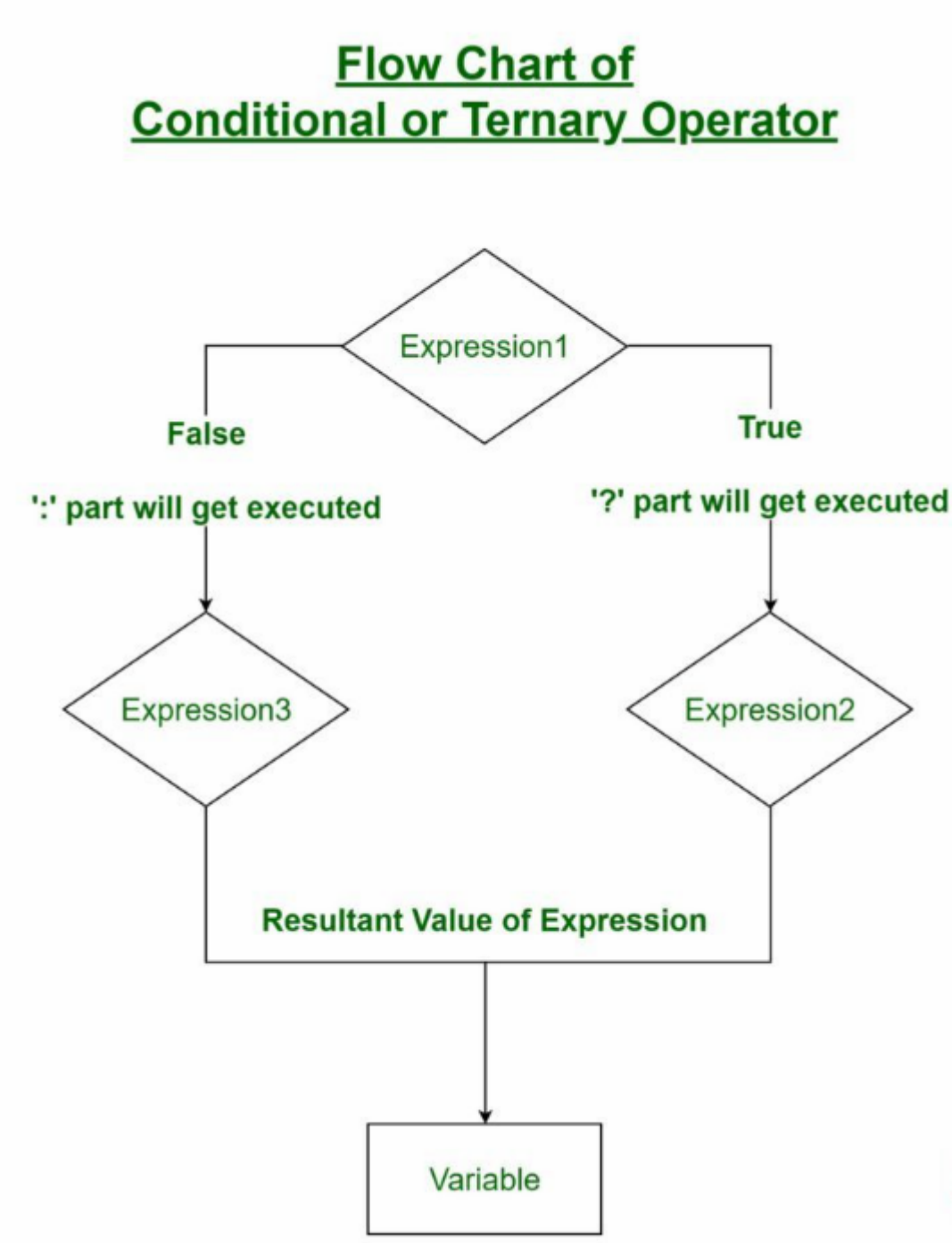

Figure: 4(b) Flowchart of Ternary Operators

EXAMPLE 3:

```
1   #include <stdio.h>
103
104 int main()
105 {
106     int m=5, n=4;
107
108     (m>n) ? printf(“M is greater than n”) : printf(“n is greater than M);
109     return 0;
110 }
```

EXAMPLE 4

```
111 #include <stdio.h>
112
113 int main()
114 {
115     int a=1, b=2, ans;
116     //Nested Ternary operator
117     ans=(a==1?(b==2?3:5):0);
118     Printf(“%d\n”,ans);
119     return 0;
120 }
```

COMMA OPERATOR

Comma operators are used to link related expressions together. For example: int a, c = 5, d;

SIZEOF OPERATOR

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

OPERATOR PRECEDENCE

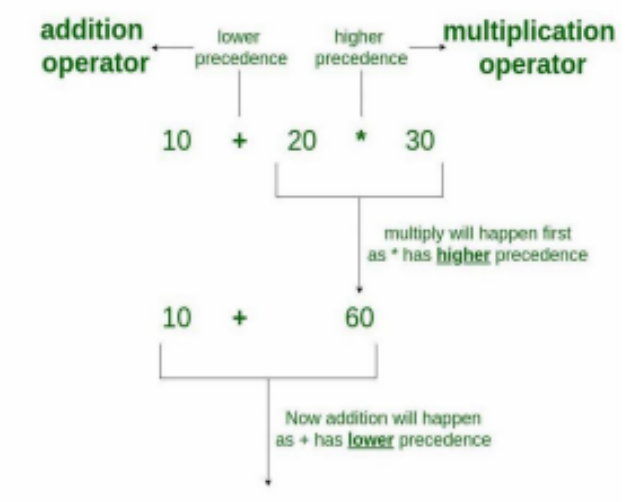

Figure: 4(c) Operator Precedence Example

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

In the table below, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

OBJECTIVE: 5 MATH LIBRARY FUNCTIONS

#include <math.h>

Math library functions allow you to perform certain common mathematical calculations.

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is 30.0 `sqrt( 9.0 )` is 3.0
`cbrt( x )`	cube root of x (C99 and C11 only)	`cbrt( 27.0 )` is 3.0 `cbrt( -8.0 )` is -2.0
`exp( x )`	exponential function e^x	`exp( 1.0 )` is 2.718282 `exp( 2.0 )` is 7.389056
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is 1.0 `log( 7.389056 )` is 2.0
`log10( x )`	logarithm of x (base 10)	`log10( 1.0 )` is 0.0 `log10( 10.0 )` is 1.0 `log10( 100.0 )` is 2.0
`fabs( x )`	absolute value of x as a floating-point number	`fabs( 13.5 )` is 13.5 `fabs( 0.0 )` is 0.0 `fabs( -13.5 )` is 13.5
`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is 10.0 `ceil( -9.8 )` is -9.0
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is 9.0 `floor( -9.8 )` is -10.0
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is 128.0 `pow( 9, .5 )` is 3.0
`fmod( x, y )`	remainder of x/y as a floating-point number	`fmod( 13.657, 2.333 )` is 1.992
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is 0.0
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is 1.0
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is 0.0

Figure: 5 Commonly used Math Library Functions