Programming for Problem Solving

Workbook

Week	Session #	Mentor	Topic
1	1	Dr.T.Upender	Operators, Precedence & Associativity
	2		Operators, Precedence & Associativity
	3		Decision Control Statements-if-else
2	4	Mrs.M.Sailaja	Decision Control Statements-if-else
	5		Decision Control Statements-Switch-Case
	6	Dr.E.V.N.Jyothi	Iterative statements-while
3	7	Mr.K.Sudhakar Reddy	Iterative statements-for
	8		Iterative statements-do-while
	9		Nested loops-patterns
4	10		Loop control statements
	11		Functions
	12	Mr.M.Shiva Kumar	Recursive Functions
5	13		Scope & Life time of variables
	14	Dr.R.Ravikumar	1D-Array-Average, max, min
	15		1D-Array-frequency, reversing largest element, Palindrome
6	16		Analysis of the Sorting-Bubble Sort
	17		Analysis of Sorting-Insertion Sort, Selection Sort
	18		Analysis of Linear Search, Binary Search
7	19	Dr.Y.Ambica	Multidimensional array-Matrix Addition
	20		Multidimensional array-Matrix Multiplication
	21		Multidimensional array-Transpose of a Matrix

Session-1

Topic: Precedence & Associativity

Q1. Suppose a, b, c are integer values that have been assigned the values a=8,b=3,

c = -5. Determine the value of each of the following expressions

i) 2*b+3*(a-c) ii)(a * c)% b iii) a* (b / c) iv) iii) a * b / c iv) a / b % c

Q2. Suppose a, b, c are integer values that have been assigned the values a=1,b=2,

c=3 then what is the value of c after the following statement is executed?

c + = (a > 0 && a <= 10)? ++a : a/b.

Q3. Suppose a = 10, b = 20, c = 30; what is the result of the expression c > b > a

Q4. a = 10, b = 5, c = 2; then what is the result of a > b && b > c;

Q5.if x = 5; then what is the result of x = 10 + x * 2;

Q6.if x = 10, y = 5, then what is the result of (x > y) ? (x - y) : (y - x);

Q7.if a = 1, b = 0, c = 1; then what is the result of a && b | | c;

Q8.if a = 5, b = 10, then what is the result of ((a*2)+(b/2));

Q9. What is the result of the expression 1 = = 3 ! = 5

Q10.What is the result of the expression 5 + 2 < 10;

END

Session-2

Topic: Operators

Q1. If a = 20, then compute the value of the following expression.

i)+a, ii)-a iii)++a iv)—a v)a++ vi) a—

Q2.if a = 20, b = 4, then compute the value of the following expression.

i)a<b ii)a>b iii)a<=b iv)a>=b v)a==b vi) a!=b

Q3.if a = 20, b = 5; then compute the value of the following expression

i)a && b ii)a || b iii)!a

Q4.if a = 20, b = 5; then compute the value of the following expression

i)a&b ii)a|b iii)a^b iv)~a v)a>>b vi) a<<b

Q5.if a = 20, b = 5; then compute the value of the following expression

1. a += b
2. a -= b
3. a *= b
4. a /= b
5. a %= b
6. a &= b
7. a |= b
8. a ^= b
9. a >>= b
10. a <<= b

Q6. What is the result of the expression a = (1, 2, 3);

Q7.What is the result of the expression z = (x = 10, y = 20, x + y);

Q8.if a = 1, b = 2, c=0 then after executing the expression c = (a += 5, a + b, b = a + 10); what are the values of a,b,c

Q9.if x=1,y=2 then evaluating the expression z=(x++, y++, x + y) what is the value of z?

Q10. What is the outcome of the expression (a = a + 2, b = b + 3, a + b > 10)

END

Session-3

Topic: Decision Control Statements-if else

Example 1: Assignment inside if

```
#include <stdio.h>
int main() {
   int x = 0;
   if (x = 5)   // Assignment, not comparison
      printf("True branch executed, x = %d\n", x);
   else
      printf("False branch executed\n");
   return 0;
}
```

****

Example 2: Nested if–else with ambiguity (dangling else problem)

```
#include <stdio.h>
int main() {
   int a = 5, b = 10;
   if (a > 0)
      if (b < 5)
         printf("b is small\n");
      else
         printf("else belongs to inner if\n");
   return 0;
}
```

****

Example 3: Multiple conditions with tricky precedence

```
#include <stdio.h>
int main() {
   int x = 5, y = 0;
```

```
    if (x && y || ++y)
        printf("Condition true, y = %d\n", y);
    else
        printf("Condition false, y = %d\n", y);
    return 0;
}
```

****

Example 4: Confusing else if chain

```
#include <stdio.h>
int main() {
    int n = 0;
    if (n > 0)
        printf("Positive\n");
    else if (n >= 0)
        printf("Zero or more\n");
    else
        printf("Negative\n");
    return 0;
}
```

****

Example 5: If with comma operator

```
#include <stdio.h>
int main() {
    int x = 0;
    if (x = 0, x = 5, x > 2)
        printf("True, x = %d\n", x);
    else
        printf("False, x = %d\n", x);
    return 0;
}
```

****

Example 6: Boolean-like trick

```
#include <stdio.h>
int main() {
   int a = -1;
   if (a)
      printf("a is treated as TRUE\n");
   else
      printf("a is FALSE\n");
   return 0;
}
```

****

Example 7: Nested ternary inside if

```
#include <stdio.h>
int main() {
   int x = 5;
   if (x > 0 ? x < 10 : x > -10)
      printf("Condition true\n");
   else
      printf("Condition false\n");
   return 0;
}
```

****

Example 8: Operator Precedence

```
#include <stdio.h>
int main() {
   int a = 0, b = 1, c = 0;
   if (a || b && c)
      printf("Condition true\n");
   else
      printf("Condition false\n");
   return 0;
}
```

****

Example 9:Comma operator inside if

```
#include <stdio.h>
int main() {
   int x = 0;
   if (x = 0, x = 3, x > 5)
      printf("True, x = %d\n", x);
   else
      printf("False, x = %d\n", x);
   return 0;
}
```

****

Example 10:Negative values in if

```
#include <stdio.h>
int main() {
   int x = -5;
   if (x)
      printf("x is treated as true\n");
   else
      printf("x is false\n");
   return 0;
}
```

****

Example 11:Nested ternary inside if

```
#include <stdio.h>
int main() {
   int n = -3;
   if (n > 0 ? n < 10 : n > -10)
      printf("Condition true\n");
```

```
    else
        printf("Condition false\n");
    return 0;
}
```

****

Example 12:Nested ternary inside if

```
#include <stdio.h>
int main() {
    int a = 5;
    if (++a > 5)
        printf("a = %d\n", a);
    else
        printf("Not greater\n");
    return 0;
}
```

****

Session-4

Topic: Decision Control Statements-if else

Example 1: Overlapping conditions

```
#include <stdio.h>
int main() {
    int n = 10;
    if (n > 0)
        printf("Positive\n");
    else if (n > 5)    // Never executed
        printf("Greater than 5\n");
    else
        printf("Negative or Zero\n");
    return 0;
}
```

****

Example 2: Multiple true conditions

```
#include <stdio.h>
int main() {
   int x = 15;
   if (x % 3 == 0)
      printf("Divisible by 3\n");
   else if (x % 5 == 0)
      printf("Divisible by 5\n");
   else if (x % 15 == 0)
      printf("Divisible by 15\n");
   return 0;
}
```

****

Example 3: Nested if with dangling else

```
#include <stdio.h>
int main() {
   int a = 5, b = 10;
   if (a > 0)
      if (b < 5)
         printf("b is less than 5\n");
      else
         printf("Else belongs to inner if\n");
   return 0;
}
```

****

Example 4: Nested if with shadowing

```
#include <stdio.h>
int main() {
   int a = 5, b = 20;
```

```
    if (a > 0) {
        if (b > 10)
            b = -1;
    } else if (b > 0)
        b = 100;
    else
        b = 200;

    printf("b = %d\n", b);
    return 0;
}
```

****

Example 5: Nested if with confusing braces

```
#include <stdio.h>
int main() {
    int x = 0, y = 1;
    if (x)
        if (y)
            printf("x and y are true\n");
        else
            printf("x is true but y is false\n");
    else
        printf("x is false\n");
    return 0;
}
```

****

Example 6: Else-if ladder with same variable

```
#include <stdio.h>
int main() {
    int score = 85;
    if (score > 90)
      printf("Grade A\n");
    else if (score > 80)
      printf("Grade B\n");
    else if (score > 70)
      printf("Grade C\n");
    else
      printf("Grade D\n");
    return 0;
}
```

****

Example 7: Nested if with modification

```
#include <stdio.h>
int main() {
  int a = 2, b = 3;
  if (a > 1)
    if (b > 2)
      a = a + b;
    else
      a = a - b;
  else
    a = 0;
  printf("a = %d\n", a);
  return 0;
}
```

END

Example 1: Missing break (fall-through)

```
#include <stdio.h>
int main() {
   int x = 2;
   switch(x) {
      case 1: printf("One\n");
      case 2: printf("Two\n");
      case 3: printf("Three\n");
      default: printf("Default\n");
   }
   return 0;
}
```

****

Example 2: Duplicate case values (Error)

```
#include <stdio.h>
int main() {
   int n = 1;
   switch(n) {
      case 1: printf("First\n"); break;
      case 2: printf("Second\n"); break;
      case 1: printf("Duplicate\n"); break;   //
   }
   return 0;
}
```

****

****

Example 3: Case with expressions

```
#include <stdio.h>
int main() {
   int x = 2;
   switch(x) {
      case 1+1: printf("Matched 2\n"); break;
      case 4/2: printf("Matched 2 again\n"); break;  // Error: duplicate
      default: printf("No match\n");
   }
   return 0;
}
```

****

Example 4: Char vs Int

```
#include <stdio.h>
int main() {
   char ch = 'A';  // ASCII 65
   switch(ch) {
      case 65: printf("Case 65\n"); break;
      case 'A': printf("Case A\n"); break;   // duplicate
   }
   return 0;
}
```

****

Example 5: Switch with no matching case

```
#include <stdio.h>
int main() {
   int n = 100;
   switch(n) {
      case 10: printf("Ten\n"); break;
      case 20: printf("Twenty\n"); break;
   }
   return 0;
}
```

****

Example 6: Variable case label (Error)

```
#include <stdio.h>
int main() {
   int x = 2, y = 3;
   switch(x) {
      case y: printf("y = 3\n"); break;  //Error
      default: printf("Default\n");
   }
   return 0;
}
```

****

Example 7: Nested switch

```
#include <stdio.h>
int main() {
   int x = 1, y = 2;
   switch(x) {
      case 1:
         switch(y) {
            case 2: printf("Inner 2\n"); break;
            default: printf("Inner default\n");
         }
         break;
      default: printf("Outer default\n");
   }
   return 0;
}
```

****

Example 8: Default in middle

```
#include <stdio.h>
int main() {
   int x = 2;
   switch(x) {
      case 1: printf("One\n"); break;
      default: printf("Default\n");    // in middle
      case 2: printf("Two\n"); break;
   }
   return 0;
}
```

****

Example 9: Fall-through puzzle

```
#include <stdio.h>
int main() {
   int n = 1;
   switch(n) {
      case 0: printf("Zero\n");
      case 1: printf("One\n");
      case 2: printf("Two\n");
      default: printf("Default\n");
   }
   return 0;
}
```

****

Example 10: Multiple labels for one block

```
#include <stdio.h>
int main() {
   int n = 3;
   switch(n) {
      case 1:
      case 2:
      case 3: printf("One, Two, or Three\n"); break;
      default: printf("Something else\n");
   }
   return 0;
}
```

END

Session-6

Topic: Iterative statements-while loop

Example 1: Assignment inside while

```
#include <stdio.h>
int main() {
   int x = 0;
   while (x = 5)   // assignment, not comparison
      printf("Inside loop, x = %d\n", x);
   return 0;
}
```

****

Example 2: Missing update → Infinite loop

```
#include <stdio.h>
int main() {
   int i = 1;
   while (i <= 5) {
      printf("%d ", i);
      // i++ missing → infinite loop
   }
   return 0;
}
```

****

Example 3: Post-increment inside condition

```
#include <stdio.h>
int main() {
   int i = 0;
   while (i++ < 5)
      printf("%d ", i);
   return 0;
}
```

****

Example 4: Pre-increment inside condition

```
#include <stdio.h>
int main() {
   int i = 0;
   while (++i < 5)
      printf("%d ", i);
   return 0;
}
```

****

Example 5: Floating-point while loop

```
#include <stdio.h>
int main() {
   float x = 0.1;
   while (x != 1.0) {
      printf("%.1f ", x);
      x += 0.1;
      if (x > 2.0) break;
   }
   return 0;
}
```

****

Example 6: Empty body while

```
#include <stdio.h>
int main() {
   int i = 0;
   while (i++ < 5);   // note semicolon
   printf("i = %d\n", i);
   return 0;
}
```

****

Example 7: Break inside while

```
#include <stdio.h>
int main() {
   int i = 0;
   while (1) {
      if (i == 3)
         break;
      printf("%d ", i);
      i++;
   }
   return 0;
}
```

****

Example 8: Continue inside while

```
#include <stdio.h>
int main() {
   int i = 0;
   while (i < 5) {
      i++;
      if (i == 3)
         continue;
      printf("%d ", i);
   }
   return 0;
}
```

****

Example 9: Nested while loops

```
#include <stdio.h>
int main() {
   int i = 1, j;
   while (i <= 3) {
      j = 1;
      while (j <= i) {
         printf("* ");
         j++;
      }
      printf("\n");
      i++;
   }
   return 0;
}
```

****

Example 10: While with comma operator

```
#include <stdio.h>
int main() {
   int i = 0, j = 5;
   while (i < 3, j > 0) {
      printf("i=%d, j=%d\n", i, j);
      i++;
      j--;
   }
   return 0;
}
```

****

Topic: Iterative statements-for loop

Example 1: Missing all three parts

```
#include <stdio.h>
int main() {
   for (;;)   // equivalent to while(1)
      printf("Hello\n");
   return 0;
}
```

****

Example 2: Multiple initializations and updates

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 0, j = 5; i < j; i++, j--)
      printf("i=%d j=%d\n", i, j);
   return 0;
}
```

****

Example 3: Update inside body instead of header

```
#include <stdio.h>
int main() {
   int i;
   for (i = 0; i < 5;) {   // no i++ in header
      printf("%d ", i);
      i++;             // increment inside body
   }
   return 0;
}
```

****

Example 4: Condition always true

```
#include <stdio.h>
int main() {
   int i;
   for (i = 0; i >= 0; i++)   // i is always >= 0 (until overflow)
      if (i == 5) break;
   printf("Stopped at i = %d\n", i);
   return 0;
}
```

****

Example 5: Empty body loop

```
#include <stdio.h>
int main() {
   int i;
   for (i = 0; i < 5; i++);   // semicolon = empty loop
   printf("i = %d\n", i);
   return 0;
}
```

****

Example 6: Using comma operator

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 0, j = 10; i < j; i++, j--)
      printf("i=%d j=%d\n", i, j);
   return 0;
}
```

****

Example 7: For loop without condition

```
#include <stdio.h>
int main() {
   int i = 0;
   for (; ; i++) {
      if (i == 3) break;
      printf("%d ", i);
   }
   return 0;
}
```

****

Example 8: Reverse loop with tricky update

```
#include <stdio.h>
int main() {
   int i;
   for (i = 5; i--;)   // condition = i-- (decrements, checks old value)
      printf("%d ", i);
   return 0;
}
```

****

Example 9: Floating-point for loop

```
#include <stdio.h>
int main() {
   float x;
   for (x = 0.1; x != 1.0; x += 0.1) {
      printf("%.1f ", x);
      if (x > 2.0) break;
   }
   return 0;
}
```

****

Example 10: Nested for loops making a pattern

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 1; i <= 3; i++) {
      for (j = 1; j <= i; j++)
         printf("* ");
      printf("\n");
   }
   return 0;
}
```

END

Session-8

Topic: Iterative statements-for loop patterns

Example 1: Right-angled triangle of stars

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 1; i <= 5; i++) {        // rows
      for (j = 1; j <= i; j++)       // columns
         printf("* ");
      printf("\n");
   }
   return 0;
}
```

****

Example 2: Inverted triangle

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 5; i >= 1; i--) {
      for (j = 1; j <= i; j++)
         printf("* ");
      printf("\n");
   }
   return 0;
}
```

****

Example 3: Pyramid pattern

```
#include <stdio.h>
int main() {
   int i, j, space;
   for (i = 1; i <= 5; i++) {
      for (space = 5; space > i; space--)   // spaces
         printf(" ");
      for (j = 1; j <= (2*i - 1); j++)     // stars
         printf("*");
      printf("\n");
   }
   return 0;
}
```

****

Example 4: Diamond pattern

```
#include <stdio.h>
int main() {
   int i, j, space, n = 5;
   // upper half
   for (i = 1; i <= n; i++) {
      for (space = n; space > i; space--)
         printf(" ");
      for (j = 1; j <= (2*i - 1); j++)
         printf("*");
      printf("\n");
   }
   // lower half
   for (i = n-1; i >= 1; i--) {
      for (space = n; space > i; space--)
         printf(" ");
      for (j = 1; j <= (2*i - 1); j++)
         printf("*");
      printf("\n");
   }
   return 0;
}
```

****

Example 5: Number triangle

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 1; i <= 5; i++) {
      for (j = 1; j <= i; j++)
         printf("%d ", j);
```

```
        printf("\n");
    }
    return 0;
}
```
****

Example 6: Floyd's triangle

```
#include <stdio.h>
int main() {
    int i, j, num = 1;
    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d ", num);
            num++;
        }
        printf("\n");
    }
    return 0;
}
```
****

Example 7: Hollow square

```
#include <stdio.h>
int main() {
    int i, j, n = 5;
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            if (i == 1 || i == n || j == 1 || j == n)
                printf("* ");
            else
                printf("  ");
        }
        printf("\n");
    }
    return 0;
}
```

****

Example 8: Pascal's triangle (binomial coefficients)

```
#include <stdio.h>
int main() {
    int i, j, n = 5, num;
    for (i = 0; i < n; i++) {
        num = 1;
        for (j = 0; j <= i; j++) {
            printf("%d ", num);
            num = num * (i - j) / (j + 1);
        }
        printf("\n");
    }
    return 0;
}
```

Session-9

Topic: Iterative statements-do-while

Example 1: Runs even if condition is false

```
#include <stdio.h>
int main() {
    int i = 5;
    do {
        printf("Executed once even though condition is false!\n");
    } while (i < 0);   // false condition
    return 0;
}
```

****

Example 2: Infinite loop with semicolon

```
#include <stdio.h>
int main() {
   int i = 0;
   do {
      printf("Hello\n");
   } while (1);   // always true
   return 0;
}
```

****

Example 3: Trick with empty body

```
#include <stdio.h>
int main() {
   int i = 1;
   do;   // empty loop body
   while (i++ <= 3);
   printf("i = %d\n", i);
   return 0;
}
```

****

Example 4: Post-increment in condition

```
#include <stdio.h>
int main() {
   int i = 1;
   do {
      printf("%d ", i);
   } while (i++ < 5);
   return 0;
}
```

****

Example 5: Pre-increment in condition

```
#include <stdio.h>
int main() {
   int i = 1;
   do {
      printf("%d ", i);
   } while (++i < 5);
   return 0;
}
```

****

Example 6: Nested do–while

```
#include <stdio.h>
int main() {
   int i = 1, j;
   do {
      j = 1;
      do {
         printf("(%d,%d) ", i, j);
         j++;
      } while (j <= 2);
      printf("\n");
      i++;
   } while (i <= 3);
   return 0;
}
```

****

Example 7: Condition modifies variable

```
#include <stdio.h>
int main() {
   int n = 10;
   do {
      printf("%d ", n);
   } while ((n /= 2) > 0);   // n is halved each time
   return 0;
}
```

****

Example 8: Printing digits of a number (reverse order)

```
#include <stdio.h>
int main() {
   int n = 1234;
   do {
      printf("%d ", n % 10);  // print last digit
      n /= 10;                // remove last digit
   } while (n > 0);
   return 0;
}
```

****

Example 9: Character input until ENTER key

```
#include <stdio.h>
int main() {
   char ch;
   printf("Enter characters (press ENTER to stop):\n");
   do {
      ch = getchar();
      putchar(ch);
   } while (ch != '\n');
   return 0;
}
```

****

Example 10: Trick with break inside do–while

```
#include <stdio.h>
```

```
int main() {
   int i = 1;
   do {
      if (i == 3)
         break;   // exit early
      printf("%d ", i);
      i++;
   } while (i <= 5);
   return 0;
}
```

Session-10

Topic: loop control statements

Example 1: break in a for loop

```
#include <stdio.h>
int main() {
   int i;
   for (i = 1; i <= 10; i++) {
      if (i == 5)
         break;   // exits loop completely
      printf("%d ", i);
   }
   return 0;
}
```

****

Example 2: continue skipping values

```
#include <stdio.h>
int main() {
   int i;
   for (i = 1; i <= 10; i++) {
      if (i % 2 == 0)
         continue;   // skip even numbers
```

```
        printf("%d ", i);
    }
    return 0;
}
```

****

Example 3: break inside nested loops

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 3; i++) {
        for (j = 1; j <= 3; j++) {
            if (j == 2)
                break;   // breaks inner loop only
            printf("(%d,%d) ", i, j);
        }
        printf("\n");
    }
    return 0;
}
```

****

Example 4: continue inside nested loops

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 3; i++) {
        for (j = 1; j <= 3; j++) {
            if (j == 2)
                continue;   // skip when j=2
            printf("(%d,%d) ", i, j);
        }
        printf("\n");
    }
    return 0;
}
```

****

Example 5: goto to exit nested loops

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 3; i++) {
        for (j = 1; j <= 3; j++) {
            if (i == 2 && j == 2)
                goto end;   // jump out of both loops
            printf("(%d,%d) ", i, j);
        }
    }
end:
    printf("\nExited at (2,2)");
    return 0;
}
```

****

Example 6: break with while loop

```
#include <stdio.h>
int main() {
    int n = 1;
    while (1) {   // infinite loop
        if (n > 5)
            break;   // exit when condition met
        printf("%d ", n);
        n++;
    }
    return 0;
}
```

****

Example 7: continue with while loop

```
#include <stdio.h>
int main() {
   int n = 0;
   while (n < 5) {
      n++;
      if (n == 3)
         continue;   // skip printing when n=3
      printf("%d ", n);
   }
   return 0;
}
```

****

Example 8: do–while with continue

```
#include <stdio.h>
int main() {
   int i = 0;
   do {
      i++;
      if (i == 3)
         continue;   // skips only printing, but loop continues
      printf("%d ", i);
   } while (i < 5);
   return 0;
}
```

****

Example 9: for loop with multiple controls and continue

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 0, j = 5; i < j; i++, j--) {
        if (i == 2)
            continue;   // skip when i=2
        printf("i=%d j=%d\n", i, j);
    }
    return 0;
}
```

****

Example 10: Trick with goto inside loop

```
#include <stdio.h>
int main() {
    int i = 1;
loop:
    if (i > 5)
        return 0;
    printf("%d ", i);
    i++;
    goto loop;   // jumps back to label
}
```

****

Session-11

Topic: Functions

Example 1: Function without return type

```
#include <stdio.h>
fun() {   // no return type → implicitly int (older C standard)
    return 5;
}
int main() {
    printf("%d", fun());
    return 0;
}
```

****

Example 2: Function returning multiple values (via pointer)

```
#include <stdio.h>
void divide(int a, int b, int *q, int *r) {
    *q = a / b;
    *r = a % b;
}
int main() {
    int q, r;
    divide(17, 3, &q, &r);
    printf("Quotient=%d Remainder=%d\n", q, r);
    return 0;
}
```

****

Example 3: Recursive function with tricky base case

```
#include <stdio.h>
int fun(int n) {
    if (n == 0) return 0;
    return n + fun(n - 1);
}
int main() {
   printf("%d", fun(5));
   return 0;
}
```

****

Example 4: Infinite recursion (no base case)

```
#include <stdio.h>
void fun() {
   printf("Hello ");
   fun();   // no base case → stack overflow
}
int main() {
   fun();
   return 0;
}
```

****

Example 5: Function inside another function call

```
#include <stdio.h>
int square(int x) {
   return x * x;
}
int add(int a, int b) {
   return a + b;
}
int main() {
   printf("%d", add(square(3), square(4)));
   return 0;
}
```

Example 6: Static variable inside function

```
#include <stdio.h>
void fun() {
   static int count = 0;  // retains value between calls
   count++;
   printf("%d ", count);
}
int main() {
   fun();
   fun();
   fun();
   return 0;
}
```

****

Example 7: Call by value vs call by reference

```
#include <stdio.h>

void changeVal(int x) {   // call by value
   x = 100;
}
void changeRef(int *x) {  // call by reference
   *x = 100;
}
int main() {
   int a = 10, b = 10;
   changeVal(a);
   changeRef(&b);
   printf("a=%d b=%d\n", a, b);
   return 0;
}
```

****

Example 8: Function pointer

```c
#include <stdio.h>
int add(int a, int b) {
   return a + b;
}
int main() {
   int (*fp)(int, int);  // function pointer
   fp = add;
   printf("%d", fp(3, 4));
   return 0;
}
```

****

Example 9: Mutual recursion

```c
#include <stdio.h>
void funA(int n);
void funB(int n);
void funA(int n) {
   if (n <= 0) return;
   printf("%d ", n);
   funB(n - 1);
}
void funB(int n) {
   if (n <= 0) return;
   printf("%d ", n);
   funA(n / 2);
}
int main() {
   funA(10);
   return 0;
}
```

****

Example 10: Function returning pointer to local variable (dangerous)

```
#include <stdio.h>
int* fun() {
   int x = 10;   // local variable (stack)
   return &x;    // returning address of local variable (dangling pointer!)
}
int main() {
   int *p = fun();
   printf("%d", *p);   // undefined behavior
   return 0;
}
```

Session-12

Topic: Recursive functions

1. **Recursive Function Without Base Case (Infinite Recursion Trap)**

```
#include <stdio.h>
void tricky(int n) {
   printf("%d ", n);
   if (n == 0)  // looks like a base case, but notice it's not stopping recursion
      tricky(n);
   else
      tricky(n - 1);
}
int main() {
   tricky(3);
   return 0;
}
```

****

2. **Indirect Recursion (Mutual Recursion)**

```
#include <stdio.h>
void funB(int n);
void funA(int n) {
   if (n <= 0) return;
   printf("%d ", n);
   funB(n - 1);
}
void funB(int n) {
   if (n <= 0) return;
   printf("%d ", n);
   funA(n / 2);
}
int main() {
   funA(20);
   return 0;
}
```

****

3. **Recursion with Static Variable (State Preserved)**

```
#include <stdio.h>
int fun(int n) {
   static int x = 0;
   if (n > 0) {
      x++;
      return fun(n - 1) + x;
   }
   return 0;
}
int main() {
   printf("%d", fun(5));
   return 0;
}
```

****

4. **Recursion That Prints in Forward and Reverse (Two Phases)**

```
#include <stdio.h>

void tricky(int n) {

   if (n == 0) return;

   printf("%d ", n);   // forward phase

   tricky(n - 1);

   printf("%d ", n);   // reverse phase

}

int main() {
   tricky(3);
   return 0;
}
```

5. Recursive Function That Calls Itself Twice (Exponential Growth)

```
#include <stdio.h>

void tricky(int n) {

   if (n == 0) return;

   printf("%d ", n);

   tricky(n - 1);

   tricky(n - 1);

}

int main() {

   tricky(3);

   return 0;

}
```

****

6. **Recursive Palindrome Checker**

```
#include <stdio.h>

#include <string.h>

int isPalindrome(char str[], int l, int r) {

   if (l >= r) return 1;

   if (str[l] != str[r]) return 0;

   return isPalindrome(str, l + 1, r - 1);
}
int main() {
   char str[] = "madam";
   if (isPalindrome(str, 0, strlen(str) - 1))
      printf("Palindrome");
   else
      printf("Not Palindrome");
   return 0;
}
```
****

7. **Tricky Recursive Sum (No Loops Allowed)**

```
#include <stdio.h>
int sum(int n) {
   return n && (n + sum(n - 1));
}
int main() {
   printf("%d", sum(5));
   return 0;
}
```

****END****

Session-12

Topic: Scope & longevity of Variables

Q1. Block Scope (Local Variables)

What is the output of the following program

```
#include <stdio.h>
int main() {
    int a = 10; // local to main
    {
        int b = 20; // scope limited to this block
        printf("Inside block: a = %d, b = %d\n", a, b);
    }
    // printf("%d", b);  // Error: b is not visible here
    return 0;
}
```

****

Q2. Function Scope (Local Variable recreated each call)

What is the output of the following program

```
#include <stdio.h>
void demo() {
    int x = 0; // local to this function, lifetime = each call
    x++;
    printf("%d\n", x);
}
int main() {
    demo();
    demo();
    demo();
    return 0;
}
```

END

Q3. Static Local Variable (Preserves Value Across Calls)

What is the output of the following program

```
#include <stdio.h>
void demo() {
   static int x = 0; // lifetime = program execution
   x++;
   printf("%d\n", x);
}
int main() {
   demo();
   demo();
   demo();
   return 0;
}
```

Q4. Global Variable

What is the output of the following program

```
#include <stdio.h>
int g = 5; // global scope, lifetime = entire program
void print() {
   printf("g = %d\n", g);
}
int main() {
   printf("In main: g = %d\n", g);
   print();
   g = 20;
   print();
   return 0;
}
```

****

Q5. Shadowing (Local hides Global)

What is the output of the following program

```
#include <stdio.h>
int x = 100; // global
int main() {
   int x = 10; // shadows global inside main
   {
      int x = 1; // shadows outer x inside this block
      printf("Block x = %d\n", x);
   }
   printf("Main x = %d\n", x);
    return 0;
}
```

****

Q6. Automatic vs Static Lifetime

What is the output of the following program

```
#include <stdio.h>
void autoVar() {
   int a = 0;   // automatic
   a++;
   printf("auto = %d\n", a);
}
void staticVar() {
   static int s = 0; // static lifetime
   s++;
   printf("static = %d\n", s);
}
int main() {
   for (int i = 0; i < 3; i++) {
      autoVar();
      staticVar();
```

```
    }
    return 0;
}
```

****

Q7.Register Storage Class (Hint to Compiler)

What is the output of the program

```
#include <stdio.h>
int main() {
    register int counter = 0; // stored in CPU register (if possible)
    for (counter = 0; counter < 3; counter++) {
        printf("counter = %d\n", counter);
    }
    // printf("%p", &counter);
    return 0;
}
```

***END**

Session-13

Topic: Recursive functions

1. **Uninitialized Array Access (Garbage Values)**

```
#include <stdio.h>
int main() {
   int arr[5];   // uninitialized
   for (int i = 0; i < 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

****

2. **Out-of-Bounds Access**

```
#include <stdio.h>
int main() {
   int arr[3] = {1, 2, 3};
   printf("%d ", arr[3]);   // out of bounds
   printf("%d ", arr[100]); // very dangerous
   return 0;
}
```

****

3. **Partial Initialization**

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20};
   for (int i = 0; i < 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

****

4. **Array Name as Pointer**

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   printf("%p\n", arr);      // base address
   printf("%p\n", &arr[0]);  // same as above
   printf("%d\n", *(arr+2)); // element at index 2 → 30
   return 0;
}
```

****

5. **Array Index Trick**

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   printf("%d\n", 2[arr]);   // same as arr[2]
   printf("%d\n", *(arr+3)); // 40
   return 0;
}
```

****

6. **Reverse Traversal**

```
#include <stdio.h>
int main() {
   int arr[] = {1, 2, 3, 4, 5};
   for (int i = 4; i >= 0; i--)
      printf("%d ", arr[i]);
   return 0;
}
```

****

7. **Array Sum using Recursion**

```
#include <stdio.h>
int sum(int arr[], int n) {
   if (n == 0) return 0;
   return arr[n-1] + sum(arr, n-1);
}
int main() {
   int arr[] = {1, 2, 3, 4, 5};
   printf("%d", sum(arr, 5));
   return 0;
}
```

****

8. **Array and Function (Pass by Reference)**

```
#include <stdio.h>
void modify(int arr[], int n) {
   for (int i = 0; i < n; i++)
      arr[i] *= 2;   // modifies original array
}
int main() {
   int arr[] = {1, 2, 3, 4, 5};
   modify(arr, 5);
   for (int i = 0; i < 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

****

9.Iterative Function for max

```
#include <stdio.h>
// Function to find maximum element
int findMax(int arr[], int n) {
   int max = arr[0];
```

```
    for (int i = 1; i < n; i++) {
        if (arr[i] > max)
            max = arr[i];
    }
    return max;
}
int main() {
    int arr[] = {10, 45, 2, 67, 23};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Maximum element = %d\n", findMax(arr, n));
    return 0;
}
```

****

10.Iterative Function for min (Simple and Efficient)

```
#include <stdio.h>
// Function to find minimum element
int findMin(int arr[], int n) {
    int min = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < min)
            min = arr[i];
    }
    return min;
}
int main() {
    int arr[] = {10, 45, 2, 67, -5, 23};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Minimum element = %d\n", findMin(arr, n));
    return 0;
}
```

****

11.Average of Array Elements (Iterative)

```
#include <stdio.h>
// Function to find average of array
double findAverage(int arr[], int n) {
   int sum = 0;
   for (int i = 0; i < n; i++) {
      sum += arr[i];
   }
   return (double)sum / n;  // typecasting to avoid integer division
}
int main() {
   int arr[] = {10, 20, 30, 40, 50};
   int n = sizeof(arr) / sizeof(arr[0]);

   printf("Average = %.2f\n", findAverage(arr, n));
   return 0;
}
```

END

Session-14

Topic: Bubble Sort

```
#include <stdio.h>
#include <stdbool.h>
// Bubble sort function
void bubbleSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      bool swapped = false;   // to detect if array is already sorted
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            // swap
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
            swapped = true;
         }
      }
      // If no two elements were swapped, break (optimization)
      if (!swapped)
         break;
   }
}
int main() {
   int n;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter %d elements:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
```

```
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Trace the bubble sort on arr = [5, 1, 4, 2, 8]

Number of passes	n-1=
Number of comparisons in pass#1	
Number of comparisons in pass#2	
Number of comparisons in pass#3	
Number of comparisons in pass#4	
--	
--	
Number of comparisons in pass#n-1	

- Derive a formula for the Number of passes
- Derive the Number of Comparisons

****

Session-15

Topic: Insertion sort

```
#include <stdio.h>
// Function for insertion sort
void insertionSort(int arr[], int n) {
   for (int i = 1; i < n; i++) {
      int key = arr[i];   // element to be inserted
      int j = i - 1;
      // Shift elements that are greater than key
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;  // insert key at correct position
   }
}
int main() {
   int arr[] = {5, 1, 4, 2, 8};
   int n = sizeof(arr) / sizeof(arr[0]);
   insertionSort(arr, n);
   printf("Sorted array: ");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

Trace the insertion sort on arr = [5, 1, 4, 2, 8]

Session-16

Topic: Selection sort

```
#include <stdio.h>
// Function for Selection Sort
void selectionSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      // Find index of minimum element in unsorted part
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex])
            minIndex = j;
      }
      // Swap only if needed
      if (minIndex != i) {
         int temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
      }
   }
}
int main() {
   int arr[] = {64, 25, 12, 22, 11};
   int n = sizeof(arr) / sizeof(arr[0]);
   selectionSort(arr, n);
   printf("Sorted array: ");
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

Trace the selection sort on arr = [5, 1, 4, 2, 8]

END

Session-17

Topic: Linear Search

```
#include <stdio.h>
// Function for linear search
int linearSearch(int arr[], int n, int key) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == key)
         return i;  // return index if found
   }
   return -1; // not found
}
int main() {
   int arr[] = {10, 25, 30, 45, 50};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key;
   printf("Enter element to search: ");
   scanf("%d", &key);
   int pos = linearSearch(arr, n, key);
   if (pos == -1)
      printf("Element %d not found.\n", key);
   else
      printf("Element %d found at position %d (index %d).\n", key, pos + 1, pos);
   return 0;
}
```

Find the number of comparisons for successful and unsuccessful search

****

Session-18

Topic: Binary Search

```
#include <stdio.h>
// Function for Binary Search
int binarySearch(int arr[], int n, int key) {
   int low = 0, high = n - 1;
   while (low <= high) {
      int mid = (low + high) / 2; // middle index
      if (arr[mid] == key)
         return mid; // found
      else if (arr[mid] < key)
         low = mid + 1; // search right half
      else
         high = mid - 1; // search left half
   }
   return -1; // not found
}
int main() {
   int arr[] = {10, 20, 30, 40, 50, 60};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key;
   printf("Enter element to search: ");
   scanf("%d", &key);
   int pos = binarySearch(arr, n, key);
   if (pos == -1)
      printf("Element %d not found.\n", key);
   else
      printf("Element %d found at position %d (index %d).\n", key, pos + 1, pos);
   return 0;
}
```

END

Session-19

Topic: Matrix Addition

```c
#include <stdio.h>
int main() {
   int m, n;
   printf("Enter rows and columns of matrix: ");
   scanf("%d %d", &m, &n);
   int A[m][n], B[m][n], C[m][n];
   // Input for first matrix
   printf("Enter elements of matrix A:\n");
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &A[i][j]);
      }
   }
   // Input for second matrix
   printf("Enter elements of matrix B:\n");
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &B[i][j]);
      }
   }
   // Matrix Addition
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[i][j] = A[i][j] + B[i][j];
      }
   }
   // Display Result
   printf("Resultant Matrix C (A + B):\n");
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         printf("%d ", C[i][j]);
      }
      printf("\n");
   }

   return 0;
}
```

Trace

```
Enter rows and columns of matrix: 2 3
Matrix A:
1 2 3
4 5 6
Matrix B:
7 8 9
1 2 3
```

END

Session-20

Topic: Matrix Multiplication

C Program for Matrix Multiplication

```c
#include <stdio.h>
int main() {
   int m, n, p;
   printf("Enter rows and columns of Matrix A: ");
   scanf("%d %d", &m, &n);
   printf("Enter columns of Matrix B: ");
   scanf("%d", &p);
   int A[m][n], B[n][p], C[m][p];
   // Input for Matrix A
   printf("Enter elements of Matrix A:\n");
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &A[i][j]);
      }
   }
   // Input for Matrix B
   printf("Enter elements of Matrix B:\n");
   for (int i = 0; i < n; i++) {
        for (int j = 0; j < p; j++) {
            scanf("%d", &B[i][j]);
        }
    }
    // Initialize Result Matrix with 0
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            C[i][j] = 0;
        }
    }
    // Matrix Multiplication
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    // Display Result
    printf("Resultant Matrix C (A × B):\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

****

Example Run (Tracing)

Matrices

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \qquad B = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

We compute $C = A \times B$, where

$$C[i][j] = \sum_{k=0}^{2} A[i][k] \cdot B[k][j].$$

Row 0 of C (i = 0)	
C[0][0]	
C[0][1]	
C[0][2]	
Row 1 of C (i = 1)	
C[1][0]	
C[1][1]	
C[1][2]	
Row 2 of C (i = 2)	
C[2][0]	
C[2][1]	
C[2][2]	

****

Session-21

Topic: Transpose of a Matrix

```
#include <stdio.h>
int main() {
    int m, n;
    printf("Enter rows and columns of matrix: ");
    scanf("%d %d", &m, &n);
    int A[m][n], T[n][m];
    // Input elements
    printf("Enter elements of Matrix A:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &A[i][j]);
        }
    }
    // Find transpose
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            T[j][i] = A[i][j];
        }
    }
    // Display transpose
    printf("Transpose of Matrix A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", T[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Tracing

```
Enter rows and columns of matrix: 2 3
Matrix A:
1 2 3
4 5 6
Transpose of Matrix A:
1 4
2 5
3 6
```

END