Karnatak University's
KARNATAK SCIENCE COLLEGE, DHARWAD
NAAC accredited College
DEPARTMENT OF PHYSICS

B.Sc. Third Semester ELECTRONCS Lab Manual

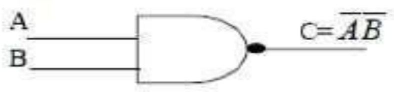
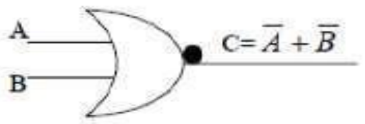
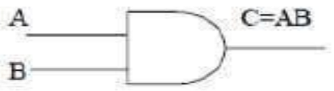
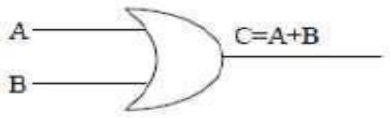
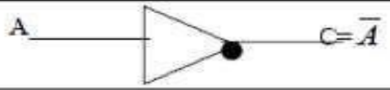
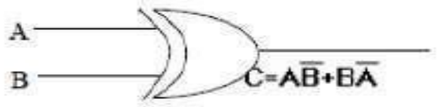
Digital Design using Verilog and Programming in C

Digital Design using Verilog and Programming in C

1. Realization of basic gates (OR, AND and NOT) using verilog code.
2. Simplify the given boolean expressions and realize using verilog programme.
3. Realize Adder/subtractor (Full/Half) circuits using verilog data flow description.
4. Realize the following code converters using verilog behavioral description.
a) Gray to Birary and Vice – Versa.
b) Binary to excess 3 and vice-versa.
5. To realize counters: Up/down (BCD & Binary) using verilog behavioral description.
6. To realize using verilog behavioral description flip flops:
a) JK - type (b) SR type (c) T-type (d) D-type.
7. To realize 4-bit ALU using verilog programme
8. C-Program to find i) area of a triangle ii) area of triangle when sides are given iii) area of a circle.
9. C-program using if-else statement i) to check whether given number is odd or even ii) to find whether a given integer is positive or negative.
10. C-program to find largest and smallest of given numbers.
11. C-program to find the roots of a quadratic equation.
12. C-program to illustrate switch statement.
13. C-program to find factorial of a number using while, do and for loops.
14. C-program to generate the Fibonacci series.
15. C-program to find sum of odd and even numbers using functions.
16. Write code to realize basic and sum & difference of two matrices using arrays.
17. C-program to find reverse of a number and to check whether it is a palindrome or no

1.Realization of basic gates (OR, AND and NOT) using verilog code.

Truth table with symbols

S.NO	GATE	SYMBOL	INPUTS		OUTPUT
			A	B	C
1.	NAND IC 7400		0	0	1
			0	1	1
			1	0	1
			1	1	0
2.	NOR IC 7402		0	0	1
			0	1	0
			1	0	0
			1	1	0
3.	AND IC 7408		0	0	0
			0	1	0
			1	0	0
			1	1	1
4.	OR IC 7432		0	0	0
			0	1	1
			1	0	1
			1	1	1
5.	NOT IC 7404		1	-	0
			0	-	1
6.	EX-OR IC 7486		0	0	0
			0	1	1
			1	0	1
			1	1	0

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module allgate_dataflow (a, b, yand, yor, ynot);
  input a, b;
  output yand, yor, ynot;
 	
  assign yand = a & b;   // AND gate
  assign yor = a | b;   // OR gate
  assign ynot = ~a;   // NOT gate
endmodule
	module allgate_structural (a, b, yand, yor, ynot);
	  input a, b;
	  output yand, yor, ynot;
	
	  and(yand, a, b); // AND gate using structural modeling
	  or(yor, a, b); // OR gate
	  not(ynot, a); // NOT gate
	endmodule
		module allgate_behavioral (a, b, yand, yor, ynot);
		  input a, b;
		  output reg yand, yor, ynot;
		
		  always @(*) begin
		        yand = a & b; // AND gate
		        yor = a | b; // OR gate
		        ynot = ~a; // NOT gate
		  end
		endmodule

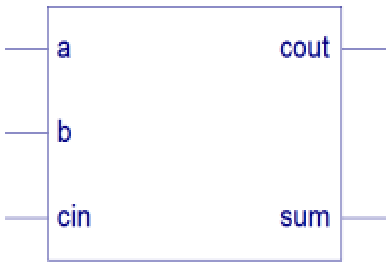
2. Simplify the given boolean expressions and realize using Verilog

Example: Boolean Expression: Y = A'B + AB'

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module boolean_expr_dataflow(output wire Y, input wire A, B);
  assign Y = (~A & B) | (A & ~B); // A'B + AB'
endmodule
	module boolean_expr_structural(output wire Y, input wire A, B);
	  wire not_A, not_B, term1, term2;
	
	  not u1 (not_A, A);
	  not u2 (not_B, B);
	  and u3 (term1, not_A, B);
	  and u4 (term2, A, not_B);
	  or u5 (Y, term1, term2);
	endmodule
		module boolean_expr_behavioral(output reg Y, input wire A, B);
		  always @(*) begin
		        Y = (~A & B) | (A & ~B); // A'B + AB'
		  end
		endmodule

3. Realize Adder/Subtractor (Full/Half) Circuits using Verilog



RTL Schematic

Half Adder

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module half_adder_dataflow(output wire sum, carry, input wire a, b);
  assign sum = a ^ b;   // Sum is XOR of a and b
  assign carry = a & b; // Carry is AND of a and b
endmodule
	module half_adder_structural(output wire sum, carry, input wire a, b);
	  wire w1, w2, w3;
	
	  // XOR gate for sum
	  xor u1(sum, a, b);
	
	  // AND gate for carry
	  and u2(carry, a, b);
	
	endmodule
		module half_adder_behavioral(output reg sum, carry, input wire a, b);
		  always @(*) begin
		        sum = a ^ b; // Sum is XOR of a and b
		        carry = a & b; // Carry is AND of a and b
		  end
		endmodule

Full Adder

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module full_adder_dataflow(output wire sum, carry, input wire a, b, cin);
  assign sum = a ^ b ^ cin;
        // Sum logic
  assign carry = (a & b) | (b & cin) | (a & cin); // Carry logic
endmodule
	module full_adder_structural(output wire sum, carry, input wire a, b, cin);
	  wire s1, c1, c2;
	
	  xor u1 (s1, a, b);
	  xor u2 (sum, s1, cin);
	
	  and u3 (c1, a, b);
	  and u4 (c2, s1, cin);
	  or u5 (carry, c1, c2);
	endmodule
		module full_adder_behavioral(output reg sum, carry, input wire a, b, cin);
		  always @(*) begin
		        sum = a ^ b ^ cin;
		        // Sum logic
		        carry = (a & b) | (b & cin) | (a & cin); // Carry logic
		  end
		endmodule

Half subtractor

Dataflow Model:	**Structural Model:**	**Behavioral Model:**
module half_subtractor_dataflow(output wire diff, borrow, input wire a, b); assign diff = a ^ b; // Difference is XOR of a and b assign borrow = ~a & b; // Borrow is AND of complement of a and b endmodule	module half_subtractor_structural(output wire diff, borrow, input wire a, b); wire w1, w2, w3; // XOR gate for difference xor u1(diff, a, b); // AND gate with NOT for borrow not u2(w1, a); and u3(borrow, w1, b); endmodule	module half_subtractor_behavioral(output reg diff, borrow, input wire a, b); always @(*) begin diff = a ^ b; // Difference is XOR of a and b borrow = ~a & b; // Borrow is AND of complement of a and b end endmodule

Full Subtractor:

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module full_subtractor_dataflow(output wire diff, bout, input wire a, b, bin);
  assign diff = a ^ b ^ bin;
        // Difference logic
  assign bout = (~a & b) | (b & bin) | (~a & bin); // Borrow logic
endmodule
	module full_subtractor_structural(output wire diff, bout, input wire a, b, bin);
	  wire d1, b1, b2;
	
	  xor u1 (d1, a, b);
	  xor u2 (diff, d1, bin);
	
	  and u3 (b1, ~a, b);
	  and u4 (b2, d1, bin);
	  or u5 (bout, b1, b2);
	endmodule
		module full_subtractor_behavioral(output reg diff, bout, input wire a, b, bin);
		  always @(*) begin
		        diff = a ^ b ^ bin;
		        // Difference logic
		        bout = (~a & b) | (b & bin) | (~a & bin); // Borrow logic
		  end
		endmodule

4. Code Converters using Verilog Behavioral Description

Gray to Binary

Dataflow Model:	**Structural Model:**	**Behavioral Model:**
module gray_to_binary_dataflow(output wire [3:0] binary, input [3:0] gray); assign binary[3] = gray[3]; assign binary[2] = binary[3] ^ gray[2]; assign binary[1] = binary[2] ^ gray[1]; assign binary[0] = binary[1] ^ gray[0]; endmodule	module gray_to_binary_structural(output wire [3:0] binary, input wire [3:0] gray); assign binary[3] = gray[3]; assign binary[2] = binary[3] ^ gray[2]; assign binary[1] = binary[2] ^ gray[1]; assign binary[0] = binary[1] ^ gray[0]; endmodule	module gray_to_binary_behavioral(output reg [3:0] binary, input [3:0] gray); always @(*) begin binary[3] = gray[3]; binary[2] = binary[3] ^ gray[2]; binary[1] = binary[2] ^ gray[1]; binary[0] = binary[1] ^ gray[0]; end endmodule

Binary to Gray

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module binary_to_gray_dataflow(output wire [3:0] gray, input wire [3:0] binary);
  assign gray[3] = binary[3];
  assign gray[2] = binary[3] ^ binary[2];
  assign gray[1] = binary[2] ^ binary[1];
  assign gray[0] = binary[1] ^ binary[0];
endmodule
	module binary_to_gray_structural(output wire [3:0] gray, input wire [3:0] binary);
	  assign gray[3] = binary[3];
	  assign gray[2] = binary[3] ^ binary[2];
	  assign gray[1] = binary[2] ^ binary[1];
	  assign gray[0] = binary[1] ^ binary[0];
	endmodule
		module binary_to_gray_behavioral(output reg [3:0] gray, input [3:0] binary);
		  always @(*) begin
		        gray[3] = binary[3];
		        gray[2] = binary[3] ^ binary[2];
		        gray[1] = binary[2] ^ binary[1];
		        gray[0] = binary[1] ^ binary[0];
		  end
		endmodule

Binary to Excess-3 Converter

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module binary_to_excess3_dataflow(output wire [3:0] excess3, input [3:0] binary);
  assign excess3 = binary + 4'b0011; // Binary to Excess-3 (add 3)
endmodule
	module binary_to_excess3_structural(output wire [3:0] excess3, input [3:0] binary);
	  wire [3:0] temp;
	
	  // Binary to Excess-3 logic: Add binary + 3
	  // Carry out calculation
	  assign temp[0] = binary[0];
	  assign temp[1] = binary[1];
	  assign temp[2] = binary[2];
	  assign temp[3] = binary[3];
	
	  // Excess-3 is simply Binary + 3
	  assign excess3 = temp + 4'b0011; // Add 3 (0011) to the binary number
	
	endmodule
		module binary_to_excess3_behavioral(output reg [3:0] excess3, input [3:0] binary);
		  always @(*) begin
		        excess3 = binary + 4'b0011; // Add 3 to convert binary to excess-3
		  end
		endmodule

Excess-3 to Binary Converter

Dataflow Model:	**Structural Model:**	**Behavioral Model:**
module excess3_to_binary_dataflow(output wire [3:0] binary, input [3:0] excess3); assign binary = excess3 - 4'b0011; // Excess-3 to binary (subtract 3) endmodule	module excess3_to_binary_structural(output wire [3:0] binary, input [3:0] excess3); wire [3:0] temp; // Excess-3 to binary logic: Subtract 3 // Carry out calculation assign temp[0] = excess3[0]; assign temp[1] = excess3[1]; assign temp[2] = excess3[2]; assign temp[3] = excess3[3]; // Binary is Excess-3 minus 3 assign binary = temp - 4'b0011; // Subtract 3 (0011) from excess-3 number endmodule	module excess3_to_binary_behavioral(output reg [3:0] binary, input [3:0] excess3); always @(*) begin binary = excess3 - 4'b0011; // Subtract 3 to convert excess-3 back to binary end endmodule

5. Counters: Up/Down (BCD & Binary) using Verilog Behavioral Description

BCD Counter:

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module
bcd_counter_dataflow(output
wire [3:0] count, input clk,
reset);
  reg [3:0] temp_count;
 	
  always @(posedge clk or
posedge reset) begin
    if (reset)
    temp_count <= 4'b0000;
    else if (temp_count ==
4'b1001)
    temp_count <= 4'b0000;
    else
    temp_count <=
temp_count + 1;
  end
 	
  assign count = temp_count;
endmodule
	odule
	bcd_counter_structural(output
	wire [3:0] count, input clk,
	reset);
	  reg [3:0] temp_count;
	
	  always @(posedge clk or
	posedge reset) begin
	    if (reset)
	    temp_count <= 4'b0000;
	    else if (temp_count ==
	4'b1001)
	    temp_count <= 4'b0000;
	    else
	    temp_count <=
	temp_count + 1;
	  end
	
	  assign count = temp_count;
	endmodule
	m
		module
		bcd_counter_behavioral(output
		reg [3:0] count, input clk,
		reset);
		  always @(posedge clk or
		posedge reset) begin
		    if (reset)
		    count <= 4'b0000;
		    else if (count ==
		4'b1001)
		    count <= 4'b0000;
		    else
		    count <= count + 1;
		  end
		endmodule

Binary Counter:

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module binary_counter_dataflow(output wire [3:0] count, input clk, reset);
  reg [3:0] temp_count;
 	
  always @(posedge clk or posedge reset) begin
    if (reset)
    temp_count <= 4'b0000;
    else
    temp_count <= temp_count + 1;
  end
 	
  assign count = temp_count;
endmodule
	module binary_counter_structural(output wire [3:0] count, input clk, reset);
	  reg [3:0] temp_count;
	
	  always @(posedge clk or posedge reset) begin
	    if (reset)
	    temp_count <= 4'b0000;
	    else
	    temp_count <= temp_count + 1;
	  end
	
	  assign count = temp_count;
	endmodule
		module binary_counter_behavioral(output reg [3:0] count, input clk, reset);
		  always @(posedge clk or posedge reset) begin
		    if (reset)
		    count <= 4'b0000;
		    else
		    count <= count + 1;
		  end
		endmodule

6. Flip-Flops using Verilog Behavioral Description

JK Flip flop

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module jk_flipflop_dataflow(output wire Q, input J, K, clk, reset);
  reg Q_reg;
 	
  always @(posedge clk or posedge reset) begin
    if (reset)
    Q_reg <= 0;
    else if (J & ~K)
    Q_reg <= 1;
    else if (~J & K)
    Q_reg <= 0;
    else if (J & K)
    Q_reg <= ~Q_reg;
  end
 	
  assign Q = Q_reg;
endmodule
	module jk_flipflop_structural(output wire Q, input J, K, clk, reset);
	  reg Q_reg;
	
	  always @(posedge clk or posedge reset) begin
	    if (reset)
	    Q_reg <= 0;
	    else if (J & ~K)
	    Q_reg <= 1;
	    else if (~J & K)
	    Q_reg <= 0;
	    else if (J & K)
	    Q_reg <= ~Q_reg;
	  end
	
	  assign Q = Q_reg;
	endmodule
		module jk_flipflop_behavioral(output reg Q, input J, K, clk, reset);
		  always @(posedge clk or posedge reset) begin
		    if (reset)
		    Q <= 0;
		    else if (J & ~K)
		    Q <= 1;
		    else if (~J & K)
		    Q <= 0;
		    else if (J & K)
		    Q <= ~Q;
		  end
		endmodule

SR Flip-Flop

Dataflow Model:	Structural Model:	Behavioral Model:

module sr_flipflop_dataflow(output wire Q, output wire Qn, input wire S, R, clk, reset);
  assign Q = (S & ~R) ? 1 : ((~S & R) ? 0 : Q);
  assign Qn = ~Q;
endmodule
	module sr_flipflop_structural(output wire Q, output wire Qn, input wire S, R, clk, reset);
	  wire nS, nR;
	
	  not u1(nS, S);
	  not u2(nR, R);
	  nand u3(Q, clk, nS, Qn);
	  nand u4(Qn, clk, nR, Q);
	
	endmodule
		module sr_flipflop_behavioral(output reg Q, output reg Qn, input wire S, R, clk, reset);
		  always @(posedge clk or posedge reset) begin
		    if (reset) begin
		    Q <= 0;
		    Qn <= 1;
		    end
		    else if (S == 1 && R == 0) begin
		    Q <= 1;
		    Qn <= 0;
		    end
		    else if (S == 0 && R == 1) begin
		    Q <= 0;
		    Qn <= 1;
		    end
		    // S = R = 0, hold state, S = R = 1 is invalid (Q and Qn undefined)
		  end
		endmodule

T Flip-Flop

Dataflow Model:	**Structural Model:**	**Behavioral Model:**

module t_flipflop_dataflow(output wire Q, input wire T, clk, reset);
  reg state;
 	
  always @(posedge clk or posedge reset) begin
        if (reset)
        state <= 0;
        else
        state <= state ^ T; // Toggle on T=1, hold on T=0
  end
 	
  assign Q = state;
endmodule
	module t_flipflop_structural(output wire Q, input wire T, clk, reset);
	  wire d;
	
	  xor u1(d, Q, T); // T flip-flop is a D flip-flop with D = Q ^ T
	  d_flipflop dff(Q, d, clk, reset);
	
	endmodule
	
	module d_flipflop(output reg Q, input wire D, clk, reset);
	  always @(posedge clk or posedge reset) begin
	        if (reset)
	        Q <= 0;
	        else
	        Q <= D;
	  end
	endmodule
		module t_flipflop_behavioral(output reg Q, input wire T, clk, reset);
		  always @(posedge clk or posedge reset) begin
		        if (reset)
		        Q <= 0;
		        else if (T)
		        Q <= ~Q;
		        else
		        Q <= Q; // Hold state
		  end
		endmodule

D Flip-Flop

Dataflow Model:	**Structural Model:**	**Behavioral Model:**
module	module	module
d_flipflop_dataflow(output	d_flipflop_structural(output	d_flipflop_behavioral(output
wire Q, input wire D, clk,	wire Q, input wire D, clk,	reg Q, input wire D, clk, reset);
reset);	reset);	always @(posedge clk or
reg state;	wire nD;	posedge reset) begin
	not u1(nD, D);	if (reset)
always @(posedge clk or		Q <= 0;
posedge reset) begin	nand u2(Q, clk, D);	else
if (reset)	nand u3(Qn, clk, nD);	Q <= D; // Capture D on
state <= 0;	endmodule	clock edge
else		end
state <= D;		endmodule
end		
assign Q = state;		
endmodule		

7. 4-bit ALU using Verilog

Dataflow Model:	Structural Model:	Behavioral Model:

module
alu_4bit_dataflow(output wire
[3:0] result, output wire
carry_out, input [3:0] a, b,
input [1:0] opcode);
  assign {carry_out, result} =
(opcode == 2'b00) ? (a + b) :
                              (opcode ==
2'b01) ? (a - b) :
                              (opcode ==
2'b10) ? (a & b) :
 	
          (a | b); // ALU logic
based on opcode
endmodule
	module
	alu_4bit_structural(output wire
	[3:0] result, output wire
	carry_out, input [3:0] a, b,
	input [1:0] opcode);
	  reg [3:0] res;
	  reg c_out;
	
	  always @(*) begin
	          case (opcode)
	          2'b00: {c_out, res} = a +
	b; // Addition
	          2'b01: {c_out, res} = a -
	b; // Subtraction
	          2'b10: res = a & b;
	          // AND operation
	          2'b11: res = a | b;
	          // OR operation
	          default: res = 4'b0000;
	          endcase
	  end
	
	  assign result = res;
	  assign carry_out = c_out;
	endmodule
		module
		alu_4bit_behavioral(output reg
		[3:0] result, output reg
		carry_out, input [3:0] a, b,
		input [1:0] opcode);
		  always @(*) begin
		          case (opcode)
		          2'b00: {carry_out,
		result} = a + b; // Addition
		          2'b01: {carry_out,
		result} = a - b; // Subtraction
		          2'b10: result = a & b;
		          // AND operation
		          2'b11: result = a | b;
		          // OR operation
		          default: result = 4'b0000;
		          endcase
		  end
		endmodule

8. C-Program to find i) area of a triangle ii) area of triangle when sides are given iii) area of a circle.

i) Area of a triangle

```
#include <stdio.h>

int main() {
	float base, height, area;
	printf("Enter base and height of the triangle: ");
	scanf("%f %f", &base, &height);
	area = 0.5 * base * height;
	printf("Area of the triangle: %.2f\n", area);
	return 0;
}
```

ii) Area of a triangle when sides are given (Heron's Formula)

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

int main() {
	float a, b, c, s, area;
	printf("Enter the three sides of the triangle: ");
	scanf("%f %f %f", &a, &b, &c);
	s = (a + b + c) / 2; // Semi-perimeter
	area = sqrt(s * (s - a) * (s - b) * (s - c));
	printf("Area of the triangle: %.2f\n", area);
	return 0;
}
```

iii) Area of a circle

```
#include <stdio.h>
#define PI 3.14159
```

```c
int main() {
	float radius, area;
	printf("Enter the radius of the circle: ");
	scanf("%f", &radius);
	area = PI * radius * radius;
	printf("Area of the circle: %.2f\n", area);
	return 0;
}
```

9. C-Program using if-else statement:

i) To check whether a given number is odd or even

```
#include <stdio.h>

int main() {
        int num;
        printf("Enter an integer: ");
        scanf("%d", &num);

        if (num % 2 == 0)
        printf("%d is even.\n", num);
        else
        printf("%d is odd.\n", num);

        return 0;
}
```

ii) To check whether a given integer is positive or negative

```
#include <stdio.h>

int main() {
        int num;
        printf("Enter an integer: ");
        scanf("%d", &num);

        if (num > 0)
        printf("%d is positive.\n", num);
        else if (num < 0)
        printf("%d is negative.\n", num);
        else
        printf("The number is zero.\n");

        return 0;
```

10. C-Program to find largest and smallest of given numbers

```
#include <stdio.h>

int main() {
        int n, num, largest, smallest;
        printf("Enter the number of elements: ");
        scanf("%d", &n);

        printf("Enter %d integers: ", n);
        scanf("%d", &num);
        largest = smallest = num;

        for (int i = 1; i < n; i++) {
        scanf("%d", &num);
        if (num > largest)
        largest = num;
        if (num < smallest)
        smallest = num;
        }

        printf("Largest: %d\n", largest);
        printf("Smallest: %d\n", smallest);

        return 0;
}
```

11. C-Program to find the roots of a quadratic equation

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

int main() {
	float a, b, c, discriminant, root1, root2, realPart, imagPart;
	printf("Enter coefficients a, b, and c: ");
	scanf("%f %f %f", &a, &b, &c);

	discriminant = b * b - 4 * a * c;

	if (discriminant > 0) {
	root1 = (-b + sqrt(discriminant)) / (2 * a);
	root2 = (-b - sqrt(discriminant)) / (2 * a);
	printf("Roots are real and different.\n");
	printf("Root 1 = %.2f\nRoot 2 = %.2f\n", root1, root2);
	} else if (discriminant == 0) {
	root1 = -b / (2 * a);
	printf("Roots are real and the same.\n");

	printf("Root = %.2f\n", root1);
	} else {
	realPart = -b / (2 * a);
	imagPart = sqrt(-discriminant) / (2 * a);
	printf("Roots are complex and different.\n");
	printf("Root 1 = %.2f + %.2fi\nRoot 2 = %.2f - %.2fi\n", realPart, imagPart, realPart,
imagPart);
	}

	return 0;
}
```

12. C-Program to illustrate switch statement

```
#include <stdio.h>

int main() {
	int choice;
	printf("Menu:\n1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n");
	printf("Enter your choice: ");
	scanf("%d", &choice);

	int a, b;
	printf("Enter two integers: ");
	scanf("%d %d", &a, &b);

	switch (choice) {
	case 1:
	printf("Result: %d\n", a + b);
	break;
	case 2:
	printf("Result: %d\n", a - b);
	break;
	case 3:
	printf("Result: %d\n", a * b);
	break;
	case 4:
	if (b != 0)
		printf("Result: %.2f\n", (float)a / b);
	else
		printf("Division by zero is not allowed.\n");
	break;
	default:
	printf("Invalid choice.\n");
	}

	return 0;
}
```

13. C-Program to find the factorial of a number using while, do, and for loops

i) Using a while loop

```
#include <stdio.h>

int main() {
	int num, factorial = 1, i = 1;
	printf("Enter a number: ");
	scanf("%d", &num);

	while (i <= num) {
	factorial *= i;
	i++;
	}

	printf("Factorial of %d is %d\n", num, factorial);
	return 0;
}
```

ii) Using a do-while loop

```
#include <stdio.h>

int main() {
	int num, factorial = 1, i = 1;
	printf("Enter a number: ");
	scanf("%d", &num);
```

```
    do {
    factorial *= i;
    i++;
    } while (i <= num);

    printf("Factorial of %d is %d\n", num, factorial);
    return 0;
}
```

iii) Using a for loop

```
#include <stdio.h>

int main() {
    int num, factorial = 1;
    printf("Enter a number: ");
    scanf("%d", &num);

    for (int i = 1; i <= num; i++) {
    factorial *= i;
    }

    printf("Factorial of %d is %d\n", num, factorial);
    return 0;
}
```

14. C-Program to generate the Fibonacci series

```
#include <stdio.h>


int main() {
        int n, t1 = 0, t2 = 1, nextTerm;
        printf("Enter the number of terms: ");
        scanf("%d", &n);


        printf("Fibonacci Series: %d, %d", t1, t2);


        for (int i = 3; i <= n; i++) {
        nextTerm = t1 + t2;
        printf(", %d", nextTerm);
        t1 = t2;
        t2 = nextTerm;
        }
        printf("\n");
        return 0;
}
```

15. C-Program to find the sum of odd and even numbers using functions

```
#include <stdio.h>

void sumOddEven(int n, int *oddSum, int *evenSum) {
        *oddSum = 0;
        *evenSum = 0;

        for (int i = 1; i <= n; i++) {
        if (i % 2 == 0)
        *evenSum += i;
        else
        *oddSum += i;
        }
}

int main() {
        int n, oddSum, evenSum;
        printf("Enter a number: ");
        scanf("%d", &n);

        sumOddEven(n, &oddSum, &evenSum);
        printf("Sum of odd numbers: %d\n", oddSum);
        printf("Sum of even numbers: %d\n", evenSum);
        return 0;
}
```

16. C-Program to realize basic sum & difference of two matrices using arrays

```
#include <stdio.h>

#define MAX 10

void printMatrix(int matrix[MAX][MAX], int row, int col) {
        for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
        printf("%d ", matrix[i][j]);
        }
        printf("\n");
        }
}

int main() {
        int a[MAX][MAX], b[MAX][MAX], sum[MAX][MAX], difference[MAX][MAX];
        int row, col;

        printf("Enter the number of rows and columns: ");
        scanf("%d %d", &row, &col);

        printf("Enter elements of the first matrix:\n");
        for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
        scanf("%d", &a[i][j]);
        }
        }

        printf("Enter elements of the second matrix:\n");
        for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
        scanf("%d", &b[i][j]);
        }
        }
```

```
    // Calculate sum and difference
    for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
    sum[i][j] = a[i][j] + b[i][j];
    difference[i][j] = a[i][j] - b[i][j];
    }
    }

    printf("Sum of matrices:\n");
    printMatrix(sum, row, col);

    printf("Difference of matrices:\n");
    printMatrix(difference, row, col);

    return 0;
}
```

17. C-Program to find the reverse of a number and check whether it is a palindrome or not

```
#include <stdio.h>

int main() {
        int num, reversedNum = 0, remainder, originalNum;

        printf("Enter an integer: ");
        scanf("%d", &num);

        originalNum = num; // Store original number

        // Reverse the number
        while (num != 0) {
        remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
        }

        printf("Reversed Number: %d\n", reversedNum);

        // Check if the original number and reversed number are the same
        if (originalNum == reversedNum)
        printf("%d is a palindrome.\n", originalNum);
        else
        printf("%d is not a palindrome.\n", originalNum);

        return 0;
}
```