1. Write a program to perform number conversions.

AIM: To demonstrate a Java program to perform number conversions.

DESCRIPTION: This program performs number system conversions using a menu-driven approach.

The user can convert numbers between:

- Decimal ↔ Binary
- Decimal ↔ Octal
- Binary ↔ Hexadecimal
- Hexadecimal → Binary

The program repeatedly displays a menu, takes the user's choice, performs the selected conversion using separate functions, and displays the result. Dynamic memory allocation is used where required, and the program terminates when the user selects the exit option.

ALGORITHM:

Step 1: Start the program.

Step 2: Display the menu with conversion options:

1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Hexadecimal to Binary
6. Binary to Hexadecimal
7. Exit

Step 3: Read the user's choice.

Step 4: If the choice is 7, display "Goodbye" and stop the program.

Step 5: Otherwise, perform the selected conversion:

- Decimal to Binary
  - Repeatedly divide the decimal number by 2
  - Store remainders
  - Reverse the result to get binary number

- Binary to Decimal
    - Multiply each bit by powers of 2
    - Add the values to get decimal
- Decimal to Octal
    - Convert decimal to octal using base-8 conversion
- Octal to Decimal
    - Multiply each digit by powers of 8
    - Add the values
- Hexadecimal to Binary
    - Convert hexadecimal to decimal
    - Convert decimal to binary
- Binary to Hexadecimal
    - Group binary digits into sets of four
    - Convert each group to its hexadecimal equivalent

Step 6: Display the converted result.

Step 7: Repeat Steps 2 to 6 until the user selects Exit.

Step 8: End the program

2. **Write a Java program to perform arithmetic operations.**

AIM: To demonstrate a Java program to perform arithmetic operations.

DESCRIPTION:

This program performs **binary arithmetic operations** such as **Addition, Subtraction, Multiplication, and Division** without using arithmetic operators (+, -, *, /).

Instead, it uses **bitwise operators** like AND (&), XOR (^), NOT (~), and Left Shift (<<).

The program is **menu-driven**, allowing the user to select an operation and enter two integers. The selected binary operation is performed and the result is displayed.

ALGORITM:

Step 1: Start the program

Step 2: Display menu with options

1. Binary Addition
2. Binary Subtraction
3. Binary Multiplication
4. Binary Division
5. Exit

Step 3: Read the user's choice

Step 4: If choice is not Exit

- Read two integers n1 and n2

Step 5: Perform operation based on choice

Case 1: Binary Addition

1. Repeat until n2 becomes 0
2. Compute carry as (n1 & n2) << 1
3. Compute sum as n1 ^ n2
4. Assign sum to n1 and carry to n2
5. Display result

Case 2: Binary Subtraction

1. Find 2's complement of n2
2. Add it to n1 using binary addition
3. Display result

Case 3: Binary Multiplication

1. Initialize result as 0
2. Add n1 to result repeatedly n2 times
3. Display result

Case 4: Binary Division

1. If n1 < n2, result is 0
2. Subtract n2 from n1 repeatedly using binary addition
3. Count number of subtractions
4. Display quotient

Step 6: Repeat steps until user selects Exit

Step 7: Stop the program

3. Implement an absolute loader.

AIM: To implement an absolute loader in C.

DESCRIPTION:

This program simulates a simple loader that reads an object program from an input file (input.dat) and produces a memory map in an output file (output.dat).

The object program contains:

- H (Header) record → starting address and program length
- T (Text) record → address and object code
- E (End) record → end of program

The program reads object code byte by byte, assigns consecutive memory addresses, and writes them into the output file until the End (E) record is encountered.

ALGORITHM:

Step 1: Start the program

Step 2: Declare variables and file pointers

- input → stores records read from file
- start, length → header information
- address → memory address
- fp1, fp2 → input and output file pointers

Step 3: Open files

- Open input.dat in read mode
- Open output.dat in write mode

Step 4: Read the first record from input file

Step 5: Repeat until End record (E) is found

a) If record is Header (H)

1. Read starting address
2. Read program length
3. Read next input record

b) If record is Text (T)

1. Read starting address of object code
2. Read object code
3. Split object code into bytes
4. Write each byte with its memory address to output file
5. Increment address
6. Read next input record

c) Otherwise (remaining object code)

1. Write object code bytes to output file
2. Assign sequential addresses
3. Read next input record

Step 6: Close input and output files

Step 7: Display “FINISHED”

Step 8: Stop the program.

4. **Implement a relocating loader.**

AIM: To implement a relocating loader in C.

DESCRIPTION:

This program implements a Relocating Loader in C.
It reads a relocatable object program from an input file (relinput.dat) and produces a relocated memory map in an output file (reloutput.dat).
The program uses a bitmask to determine whether an address field in the object code needs relocation or not.

If the relocation bit is 1, the actual starting address is added to the address field.

If the relocation bit is 0, the address remains unchanged.
The user provides the actual starting address at runtime.

ALGORITHM:

Step 1: Start the program

Step 2: Declare variables and file pointers

- add, length, input → store records
- bitmask → relocation bits
- start → actual starting address
- address, opcode, addr, actualadd → processing variables

Step 3: Read the actual starting address from the user

Step 4: Open files

- Open relinput.dat in read mode
- Open reloutput.dat in write mode

Step 5: Read the first record from input file

Step 6: Repeat until End record (E) is encountered

a) If record is Header (H)

1. Read program name/address
2. Read program length
3. Read next recordb) If record is Text (T)

1. Read starting address of text record
2. Read relocation bitmask
3. Add actual starting address to the text record address
4. Find length of bitmask
5. For each bit in the bitmask:
    - Read opcode and address field
    - If relocation bit = 0
        - Actual address = address field
    - If relocation bit = 1
        - Actual address = address field + starting address
    - Write relocated address and object code to output file
    - Increment address by instruction length
6. Read next input record

Step 7: Close input and output files

Step 8: Display “FINISHED”

Step 9: Stop the program.

5. **Write a program to perform register operations.**

AIM: To demonstrate a Java program to perform register operation.

DESCRIPTION:

This program simulates basic micro-operations performed on 8-bit CPU registers using the Java programming language.
Two registers, R1 and R2, are represented using the uint8_t data type to ensure 8-bit behaviour.

The program demonstrates common register-level micro-operations such as:

- Register Transfer
- Addition
- Bitwise AND
- Bitwise OR
- Left Shift
- Right Shift

A menu-driven interface allows the user to select an operation. After each operation, the contents of the registers are displayed in hexadecimal format.

ALGORITHM:

Step 1: Start the program

Step 2: Initialize registers

- Set R1 = 0x0F
- Set R2 = 0xF0

Step 3: Display menu repeatedly until Exit is selected

Step 4: Read user choice

Step 5: Perform selected micro-operation

Case 1: Register Transfer

1. Copy contents of R2 into R1
2. Display register values

Case 2: Addition

1. Add contents of R1 and R2
2. Store result in R1
3. Display register values

Case 3: Bitwise AND

1. Perform AND operation between R1 and R2
2. Store result in R1
3. Display register values

Case 4: Bitwise OR

1. Perform OR operation between R1 and R2
2. Store result in R1
3. Display register values

Case 5: Left Shift

1. Shift bits of R1 left by one position
2. Store result in R1
3. Display register values

Case 6: Right Shift

1. Shift bits of R1 right by one position
2. Store result in R1
3. Display register values

Case 7: Reset Registers

1. Reset R1 and R2 to initial values
2. Display register values

Case 8: Exit

1. Terminate the program

Step 6: Stop the program.

6. **Program to print ASCII for characters.**

AIM: To demonstrate ASCII for characters in Java.

DESCRIPTION:

This program displays the ASCII values of characters from 0 to 127.
ASCII (American Standard Code for Information Interchange) assigns a unique numeric value to each character.

The program uses a loop to iterate through all ASCII values and checks whether a character is printable or non-printable.
Printable characters (ASCII 32 to 126) are displayed as characters, while non-printable characters are labelled accordingly.

ALGORITHM:

Step 1: Start the program

Step 2: Declare an integer variable i

Step 3: Display headings for ASCII table

Step 4: Use a for loop from i = 0 to i = 127

Step 5: For each value of i

1. Check if i is between 32 and 126
    - If yes, print the character and its ASCII value
    - If no, print “Non-printable” and the ASCII value

Step 6: Continue until all ASCII values are printed

Step 7: Stop the program.

7. Program to implement logic gates.

AIM: To Implement logic gates using Java.

DESCRIPTION:

This program implements basic digital logic gates using the C programming language. The logic gates included are AND, OR, NOT, XOR, NAND, NOR, and XNOR.

The program accepts binary inputs (0 or 1) from the user, performs the selected logic gate operation using bitwise and logical operators, and displays the result.
A menu-driven approach is used to allow the user to select the required logic gate.

ALGORITHM:

Step 1: Start the program

Step 2: Display the list of logic gates

1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. XNOR

Step 3: Read the user's choice

Step 4: Read input values

- If the selected gate is NOT, read only one input
- For all other gates, read two inputs

Step 5: Validate inputs

- Check whether the inputs are either 0 or 1
- If invalid, display an error message and terminate

Step 6: Perform the selected logic operation using switch-case

Case 1: AND Gate

- Output = a & b

Case 2: OR Gate

- Output = a | b

Case 3: NOT Gate

- Output = !a

Case 4: XOR Gate

- Output = a ^ b

Case 5: NAND Gate

- Output = !(a & b)

Case 6: NOR Gate

- Output = !(a | b)

Case 7: XNOR Gate

- Output = !(a ^ b)

Step 7: Display the result

Step 8: Stop the program.

8. **Java Program to implement Half Adder and Full Adder.**

AIM: To implement Half adder and Full adder using Java.

DESCRIPTION:

This program implements binary addition circuits: Half Adder and Full Adder using the C programming language.
The program accepts binary inputs (0 or 1) from the user and calculates the Sum and Carry using bitwise operators.

- Half Adder adds two single-bit binary numbers.
- Full Adder adds three single-bit binary numbers (including carry-in).

A menu-driven approach allows the user to choose between Half Adder and Full Adder.

ALGORITHM:

Step 1: Start the program

Step 2: Display menu

1. Half Adder
2. Full Adder

Step 3: Read the user's choice

Step 4: Read input values

- If Half Adder is selected:
  - Read two inputs a and b
- If Full Adder is selected:
  - Read three inputs a, b, and cin

Step 5: Validate inputs

- Check whether all inputs are either 0 or 1
- If invalid, display error message and stop

Step 6: Perform selected operation

Half Adder

1. Compute Sum using a XOR b
2. Compute Carry using a AND b
3. Display Sum and Carry

Full Adder

1. Compute Sum using a XOR b XOR cin
2. Compute Carry using
   (a AND b) OR (b AND cin) OR (a AND cin)
3. Display Sum and Carry

Step 7: Stop the program.