1. Inventory Update System
    - Input: An array of integers representing inventory levels and an array of changes in stock.
    - Process: Pass the arrays to a function by reference to update inventory levels.
    - Output: Print the updated inventory levels and flag items below the restocking threshold.
    - Concepts: Arrays, functions, pass by reference, decision-making (if-else).

```
#include <stdio.h>
#define SIZE 5
#define THRESHOLD 10

// Function prototype
void updateInventory(int *inventory, int *changes);

int main()
{
   // Input the inventory levels and stock changes
   int inventory[SIZE] = {18, 8, 20, 15, 13};
   int changes[SIZE] = {-6, 4, 1, -2, -5};

   printf("Initial inventory levels\n");
   for (int i = 0; i < SIZE; i++)
      printf("Item %d: %d\n", i + 1, inventory[i]);
```

```
    // Call the function to update the inventory levels by passing arrays
by reference
    updateInventory(&inventory[0], &changes[0]);

    printf("\nUpdated inventory levels\n");
    for (int i = 0; i < SIZE; i++)
    {
        printf("Item %d: %d", i + 1, inventory[i]);
        if (inventory[i] < THRESHOLD)
            printf(" - Below restocking threshold\n");
        printf("\n");
    }
    return 0;
}

/*
Name: updateInventory()
Return Type: void
Parameter:(data type of each parameter): int* and int*
Short description: it is used to update the inventory levels
*/

// Function to update inventory levels
void updateInventory(int *inventory, int *changes)
{
    for (int i = 0; i < SIZE; i++)
        inventory[i] += changes[i];
}
```

O/P:

Initial inventory levels

Item 1: 18

Item 2: 8

Item 3: 20

Item 4: 15

Item 5: 13

Updated inventory levels

Item 1: 12

Item 2: 12

Item 3: 21

Item 4: 13

Item 5: 8 - Below restocking threshold

2. Product Price Adjustment
    - Input: An array of demand levels (constant) and an array of product prices.
    - Process: Use a function to calculate new prices based on demand levels. The function should return a pointer to an array of adjusted prices.
    - Output: Display the original and adjusted prices.
    - Concepts: Passing constant data, functions, pointers, arrays.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
```

```
#define DEMAND_THRESHOLD 50
#define PRICE_INCREASE 1.2  // Increase price by 20%
#define PRICE_DECREASE 0.8  // Decrease price by 20%

// Function prototype
float* adjustPrices(const int *demand, float *prices);

int main()
{
   // Input the demand levels and product prices
   const int demand[SIZE] = {20, 40, 60, 80, 50};
   float prices[SIZE] = {100.0, 150.0, 200.0, 120.0, 180.0};

   printf("Original prices\n");
   for (int i = 0; i < SIZE; i++)
      printf("Product %d: %.2f\n", i + 1, prices[i]);

   // Call the function to calculate adjusted prices
   float *adjustedPrices = adjustPrices(demand, prices);

   printf("\nAdjusted prices\n");
   for (int i = 0; i < SIZE; i++)
      printf("Product %d: %.2f\n", i + 1, adjustedPrices[i]);

   free(adjustedPrices);
   return 0;
}
```

```
/*
Name: adjustPrices()
Return Type: float*
Parameter:(data type of each parameter): const int* and float*
Short description: it is used to adjust prices based on demand levels
*/

// Function to adjust prices based on demand levels
float* adjustPrices(const int *demand, float *prices)
{
    // Allocate memory for the adjusted prices
    float *adjustedPrices = (float*) malloc(SIZE * sizeof(float));
    if (adjustedPrices == NULL)
    {
        printf("Memory allocation failed\n");
        exit(1);
    }

    // Calculate adjusted prices
    for (int i = 0; i < SIZE; i++)
    {
        if (demand[i] > DEMAND_THRESHOLD)
            adjustedPrices[i] = prices[i] * PRICE_INCREASE;
        else
            adjustedPrices[i] = prices[i] * PRICE_DECREASE;
    }
    return adjustedPrices;
```

}

O/P:

Original prices

Product 1: 100.00

Product 2: 150.00

Product 3: 200.00

Product 4: 120.00

Product 5: 180.00

Adjusted prices

Product 1: 80.00

Product 2: 120.00

Product 3: 240.00

Product 4: 144.00

Product 5: 144.00

3. Daily Sales Tracker
    - Input: Array of daily sales amounts.
    - Process: Use do-while to validate sales data input. Use a function to calculate total sales using pointers.
    - Output: Display total sales for the day.
    - Concepts: Loops, arrays, pointers, functions.

```
#include <stdio.h>
#define SIZE 5
```

```
// Function prototype
float calculateTotalSales(float *sales);

int main()
{
    float sales[SIZE];
    int i = 0;

    // Input the collect daily sales data using do-while loop
    printf("Enter sales amounts for %d items\n", SIZE);
    do {
        printf("Sales for item %d: ", i + 1);
        scanf("%f", &sales[i]);
        if (sales[i] < 0)
            printf("Invalid input\n");
        else
            i++;
    } while (i < SIZE);

    // Call the function to calculate total sales
    float totalSales = calculateTotalSales(sales);

    printf("Total sales for the say: %.2f\n", totalSales);
    return 0;
}
```

```
/*
Name: calculateTotalSales()
Return Type: float
Parameter:(data type of each parameter): float*
Short description: it is used to adjust prices based on demand levels
*/

// Function to calculate total sales using pointers
float calculateTotalSales(float *sales)
{
   float total = 0.0;
   for (int i = 0; i < SIZE; i++)
      total += *(sales + i);
   return total;
}
```

O/P:

```
Enter sales amounts for 5 items
Sales for item 1: 50
Sales for item 2: 150
Sales for item 3: 250
Sales for item 4: 350
Sales for item 5: 450
Total sales for the say: 1250.00
```

4. Discount Decision System
    - Input: Array of sales volumes.
    - Process: Pass the sales volume array by reference to a function. Use a switch statement to assign discount rates.
    - Output: Print discount rates for each product.
    - Concepts: Decision-making (switch), arrays, pass by reference, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void discount_rates(const int *salesVolumes, float *discountRates);

int main()
{
   int salesVolumes[SIZE];
   float discountRates[SIZE];

   // Input the sales volumes for each product
   printf("Enter sales volumes for %d products\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Sales volume for product %d: ", i + 1);
      scanf("%d", &salesVolumes[i]);
   }
   // Call the function to assign discount rates
   discount_rates(salesVolumes, discountRates);
```

```
    printf("\nDiscount rates\n");
    for (int i = 0; i < SIZE; i++)
        printf("Product %d: %.2f%%\n", i + 1, discountRates[i]);
    return 0;
}

/*
Name: discount_rates()
Return Type: void
Parameter:(data type of each parameter): const int* and float*
Short description: it is used to assign discount rates based on sales volumes
*/

// Function to assign discount rates based on sales volumes
void discount_rates(const int *salesVolumes, float *discountRates)
{
    for (int i = 0; i < SIZE; i++)
    {
        switch (salesVolumes[i] / 100)  // Use ranges of 100
        {
            case 0: discountRates[i] = 0.0; // Sales volume: 0-99
                    break;
            case 1:  discountRates[i] = 5.0; // Sales volume: 100-199
                     break;
            case 2:  discountRates[i] = 10.0; // Sales volume: 200-299
                     break;
```

```
            case 3:  discountRates[i] = 15.0; // Sales volume: 300-399
                    break;
             default: discountRates[i] = 20.0; // Sales volume: 400 and
above
                    break;
        }
    }
}
```

O/P:

```
Enter sales volumes for 5 products
Sales volume for product 1: 500
Sales volume for product 2: 300
Sales volume for product 3: 100
Sales volume for product 4: 400
Sales volume for product 5: 200

Discount rates
Product 1: 20.00%
Product 2: 15.00%
Product 3: 5.00%
Product 4: 20.00%
Product 5: 10.00%
```

5. Transaction Anomaly Detector
    - Input: Array of transaction amounts.

- Process: Use pointers to traverse the array. Classify transactions as "Normal" or "Suspicious" based on thresholds using if-else.
- Output: Print classification for each transaction.
- Concepts: Arrays, pointers, loops, decision-making.

```
#include <stdio.h>
#define SIZE 5
#define LOWER_THRESHOLD 100.0
#define UPPER_THRESHOLD 1000.0

// Function prototype
void classifyTransactions(const float *transactions);

int main()
{
    float transactions[SIZE];

    // Input the transaction amounts
    printf("Enter transaction amounts for %d transactions\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Transaction %d amount: ", i + 1);
        scanf("%f", &transactions[i]);
    }

    // Call the function to classify transactions
    printf("\nTransaction classifications:\n");
    classifyTransactions(transactions);
```

```
    return 0;
}

/*
Name: classifyTransactions()
Return Type: void
Parameter:(data type of each parameter): const float*
Short description: it is used to classify transactions
*/

// Function to classify transactions
void classifyTransactions(const float *transactions)
{
    for (int i = 0; i < SIZE; i++)
    {
            if (*(transactions + i) < LOWER_THRESHOLD || *(transactions + i) > UPPER_THRESHOLD)
                printf("Transaction %d: %.2f - Suspicious\n", i + 1, *(transactions + i));
        else
                printf("Transaction %d: %.2f - Normal\n", i + 1, *(transactions + i));
    }
}
```

O/P:

```
Enter transaction amounts for 5 transactions
Transaction 1 amount: 800
```

Transaction 2 amount: 500

Transaction 3 amount: 1200

Transaction 4 amount: 400

Transaction 5 amount: 100

Transaction classifications:

Transaction 1: 800.00 - Normal

Transaction 2: 500.00 - Normal

Transaction 3: 1200.00 - Suspicious

Transaction 4: 400.00 - Normal

Transaction 5: 100.00 – Normal

6. Account Balance Operations
   - Input: Array of account balances.
   - Process: Pass the balances array to a function that calculates interest. Return a pointer to the updated balances array.
   - Output: Display updated balances.
   - Concepts: Functions, arrays, pointers, loops.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
#define INTEREST_RATE 0.05

// Function prototype
float* calculateInterest(float *balances);
```

```
int main()
{
    float balances[SIZE];

    // Input the account balances
    printf("Enter the account balances for %d accounts\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Balance for account %d: ", i + 1);
        scanf("%f", &balances[i]);
    }

    // Call the function to caculate updated balances with interest
    float *updatedBalances = calculateInterest(balances);

    printf("\nUpdated account balances with interest:\n");
    for (int i = 0; i < SIZE; i++)
        printf("Account %d: %.2f\n", i + 1, updatedBalances[i]);
    free(updatedBalances);
    return 0;
}

/*
Name: calculateInterest()
Return Type: float*
Parameter:(data type of each parameter): float*
Short description: it is used to calculate interest and return updated
balances
```

```
*/

// Function to calculate interest and return updated balances
float* calculateInterest(float *balances)
{
   // Allocate memory for updated balances
   float *updatedBalances = (float*) malloc(SIZE * sizeof(float));
   if (updatedBalances == NULL)
   {
      printf("Memory allocation failed!\n");
      exit(1);
   }

   // Calculate interest for each balance
   for (int i = 0; i < SIZE; i++)
            updatedBalances[i] = balances[i] + (balances[i] *
INTEREST_RATE);
   return updatedBalances;
}
```

O/P:

```
Enter the account balances for 5 accounts
Balance for account 1: 1500
Balance for account 2: 1000
Balance for account 3: 500
Balance for account 4: 300
Balance for account 5: 800
```

Updated account balances with interest:

Account 1: 1575.00

Account 2: 1050.00

Account 3: 525.00

Account 4: 315.00

Account 5: 840.00

7. Bank Statement Generator
   - Input: Array of transaction types (e.g., 1 for Deposit, 2 for Withdrawal) and amounts.
   - Process: Use a switch statement to classify transactions. Pass the array as a constant parameter to a function.
   - Output: Summarize total deposits and withdrawals.
   - Concepts: Decision-making, passing constant data, arrays, functions.

```
#include <stdio.h>

#define SIZE 5


// Function prototype

void summarizeTransactions(const int *transactionTypes, const float *transactionAmounts, float *totalDeposits, float *totalWithdrawals);


int main()

{

   int transactionTypes[SIZE];

   float transactionAmounts[SIZE];
```

```
    float totalDeposits = 0.0, totalWithdrawals = 0.0;

    // Input the transaction types and amounts
    printf("Enter transaction details for %d transactions\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Transaction %d type (1 for Deposit, 2 for Withdrawal): ", i + 1);
        scanf("%d", &transactionTypes[i]);
        if (transactionTypes[i] != 1 && transactionTypes[i] != 2)
        {
            printf("Invalid transaction type\n");
            i--;
            continue;
        }
        printf("Transaction %d amount: ", i + 1);
        scanf("%f", &transactionAmounts[i]);
        if (transactionAmounts[i] < 0)
        {
            printf("Invalid transaction amount\n");
            i--;
        }
    }

    // Call the function to summarize total deposits and withdrawals
      summarizeTransactions(transactionTypes,  transactionAmounts, &totalDeposits, &totalWithdrawals);
    printf("\nBank statement summary\n");
```

```
    printf("Total Deposits: %.2f\n", totalDeposits);
    printf("Total Withdrawals: %.2f\n", totalWithdrawals);
    return 0;
}

/*
Name: summarizeTransactions()
Return Type: void
Parameter:(data type of each parameter): const int*, const float*, float* and float*
Short description: it is used to summarize deposits and withdrawals
*/

// Function to summarize deposits and withdrawals
void summarizeTransactions(const int *transactionTypes, const float *transactionAmounts, float *totalDeposits, float *totalWithdrawals)
{
    for (int i = 0; i < SIZE; i++)
    {
        switch (transactionTypes[i]) {
            case 1: *totalDeposits += transactionAmounts[i];
                    break;
             case 2: *totalWithdrawals += transactionAmounts[i];
                    break;
            default:printf("Invalid transaction type\n");
                    break;
        }
```

```
    }
}
```

O/P:

```
Enter transaction details for 5 transactions
Transaction 1 type (1 for Deposit, 2 for Withdrawal): 1
Transaction 1 amount: 1000
Transaction 2 type (1 for Deposit, 2 for Withdrawal): 2
Transaction 2 amount: 500
Transaction 3 type (1 for Deposit, 2 for Withdrawal): 2
Transaction 3 amount: 300
Transaction 4 type (1 for Deposit, 2 for Withdrawal): 1
Transaction 4 amount: 1500
Transaction 5 type (1 for Deposit, 2 for Withdrawal): 1
Transaction 5 amount: 2000

Bank statement summary
Total Deposits: 4500.00
Total Withdrawals: 800.0
```

8. Loan Eligibility Check
   - Input: Array of customer credit scores.
   - Process: Use if-else to check eligibility criteria. Use pointers to update eligibility status.
   - Output: Print customer eligibility statuses.
   - Concepts: Decision-making, arrays, pointers, functions.

```
#include <stdio.h>
#define SIZE 5
#define MIN_CREDIT_SCORE 800

// Function prototype
void checkLoanEligibility(const int *creditScores, char *eligibilityStatuses);

int main()
{
    int creditScores[SIZE];
    char eligibilityStatuses[SIZE];

    // Input the customer credit scores
    printf("Enter credit scores for %d customers\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Credit score for customer %d: ", i + 1);
        scanf("%d", &creditScores[i]);
        if (creditScores[i] < 0 || creditScores[i] > 900)
        {
            printf("Invalid credit score\n");
            i--;
        }
    }

    // Call the function to check loan eligibility
    checkLoanEligibility(creditScores, eligibilityStatuses);
```

```
    // Output to display eligibility statuses
    printf("\nLoan eligibility statuses\n");
    for (int i = 0; i < SIZE; i++)
    {
        printf("Customer %d: Credit Score: %d - %s\n", i + 1, creditScores[i],
            eligibilityStatuses[i] == 'Y' ? "Eligible" : "Not Eligible");
    }
    return 0;
}

/*
Name: checkLoanEligibility()
Return Type: void
Parameter:(data type of each parameter): const int* and char*
Short description: it is used to check loan eligibility
*/

// Function to check loan eligibility
void checkLoanEligibility(const int *creditScores, char *eligibilityStatuses)
{
    for (int i = 0; i < SIZE; i++)
    {
        if (*(creditScores + i) >= MIN_CREDIT_SCORE)
            *(eligibilityStatuses + i) = 'Y';
        else
```

```
            *(eligibilityStatuses + i) = 'N';
        }
    }
```

O/P:

```
Enter credit scores for 5 customers
Credit score for customer 1: 400
Credit score for customer 2: 500
Credit score for customer 3: 700
Credit score for customer 4: 800
Credit score for customer 5: 900

Loan eligibility statuses
Customer 1: Credit Score: 400 - Not Eligible
Customer 2: Credit Score: 500 - Not Eligible
Customer 3: Credit Score: 700 - Not Eligible
Customer 4: Credit Score: 800 - Eligible
Customer 5: Credit Score: 900 – Eligible
```

9. Order Total Calculator
    - Input: Array of item prices.
    - Process: Pass the array to a function. Use pointers to calculate the total cost.
    - Output: Display the total order value.
    - Concepts: Arrays, pointers, functions, loops.

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

```
#define SIZE 5

// Function prototype
float calculateTotal(const float *prices);

int main()
{
    float itemPrices[SIZE];

    // Input the item prices
    printf("Enter the prices of %d items\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Price of item %d: ", i + 1);
        scanf("%f", &itemPrices[i]);
        if (itemPrices[i] < 0)
        {
            printf("Invalid price\n");
            i--;
        }
    }

    // Call the function to calculate the total order value
    float totalOrderValue = calculateTotal(itemPrices);

    // Output to display the total order value
    printf("Total Order Value: %.2f\n", totalOrderValue);
```

```
    return 0;
}

/*
Name: calculateTotal()
Return Type: float
Parameter:(data type of each parameter): const float*
Short description: it is used to calculate the total cost
*/

// Function to calculate the total cost
float calculateTotal(const float *prices)
{
    float total = 0.0;
    for (int i = 0; i < SIZE; i++)
        total += *(prices + i);
    return total;
}
```

O/P:

```
Enter the prices of 5 items
Price of item 1: 1000
Price of item 2: 500
Price of item 3: 800
Price of item 4: 1500
Price of item 5: 300
Total Order Value: 4100.00
```

10. Stock Replenishment Alert
    - Input: Array of inventory levels.
    - Process: Use a function to flag products below a threshold. Return a pointer to flagged indices.
    - Output: Display flagged product indices.
    - Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
#define THRESHOLD 10

// Function prototype
int* flagLowStock(const int *inventory, int *flagCount);

int main()
{
   int inventoryLevels[SIZE];
   int flagCount = 0;

   // Input thenventory levels
   printf("Enter inventory levels for %d products\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Inventory for product %d: ", i + 1);
      scanf("%d", &inventoryLevels[i]);
      if (inventoryLevels[i] < 0)
      {
```

```
            printf("Invalid inventory level\n");
            i--;
        }
    }

    // Call the function to show flag products below the threshold
        int *flaggedIndices = flagLowStock(inventoryLevels, &flagCount);

    // Output to display flagged product indices
    if (flagCount > 0)
    {
        printf("\nProducts that need replenishment\n");
        for (int i = 0; i < flagCount; i++)
            printf("Product %d (Index %d)\n", flaggedIndices[i] + 1, flaggedIndices[i]);
    }
    free(flaggedIndices);
    return 0;
}

/*
Name: flagLowStock()
Return Type: int*
Parameter:(data type of each parameter): const int* and int*
Short description: it is used to flag products below the threshold
*/
```

```
// Function to flag products below the threshold
int* flagLowStock(const int *inventory, int *flagCount)
{
   int *flags = (int*) malloc(SIZE * sizeof(int));
   if (flags == NULL)
   {
      printf("Memory allocation failed!\n");
      exit(1);
   }
   *flagCount = 0;
   for (int i = 0; i < SIZE; i++)
   {
      if (*(inventory + i) < THRESHOLD)
      {
         flags[*flagCount] = i;
         (*flagCount)++;
      }
   }
   return flags;
}
```

O/P:

```
Enter inventory levels for 5 products
Inventory for product 1: 12
Inventory for product 2: 15
Inventory for product 3: 18
Inventory for product 4: 8
```

Inventory for product 5: 10

Products that need replenishment

Product 4 (Index 3)

11. Customer Reward Points

- Input: Array of customer purchase amounts.
- Process: Pass the purchase array by reference to a function that calculates reward points using if-else.
- Output: Display reward points for each customer.
- Concepts: Arrays, functions, pass by reference, decision-making.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void calculateRewardPoints(const float *purchases, int *rewardPoints);

int main()
{
   float purchaseAmounts[SIZE];
   int rewardPoints[SIZE];

   // Input the customer purchase amounts
   printf("Enter purchase amounts for %d customers\n", SIZE);
   for (int i = 0; i < SIZE; i++)
```

```
    {
        printf("Purchase amount for customer %d: ", i + 1);
        scanf("%f", &purchaseAmounts[i]);
        if (purchaseAmounts[i] < 0)
        {
            printf("Invalid amount\n");
            i--;
        }
    }

    // Call the function to calculate reward points
    calculateRewardPoints(&purchaseAmounts[0],
&rewardPoints[0]);

    // Output to display reward points for each customer
    printf("\nCustomer reward points\n");
    for (int i = 0; i < SIZE; i++)
        printf("Customer %d: %d\n", i + 1, rewardPoints[i]);
    return 0;
}

/*
Name: calculateRewardPoints()
Return Type: void
Parameter:(data type of each parameter): const float* and int*
Short description: it is used to calculate reward points
*/
```

```
// Function to calculate reward points
void calculateRewardPoints(const float *purchases, int *rewardPoints)
{
   for (int i = 0; i < SIZE; i++)
   {
      if (*(purchases + i) >= 1000)
         *(rewardPoints + i) = 50;
      else if (*(purchases + i) >= 500)
         *(rewardPoints + i) = 25;
      else if (*(purchases + i) >= 100)
         *(rewardPoints + i) = 10;
      else
         *(rewardPoints + i) = 0;
   }
}
```

O/P:

```
Enter purchase amounts for 5 customers
Purchase amount for customer 1: 1245
Purchase amount for customer 2: 800
Purchase amount for customer 3: 300
Purchase amount for customer 4: 75
Purchase amount for customer 5: 1000

Customer reward points
Customer 1: 50
Customer 2: 25
```

Customer 3: 10

Customer 4: 0

Customer 5: 50

12. Shipping Cost Estimator

- Input: Array of order weights and shipping zones.
- Process: Use a switch statement to calculate shipping costs based on zones. Pass the weight array as a constant parameter.
- Output: Print the shipping cost for each order.
- Concepts: Decision-making, passing constant data, arrays, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void calculateShippingCosts(const float *weights, const int *zones, float *costs);

int main()
{
    float orderWeights[SIZE];
    int shippingZones[SIZE];
    float shippingCosts[SIZE];

    // Input the order weights and shipping zones
    printf("Enter the order weights (in kg) and corresponding shipping zones (1-3)\n");
```

```
    for (int i = 0; i < SIZE; i++)
    {
        printf("Order %d - Weight (kg): ", i + 1);
        scanf("%f", &orderWeights[i]);
        if (orderWeights[i] <= 0)
        {
            printf("Invalid weight\n");
            i--;
            continue;
        }
        printf("Order %d - Shipping zone (1-3): ", i + 1);
        scanf("%d", &shippingZones[i]);
        if (shippingZones[i] < 1 || shippingZones[i] > 3)
        {
            printf("Invalid zone\n");
            i--;
        }
    }

    // Call the function to calculate shipping costs
    calculateShippingCosts(orderWeights, shippingZones, shippingCosts);

    // Output to display shipping costs for each order
    printf("\nShipping costs\n");
    for (int i = 0; i < SIZE; i++)
    {
```

```
        printf("Order %d - Weight: %.2f kg, Zone: %d, Shipping Cost: %.2f\n",
               i + 1, orderWeights[i], shippingZones[i], shippingCosts[i]);
    }
    return 0;
}

/*
Name: calculateShippingCosts()
Return Type: void
Parameter:(data type of each parameter): const float*, const int* and float*
Short description: it is used to calculate reward points
*/

// Function to calculate shipping costs
void calculateShippingCosts(const float *weights, const int *zones, float *costs)
{
    for (int i = 0; i < SIZE; i++)
    {
        switch (*(zones + i))
        {
            case 1: *(costs + i) = *(weights + i) * 5.0;
                    break; // Zone 1 - 5 per kg
            case 2: *(costs + i) = *(weights + i) * 7.0;
                    break; // Zone 2 - 7 per kg
            case 3: *(costs + i) = *(weights + i) * 10.0;
```

```
            break; // Zone 3 - 10 per kg
        default:*(costs + i) = 0.0;
            break;
    }
  }
}
```

O/P:

Enter the order weights (in kg) and corresponding shipping zones (1-3)

Order 1 - Weight (kg): 20

Order 1 - Shipping zone (1-3): 1

Order 2 - Weight (kg): 80

Order 2 - Shipping zone (1-3): 3

Order 3 - Weight (kg): 40

Order 3 - Shipping zone (1-3): 2

Order 4 - Weight (kg): 35

Order 4 - Shipping zone (1-3): 1

Order 5 - Weight (kg): 50

Order 5 - Shipping zone (1-3): 2

Shipping costs

Order 1 - Weight: 20.00 kg, Zone: 1, Shipping Cost: 100.00

Order 2 - Weight: 80.00 kg, Zone: 3, Shipping Cost: 800.00

Order 3 - Weight: 40.00 kg, Zone: 2, Shipping Cost: 280.00

Order 4 - Weight: 35.00 kg, Zone: 1, Shipping Cost: 175.00

Order 5 - Weight: 50.00 kg, Zone: 2, Shipping Cost: 350.00

13.Missile Trajectory Analysis

- Input: Array of trajectory data points.
- Process: Use functions to find maximum and minimum altitudes. Use pointers to access data.
- Output: Display maximum and minimum altitudes.
- Concepts: Arrays, pointers, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototypes
float findMaxAltitude(const float *data, int size);
float findMinAltitude(const float *data, int size);

int main()
{
   float trajectoryData[SIZE];

   // Input the trajectory data points
   printf("Enter %d trajectory altitude points\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Altitude point %d: ", i + 1);
      scanf("%f", &trajectoryData[i]);
   }

   // Call the function to find maximum and minimum altitudes
   float maxAltitude = findMaxAltitude(trajectoryData, SIZE);
```

```
    float minAltitude = findMinAltitude(trajectoryData, SIZE);

    // Display maximum and minimum altitudes
    printf("\nTrajectory analysis\n");
    printf("Maximum altitude: %.2f meters\n", maxAltitude);
    printf("Minimum altitude: %.2f meters\n", minAltitude);
    return 0;
}

/*
Name: findMaxAltitude()
Return Type: float
Parameter:(data type of each parameter): const float* and int
Short description: it is used to find the maximum altitude
*/

// Function to find the maximum altitude
float findMaxAltitude(const float *data, int size)
{
    float max = *data;
    for (int i = 1; i < size; i++)
    {
        if (*(data + i) > max)
            max = *(data + i);
    }
    return max;
}
```

```
/*
Name: findMinAltitude()
Return Type: float
Parameter:(data type of each parameter): const float* and int
Short description: it is used to find the minimum altitude
*/

// Function to find the minimum altitude
float findMinAltitude(const float *data, int size)
{
   float min = *data;
   for (int i = 1; i < size; i++)
   {
      if (*(data + i) < min)
         min = *(data + i);
   }
   return min;
}
```

O/P:

```
Enter 5 trajectory altitude points
Altitude point 1: 500
Altitude point 2: 800
Altitude point 3: 1200
Altitude point 4: 900
Altitude point 5: 100
```

Trajectory analysis

Maximum altitude: 1200.00 meters

Minimum altitude: 100.00 meters

14. Target Identification System

- Input: Array of radar signal intensities.
- Process: Classify signals into categories using a switch statement. Return a pointer to the array of classifications.
- Output: Display classified signal types.
- Concepts: Decision-making, functions returning pointers, arrays.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
char* classifySignals(const float *signals, char *categories, int size);

int main()
{
   float signalIntensities[SIZE];
   char classifications[SIZE];

   // Input the radar signal intensities
   printf("Enter %d radar signal intensities\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Signal intensity %d: ", i + 1);
```

```
        scanf("%f", &signalIntensities[i]);
        if (signalIntensities[i] < 0)
        {
            printf("Invalid intensity\n");
            i--;
        }
    }

    // Call the function to classify signals
    classifySignals(signalIntensities, classifications, SIZE);

    // Display classified signal types
    printf("\nRadar signal classification\n");
    for (int i = 0; i < SIZE; i++)
        printf("Signal %d: Intensity %.2f - Category %c\n", i + 1,
signalIntensities[i], classifications[i]);
    return 0;
}

/*
Name: classifySignals()
Return Type: char*
Parameter:(data type of each parameter): const float*, char* and int
Short description: it is used to classify signals
*/

// Function to classify signals
char* classifySignals(const float *signals, char *categories, int size)
```

```
{
    for (int i = 0; i < size; i++)
    {
        // Classification based on intensity
        // Divide intensity by 100 and cast to integer
        switch ((int)(*(signals + i) / 100))
        {
            case 0: *(categories + i) = 'L';
                    break; // (0 - 99) Low signal
            case 1: *(categories + i) = 'M';
                    break; // (100 - 299) Medium signal
            default: *(categories + i) = 'H';
                     break;// (300 and above) High signal
        }
    }
    return categories;
}
```

O/P:

```
Enter 5 radar signal intensities
Signal intensity 1: 50
Signal intensity 2: 500
Signal intensity 3: 8
Signal intensity 4: 1000
Signal intensity 5: 69

Radar signal classification
```

Signal 1: Intensity 50.00 - Category L

Signal 2: Intensity 500.00 - Category H

Signal 3: Intensity 8.00 - Category L

Signal 4: Intensity 1000.00 - Category H

Signal 5: Intensity 69.00 - Category L

15. Threat Level Assessment

- Input: Array of sensor readings.
- Process: Pass the array by reference to a function that uses if-else to categorize threats.
- Output: Display categorized threat levels.
- Concepts: Arrays, functions, pass by reference, decision-making.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void categorizeThreats(const float *readings, char *threatLevels, int size);

int main()
{
   float sensorReadings[SIZE];
   char threatLevels[SIZE];

   // Input the sensor readings
   printf("Enter %d sensor readings\n", SIZE);
```

```
    for (int i = 0; i < SIZE; i++)
    {
        printf("Reading %d: ", i + 1);
        scanf("%f", &sensorReadings[i]);
        if (sensorReadings[i] < 0)
        {
            printf("Invalid reading\n");
            i--;
        }
    }

    // Call the function to categorize threats
    categorizeThreats(sensorReadings, threatLevels, SIZE);

    // Display categorized threat levels
    printf("\nThreat level assessment\n");
    for (int i = 0; i < SIZE; i++)
            printf("Sensor Reading %.2f - Threat Level: %c\n",
sensorReadings[i], threatLevels[i]);
    return 0;
}

/*
Name: categorizeThreats()
Return Type: void
Parameter:(data type of each parameter): const float*, char* and int
Short description: it is used to categorize threats
*/
```

```
// Function to categorize threats
void categorizeThreats(const float *readings, char *threatLevels, int size)
{
    for (int i = 0; i < size; i++)
    {
        if (*(readings + i) < 50)
            *(threatLevels + i) = 'L';
        else if (*(readings + i) >= 50 && *(readings + i) < 100)
            *(threatLevels + i) = 'M';
        else
            *(threatLevels + i) = 'H';
    }
}
```

O/P:

```
Enter 5 sensor readings
Reading 1: 75
Reading 2: 45
Reading 3: 50
Reading 4: 100
Reading 5: 150

Threat level assessment
Sensor Reading 75.00 - Threat Level: M
Sensor Reading 45.00 - Threat Level: L
Sensor Reading 50.00 - Threat Level: M
Sensor Reading 100.00 - Threat Level: H
```

Sensor Reading 150.00 - Threat Level: H

16. Signal Calibration

- Input: Array of raw signal data.
- Process: Use a function to adjust signal values by reference. Use pointers for data traversal.
- Output: Print calibrated signal values.
- Concepts: Arrays, pointers, functions, loops.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void calibrateSignal(float *data, int size);

int main()
{
   float rawSignalData[SIZE];

   // Input the raw signal data
   printf("Enter %d raw signal values:\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Signal %d: ", i + 1);
      scanf("%f", &rawSignalData[i]);
   }
```

```
    // Call the function to calibrate the signal data
    calibrateSignal(&rawSignalData[0], SIZE);

    //Display calibrated signal values
    printf("\nCalibrated signal values\n");
    for (int i = 0; i < SIZE; i++)
        printf("Calibrated Signal %d: %.2f\n", i + 1, rawSignalData[i]);
    return 0;
}

/*
Name: calibrateSignal()
Return Type: void
Parameter:(data type of each parameter): float* and int
Short description: it is used to calibrate the signal by adjusting the signal values
*/

// Function to calibrate the signal by adjusting the signal values
void calibrateSignal(float *data, int size)
{
    for (int i = 0; i < size; i++)
        *(data + i) *= 1.1;  // Adjust by a factor of 10%
}
```

O/P:

```
Enter 5 raw signal values:
Signal 1: 150
```

Signal 2: 800

Signal 3: 10

Signal 4: 600

Signal 5: 1000

Calibrated signal values

Calibrated Signal 1: 165.00

Calibrated Signal 2: 880.00

Calibrated Signal 3: 11.00

Calibrated Signal 4: 660.00

Calibrated Signal 5: 1100.00

17.Matrix Row Sum

- Input: 2D array representing a matrix.
- Process: Write a function that calculates the sum of each row. The function returns a pointer to an array of row sums.
- Output: Display the row sums.
- Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>

#define ROWS 3

#define COLUMNS 4


// Function prototype

int* calculateRowSums(int matrix[ROWS][COLUMNS], int *rowSums, int rows, int cols);
```

```
int main()
{
    int matrix[ROWS][COLUMNS];
    int rowSums[ROWS];

    // Input the matrix elements
    printf("Enter the elements of the matrix (%d x %d):\n", ROWS, COLUMNS);
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            printf("Matrix[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }

    // Call the function to calculate the row sums
    calculateRowSums(matrix, rowSums, ROWS, COLUMNS);

    // Display row sums
    printf("\nRow sums\n");
    for (int i = 0; i < ROWS; i++)
        printf("Row %d sum: %d\n", i + 1, rowSums[i]);
    return 0;
}

/*
```

Name: calculateRowSums()

Return Type: int*

Parameter:(data type of each parameter): int, int*, int and int

Short description: it is used to calculate the sum of each row in the matrix

*/

```
// Function to calculate the sum of each row in the matrix
int* calculateRowSums(int matrix[ROWS][COLUMNS], int *rowSums, int rows, int columns)
{
   for (int i = 0; i < rows; i++)
   {
      rowSums[i] = 0;
      for (int j = 0; j < columns; j++)
         rowSums[i] += matrix[i][j];
   }
   return rowSums;
}
```

O/P:

```
Enter the elements of the matrix (3 x 4):
Matrix[1][1]: 4
Matrix[1][2]: 5
Matrix[1][3]: 6
Matrix[1][4]: 7
Matrix[2][1]: 8
Matrix[2][2]: 9
```

Matrix[2][3]: 1

Matrix[2][4]: 2

Matrix[3][1]: 3

Matrix[3][2]: 5

Matrix[3][3]: 6

Matrix[3][4]: 7

Row sums

Row 1 sum: 22

Row 2 sum: 20

Row 3 sum: 21

18. Statistical Mean Calculator

- Input: Array of data points.
- Process: Pass the data array as a constant parameter. Use pointers to calculate the mean.
- Output: Print the mean value.
- Concepts: Passing constant data, pointers, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
double calculateMean(const double *data, int size);

int main()
{
```

```
    double dataPoints[SIZE];

    // Input the array of data points
    printf("Enter %d data points\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Data point %d: ", i + 1);
        scanf("%lf", &dataPoints[i]);
    }

    // Call the function to calculate the mean
    double mean = calculateMean(dataPoints, SIZE);

    // Display the mean value
    printf("\nThe mean of the data points is: %.2f\n", mean);
    return 0;
}

/*
Name: calculateMean()
Return Type: double
Parameter:(data type of each parameter): const double* and int
Short description: it is used to calculate the mean
*/

// Function to calculate the mean
double calculateMean(const double *data, int size)
```

```
{
    double sum = 0;
    for (int i = 0; i < size; i++)
        sum += *(data + i);
    return sum / size;
}
```

O/P:

```
Enter 5 data points
Data point 1: 18.5
Data point 2: 15.8
Data point 3: 8.8
Data point 4: 50.6
Data point 5: 79

The mean of the data points is: 34.54
```

19. Temperature Gradient Analysis
    - Input: Array of temperature readings.
    - Process: Compute the gradient using a function that returns a pointer to the array of gradients.
    - Output: Display temperature gradients.
    - Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>
#define SIZE 5
```

```
// Function prototype
float* calculateTemperatureGradient(float *readings, float *gradients, int size);

int main()
{
    float temperatureReadings[SIZE];

    // Gradients array will have one less element than the readings
    float gradients[SIZE - 1];

    // Input the array of temperature readings
    printf("Enter %d temperature readings\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Reading %d: ", i + 1);
        scanf("%f", &temperatureReadings[i]);
    }

    // Call the function to calculate the temperature gradients
     calculateTemperatureGradient(temperatureReadings,  gradients, SIZE);

    // Display the temperature gradients
    printf("\nTemperature gradients\n");
    for (int i = 0; i < SIZE - 1; i++)
        printf("Gradient %d: %.2f\n", i + 1, gradients[i]);
    return 0;
```

```
}

/*
Name: calculateTemperatureGradient()
Return Type: float
Parameter:(data type of each parameter): float*, float* and int
Short description: it is used to calculate the temperature gradient
*/

// Function to calculate the temperature gradient
float* calculateTemperatureGradient(float *readings, float *gradients, int size)
{
   for (int i = 0; i < size - 1; i++)
      // Gradient = difference between consecutive readings
      *(gradients + i) = *(readings + i + 1) - *(readings + i);
   return gradients;
}
```

O/P:

```
Enter 5 temperature readings
Reading 1: 28.2
Reading 2: 29.3
Reading 3: 32.4
Reading 4: 30
Reading 5: 19

Temperature gradients
```

Gradient 1: 1.10

Gradient 2: 3.10

Gradient 3: -2.40

Gradient 4: -11.00

20. Data Normalization

- Input: Array of data points.
- Process: Pass the array by reference to a function that normalizes values to a range of 0–1 using pointers.
- Output: Display normalized values.
- Concepts: Arrays, pointers, pass by reference, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void normalizeData(float *data, int size);

int main()
{
   float dataPoints[SIZE];

   // Input the array of data points
   printf("Enter %d data points\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Data point %d: ", i + 1);
```

```
        scanf("%f", &dataPoints[i]);
    }

    // Call the function to normalize the data
    normalizeData(dataPoints, SIZE);

    // Display normalized values
    printf("\nNormalized data points\n");
    for (int i = 0; i < SIZE; i++)
        printf("Normalized data point %d: %.2f\n", i + 1, dataPoints[i]);
    return 0;
}

/*
Name: normalizeData()
Return Type: void
Parameter:(data type of each parameter): float* and int
Short description: it is used to normalize data points
*/

// Function to normalize data points to a range of 0-1
void normalizeData(float *data, int size)
{
    float min = *data, max = *data;

    // Find the minimum and maximum values in the data
    for (int i = 1; i < size; i++)
```

```
    {
        if (*(data + i) < min)
            min = *(data + i);
        else if (*(data + i) > max)
            max = *(data + i);
    }

    // Normalize the data points
    for (int i = 0; i < size; i++)
        *(data + i) = (*(data + i) - min) / (max - min);
}
```

O/P:

```
Enter 5 data points
Data point 1: 15.8
Data point 2: 76.4
Data point 3: 18.8
Data point 4: 65.4
Data point 5: 10.5

Normalized data points
Normalized data point 1: 0.08
Normalized data point 2: 1.00
Normalized data point 3: 0.13
Normalized data point 4: 0.83
Normalized data point 5: 0.00
```

21. Exam Score Analysis

- Input: Array of student scores.
- Process: Write a function that returns a pointer to the highest score. Use loops to calculate the average score.
- Output: Display the highest and average scores.
- Concepts: Arrays, functions returning pointers, loops.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
float* findHighestScore(float *scores, int size);

int main()
{
   float scores[SIZE];

   // Input the array of student scores
   printf("Enter %d student scores\n", SIZE);
   for (int i = 0; i < SIZE; i++)
   {
      printf("Score %d: ", i + 1);
      scanf("%f", &scores[i]);
   }

   float averageScore, sum = 0;
   for (int i = 0; i < SIZE; i++)
      sum += *(scores + i);
```

```
    averageScore = sum / SIZE;
    printf("Average score: %.2f\n", averageScore);

    // Call the function to find the highest score
    float *highestScore = findHighestScore(scores, SIZE);

    // Display the highest score
    printf("Highest score: %.2f\n", *highestScore);
    return 0;
}

/*
Name: findHighestScore()
Return Type: float*
Parameter:(data type of each parameter): float* and int
Short description: it is used to find the highest score
*/

// Function to find the highest score
float* findHighestScore(float *scores, int size)
{
    float *maxScore = scores;
    for (int i = 1; i < size; i++)
    {
        if (*(scores + i) > *maxScore)
            maxScore = scores + i;
    }
```

```
    return maxScore;
}
```

O/P:

```
Enter 5 student scores
Score 1: 45.6
Score 2: 57.4
Score 3: 75.5
Score 4: 94.3
Score 5: 34.7
Average score: 61.50
Highest score: 94.30
```

22. Grade Assignment

- Input: Array of student marks.
- Process: Pass the marks array by reference to a function. Use a switch statement to assign grades.
- Output: Display grades for each student.
- Concepts: Arrays, decision-making, pass by reference, functions.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void assignGrades(int *marks, char *grades, int size);

int main()
```

```
{
    int marks[SIZE];
    char grades[SIZE];

    // Input the array of student marks
    printf("Enter marks for %d students\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Student %d mark: ", i + 1);
        scanf("%d", &marks[i]);
    }

    // Call the function to assign grades based on marks
    assignGrades(marks, grades, SIZE);

    // Display grades for each student
    printf("\nStudent grades\n");
    for (int i = 0; i < SIZE; i++)
        printf("Student %d grade: %c\n", i + 1, grades[i]);
    return 0;
}

/*
Name: assignGrades()
Return Type: void
Parameter:(data type of each parameter): int*, char* and int
Short description: it is used to assign grades based on marks
```

```
*/

// Function to assign grades based on marks
void assignGrades(int *marks, char *grades, int size)
{
   for (int i = 0; i < size; i++)
   {
      switch (*(marks + i))
      {
         case 90 ... 100: *(grades + i) = 'A';
                          break;
         case 80 ... 89: *(grades + i) = 'B';
                         break;
         case 70 ... 79: *(grades + i) = 'C';
                         break;
         case 60 ... 69: *(grades + i) = 'D';
                         break;
         default: *(grades + i) = 'F'; // For marks below 60
                  break;
      }
   }
}
```

O/P:

```
Enter marks for 5 students
Student 1 mark: 35
Student 2 mark: 93
```

Student 3 mark: 88

Student 4 mark: 71

Student 5 mark: 64

Student grades

Student 1 grade: F

Student 2 grade: A

Student 3 grade: B

Student 4 grade: C

Student 5 grade: D

23. Student Attendance Tracker

- Input: Array of attendance percentages.
- Process: Use pointers to traverse the array. Return a pointer to an array of defaulters.
- Output: Display defaulters' indices.
- Concepts: Arrays, pointers, functions returning pointers.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
int* findDefaulters(float *attendance, int size, int *defaulterCount);

int main()
{
   float attendance[SIZE];
```

```
    int defaulterCount = 0;
    int *defaulters;

    // Input the array of attendance percentages
    printf("Enter attendance percentages for %d students\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Student %d attendance: ", i + 1);
        scanf("%f", &attendance[i]);
    }

    // Call the function to find defaulters
    defaulters = findDefaulters(attendance, SIZE, &defaulterCount);

    // Display defaulters' indices
    if (defaulterCount > 0)
    {
        printf("\nDefaulters (attendance < 75%%)\n");
        for (int i = 0; i < defaulterCount; i++)
            // Indices are 0-based, so add 1 to match student number
            printf("Student %d\n", *(defaulters + i) + 1);
    }
    else
        printf("\nNo defaulters found\n");
    return 0;
}
```

```
/*
Name: findDefaulters()
Return Type: int*
Parameter:(data type of each parameter): float*, int and int*
Short description: it is used to assign grades based on marks
*/

// Function to find defaulters and return their indices
int* findDefaulters(float *attendance, int size, int *defaulterCount)
{
   static int defaulterIndices[SIZE];
   *defaulterCount = 0;

   // Traverse the array to find students with attendance < 75%
   for (int i = 0; i < size; i++)
   {
      if (*(attendance + i) < 75)
      {
         defaulterIndices[*defaulterCount] = i;
         (*defaulterCount)++;
      }
   }
   return defaulterIndices;
}
```

O/P:

```
Enter attendance percentages for 5 students
```

Student 1 attendance: 90

Student 2 attendance: 74

Student 3 attendance: 75

Student 4 attendance: 86

Student 5 attendance: 65

Defaulters (attendance < 75%)

Student 2

Student 5

24. Quiz Performance Analyzer
    - Input: Array of quiz scores.
    - Process: Pass the array as a constant parameter to a function that uses if-else for performance categorization.
    - Output: Print categorized performance.
    - Concepts: Arrays, passing constant data, functions, decision-making.

```
#include <stdio.h>
#define SIZE 5

// Function prototype
void analyzePerformance(const int *scores, int size);

int main()
{
   int scores[SIZE];
```

```
    // Input the array of quiz scores
    printf("Enter quiz scores for %d students\n", SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        printf("Student %d score: ", i + 1);
        scanf("%d", &scores[i]);
    }

    // Call the function to analyze quiz performance
    analyzePerformance(scores, SIZE);
    return 0;
}

/*
Name: analyzePerformance()
Return Type: void
Parameter:(data type of each parameter): const int* and int
Short description: it is used to categorize performance based on quiz scores
*/

// Function to categorize performance based on quiz scores
void analyzePerformance(const int *scores, int size)
{
    printf("\n");
    for (int i = 0; i < size; i++)
    {
```

```
        if (*(scores + i) >= 90)
            printf("Student %d: Excellent\n", i + 1);
        else if (*(scores + i) >= 75)
            printf("Student %d: Good\n", i + 1);
        else if (*(scores + i) >= 50)
            printf("Student %d: Average\n", i + 1);
        else
            printf("Student %d: Poor\n", i + 1);
    }
}
```

O/P:

```
Enter quiz scores for 5 students
Student 1 score: 88
Student 2 score: 93
Student 3 score: 50
Student 4 score: 35
Student 5 score: 66

Student 1: Good
Student 2: Excellent
Student 3: Average
Student 4: Poor
Student 5: Average
```