Codekata Report:

Name: Deepu Pandey

Email: deepu.24scse1011405@galgotiasuniversity.ac.in

Specialization: School of Computing Science & Engineering

Completion Year: 2028

Section: Section-13

1. Problem: Mixed Data Type CalculationsObjective:

Write a C program that reads a series of values, performs arithmetic calculations, and then outputs the results. The values and operations are defined as follows:

Input:

Integer n which is representing the number of sets of data to process.For each set:An integer value a.A floating-point number b.A character op representing an arithmetic operation (+, -, *, /).A double precision number c.Output:

For each set of data, perform the operation specified by op on a and b, and add the result to c.Output the result as a double-precision floating-point number with two decimal places.Constraints:

1 <= n <= 1000 <= a <= 10000.0 <= b <= 1000.0op will be one of the characters: +, -, *, /.c will be a valid double precision number.If the operation is division (/), ensure no division by zero occurs.

ExampleInput:35 3.2 + 2.58 4.0 * 1.210 0.0 / 10.0

Output:10.7033.2010.00

Explanation:

First Set: 5 + 3.2 + 2.5 = 10.7Second Set: 8 * 4.0 + 1.2 = 33.2Third Set: 10.0 + 0.0 = 10.0 (Division by zero avoided)

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n;
scanf("%d", &n);

for (int i = 0; i < n; i++) {
int a;
float b;
char op;
double c, result = 0.0;


scanf("%d %f %c %lf", &a, &b, &op, &c);


if (op == '+') {
result = a + b + c;
} else if (op == '-') {
result = a - b + c;
} else if (op == '*') {
result = a * b + c;
} else if (op == '/') {
if (b != 0.0) {
result = a / b + c;
} else {
result = c;
}
}


printf("%.2lf\n", result);
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
16.00
```

101.00

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

10.00

Compilation Status: Passed

Execution Time:

0.001s

2. Problem: Student Records ManagementObjective:

Write a C program to manage a set of student records using arrays, pointers, structures, and unions. The program should allow storing and displaying information about each student, including their grades and status.

Specifications:

Structures:

Please define a structure named Student with these following members:Name (Its an array of characters with a maximum length = 50)id (an integer)grades (an array of five floating-point numbers)average (a floating-point number)status (a union that can hold either a character to represent pass/fail or an integer for a grade category)Union:

The union status can either hold:char passFail ('P' for pass, 'F' for fail) if the average is greater than or equal to 50.int gradeCategory (1 for excellent, 2 for good, 3 for average, 4 for poor) based on the average score:Excellent (>= 85)Good (>= 70 and < 85)Average (>= 50 and < 70)Poor (< 50)Functions:

Implement the following functions:void inputStudentData(Student *s, int n) - to input data for n students.void calculateAverage(Student *s, int n) - to calculate the average grades for each student.void determineStatus(Student *s, int n) - to set the status of each student based on their average.void displayStudents(const Student *s, int n) - to display the information of each student.Input/Output:

Your code should first read an integer n = the number of students.Then, it should read the name, ID, and the five grades for each student.Calculate the average, determine

the status, and display the details of each student.ExampleInput:2Alice 101 75.0 82
90.0 70.0 88.0Bob 102 45.0 55.0 65.0 35.0 60.0

Output:Student: AliceID: 101Grades: 75.0, 82.0, 90.0, 70.0, 88.0Average: 81.0Status
Grade Category - 2

Student: BobID: 102Grades: 45.0, 55.0, 65.0, 35.0, 60.0Average: 52.0Status: Pass/Fail
- P

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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


typedef struct {
char name[50];
int id;
float grades[5];
float average;
union {
char passFail;
int gradeCategory;
} status;
} Student;


void inputStudentData(Student *s, int n) {
for (int i = 0; i < n; i++) {
scanf("%s", s[i].name);
scanf("%d", &s[i].id);
for (int j = 0; j < 5; j++) {
scanf("%f", &s[i].grades[j]);
}
}
}


void calculateAverage(Student *s, int n) {
for (int i = 0; i < n; i++) {
float sum = 0.0;
for (int j = 0; j < 5; j++) {
sum += s[i].grades[j];
}
s[i].average = sum / 5;
}
}


void determineStatus(Student *s, int n) {
for (int i = 0; i < n; i++) {
if (s[i].average >= 85) {
s[i].status.gradeCategory = 1;
} else if (s[i].average >= 70) {
s[i].status.gradeCategory = 2;
} else if (s[i].average >= 50) {
s[i].status.passFail = 'P';
} else {
s[i].status.passFail = 'F';
}
}
}


void displayStudents(const Student *s, int n) {
for (int i = 0; i < n; i++) {
printf("Student: %s\n", s[i].name);
printf("ID: %d\n", s[i].id);
printf("Grades: ");
for (int j = 0; j < 5; j++) {
printf("%.1f", s[i].grades[j]);
if (j < 4) printf(", ");
}
printf("\nAverage: %.1f\n", s[i].average);


if (s[i].average >= 70) {
printf("Status: Grade Category - %d\n", s[i].status.gradeCategory);
} else {
printf("Status: Pass/Fail - %c\n", s[i].status.passFail);
}
}
}


int main() {
int n;
scanf("%d", &n);

Student students[n];

inputStudentData(students, n);
calculateAverage(students, n);
determineStatus(students, n);
displayStudents(students, n);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Student: Alice
ID: 101
Grades: 85.0, 90.0, 78.0, 92.0, 88.0
Average: 86.6
Status: Grade Category - 1
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Student: Bob
ID: 102
Grades: 70.0, 72.0, 68.0, 75.0, 80.0
Average: 73.0
Status: Grade Category - 2
```

Compilation Status: Passed

Execution Time:

0.001s

3. Problem StatementYou are tasked with creating a simple financial application that can manage and display data for different types of transactions. The application should be able to handle three types of transactions: Deposit, Withdrawal, and Transfer. Each

transaction type has different attributes, and you will use a union t store the transaction data efficiently.

Transaction Types and Attributes:Deposit:Amount (float)Account Number (integer)Withdrawal:Amount (float)Account Number (integer)ATM Location (string)Transfer:Amount (float)From Account Number (integer)To Account Number (integer)

TaskWrite a program that takes user input to fill in the transaction details and then prints out the transaction information.

InputThis first line will contain a character representing the transaction type:D for DepositW for WithdrawalT for TransferThe subsequent lines contain the necessary transaction details based on the transaction type:For Deposit:A floating-point number representing the amount.An integer for the account number.For Withdrawal:A floating-point number representing the amount.An integer for the account number.A string for the ATM location.For Transfer:A floating-point number representing the amount.An integer for the from account number.An integer for the to account number.

OutputThe program should output the transaction details formatted appropriately, including the transaction type, amount, account numbers, and other relevant details.

Sample Input:D5000.0012345

Sample Output:Transaction Type: DepositAmount: 5000.00Account Number: 12345

ConstraintsThe amount will always be a positive float less than 1,000,000.Account numbers are the integers and they will be positive numbers less than 100,000.The ATM location string will not exceed 50 characters.Input should be validated to ensure correct transaction type.

ExplanationUse a union to store different transaction details. Based on the user's input, fill the appropriate fields in the union and display the transaction data. This approach helps in managing memory efficiently, as a union stores only one of the members at any given time.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

union TransactionDetails {
struct {
float amount;
```

```
int account_number;
} deposit;
struct {
float amount;
int account_number;
char atm_location[51];
} withdrawal;
struct {
float amount;
int from_account;
int to_account;
} transfer;
};

void handle_deposit(union TransactionDetails *transaction) {
printf("Transaction Type: Deposit\n");
printf("Amount: %.2f\n", transaction->deposit.amount);
printf("Account Number: %d\n", transaction->deposit.account_number);
}

void handle_withdrawal(union TransactionDetails *transaction) {
printf("Transaction Type: Withdrawal\n");
printf("Amount: %.2f\n", transaction->withdrawal.amount);
printf("Account Number: %d\n", transaction->withdrawal.account_number);
printf("ATM Location: %s\n", transaction->withdrawal.atm_location);
}

void handle_transfer(union TransactionDetails *transaction) {
printf("Transaction Type: Transfer\n");
printf("Amount: %.2f\n", transaction->transfer.amount);
printf("From Account Number: %d\n", transaction->transfer.from_account);
printf("To Account Number: %d\n", transaction->transfer.to_account);
}

int main() {
char transaction_type;
union TransactionDetails transaction;

scanf(" %c", &transaction_type);

if (transaction_type == 'D') {
scanf("%f %d", &transaction.deposit.amount, &transaction.deposit.account_number);
if (transaction.deposit.amount > 0 && transaction.deposit.amount < 1000000 &&
transaction.deposit.account_number > 0 && transaction.deposit.account_number <
100000) {
handle_deposit(&transaction);
} else {
printf("Invalid input for Deposit.\n");
}
} else if (transaction_type == 'W') {
scanf("%f %d", &transaction.withdrawal.amount,
&transaction.withdrawal.account_number);
getchar();
fgets(transaction.withdrawal.atm_location, 51, stdin);
```

```
transaction.withdrawal.atm_location[strcspn(transaction.withdrawal.atm_location,
"\n")] = '\0';

if (transaction.withdrawal.amount > 0 && transaction.withdrawal.amount < 100000
&&
transaction.withdrawal.account_number > 0 &&
transaction.withdrawal.account_number < 100000) {
handle_withdrawal(&transaction);
} else {
printf("Invalid input for Withdrawal.\n");
}
} else if (transaction_type == 'T') {
scanf("%f %d %d", &transaction.transfer.amount,
&transaction.transfer.from_account, &transaction.transfer.to_account);
if (transaction.transfer.amount > 0 && transaction.transfer.amount < 1000000 &&
transaction.transfer.from_account > 0 && transaction.transfer.from_account <
100000 &&
transaction.transfer.to_account > 0 && transaction.transfer.to_account < 100000) {
handle_transfer(&transaction);
} else {
printf("Invalid input for Transfer.\n");
}
} else {
printf("Invalid transaction type.\n");
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Transaction Type: Withdrawal
Amount: 200.00
Account Number: 67890
ATM Location: Main Street ATM
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Transaction Type: Transfer
Amount: 1000.00
From Account Number: 54321
To Account Number: 98765
```

Compilation Status: Passed

Execution Time:

0.001s

4. Problem StatementYou are required to write a C program to handle and manipulate different types of integers using type modifiers. Specifically, you need to write a program that will:

Read two integers from the user, one of type short and the other of type long long.Calculate and display the following:The sum of the two integers.The difference between the long long integer and the short integer.The product of the two integers.The result of dividing the long long integer by the short integer (consider integer division).InputTwo integers:a (short integer)b (long long integer)OutputThe sum of a and b.The difference between b and a.The product of a and b.The result of b divided by a (integer division).ConstraintsThe short integer a should be within the range of -32,768 to 32,767.The long long integer b should be within the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.The short integer a should not be zero for the division operation.ExplanationThe program demonstrates the use of different type modifiers (short and long long) and performs basic arithmetic operations. It will handle integer overflow cases by leveraging the large range of long long and provide meaningful outputs.

ExampleInput:510000000000

Output:Sum: 10000000005Difference: 9999999995Product: 50000000000Division: 2000000000

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
short a;
long long b;

scanf("%hd", &a);
scanf("%lld", &b);

if (a == 0) {
printf("Error\n");
return 1;
}

long long sum = a + b;
long long difference = b - a;
long long product = (long long)a * b;
long long division = b / a;

printf("Sum: %lld\n", sum);
printf("Difference: %lld\n", difference);
printf("Product: %lld\n", product);
printf("Division: %lld\n", division);

return 0;}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Sum: 123456789012347
Difference: 123456789012343
Product: 246913578024690
Division: 61728394506172
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Sum: -100000000000001
Difference: -99999999999999
Product: 100000000000000
Division: 100000000000000
```

Compilation Status: Passed

Execution Time:

0.001s

5. Problem Statement:You are tasked with writing a program that determines whether an inputted number is within the range of values for various type modifiers (short, long, signed, unsigned). The program should analyze the user input to determine if it can fit into any of the predefined C types with their associated ranges and then print which types the number can fit into.

Description:The C language provides type modifiers (short, long, signed, unsigned) that control the size and representation of variables. Depending on the input value, it may fit within certain ranges of these types. Your job is to create a program that takes an integer input and determines the type modifiers it can fit into based on its size.

The program must evaluate the number with respect to the ranges for:

short intlong intunsigned short intunsigned long intFor example, signed short int can represent values from -32,768 to 32,767, and unsigned short int can represent values from 0 to 65,535.

Input Format:A single integer input n is provided by the user.

Output Format:The program should output the types the number can fit into. For each type, print:Fits in short int (if the number fits)Fits in long int (if the number fits)Fits in unsigned short int (if the number fits)Fits in unsigned long int (if the number fits)If the number does not fit into any of these types, print Does not fit in any type.

Sample Input:12345

Sample Output:

Fits in short intFits in long intFits in unsigned short intFits in unsigned long int

Constraints:The number can be as large as a long long int, i.e., in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
long long n;
scanf("%lld", &n);

int fits_short = 0, fits_long = 0, fits_unsigned_short = 0, fits_unsigned_long = 0;

if (n >= SHRT_MIN && n <= SHRT_MAX) {
printf("Fits in short int\n");
fits_short = 1;
}
if (n >= LONG_MIN && n <= LONG_MAX) {
printf("Fits in long int\n");
fits_long = 1;
}
if (n >= 0 && n <= USHRT_MAX) {
printf("Fits in unsigned short int\n");
fits_unsigned_short = 1;
}
if (n >= 0 && n <= ULONG_MAX) {
printf("Fits in unsigned long int\n");
fits_unsigned_long = 1;
}

if (!(fits_short || fits_long || fits_unsigned_short || fits_unsigned_long)) {
printf("Does not fit in any type\n");
} else if (!fits_short || !fits_long || !fits_unsigned_short || !fits_unsigned_long) {
printf("Does not fit in any type\n");
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Fits in short int
Fits in long int
Fits in unsigned short int
Fits in unsigned long int
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Fits in long int
Does not fit in any type
```

Compilation Status: Passed

Execution Time:

0.001s

6. Problem StatementWrite a C program that manages and analyzes a dynamically sized array of floating-point numbers. The program should:

Take(read) the size of the array from the user.Declare and initialize an array of the specified size.Populate(fill) the array with user-input floating-point numbers.Perform these following operations and display the results:Calculate the average of the numbers.Compute the maximum and minimum numbers present in the array.Compute the standard deviation of the numbers.

Input:An integer n (size of the array, where 1 <= n <= 100).n floating-point numbers.

Output:The average of the numbers with two decimal precision.The maximum number in the array.The minimum number in the array.The standard deviation of the numbers with two decimal precision.

Constraints:The array size n must be between 1 and 100.The input floating-point numbers should be in the range of -1e6 to 1e6.The calculations should be done with precision.ExplanationThis problem emphasizes the understanding of variable declaration and initialization, including handling user input for dynamically sized data. The program should efficiently handle floating-point arithmetic and use appropriate data types and methods to ensure accuracy and precision.

ExampleInput:510.5 20.8 30.1 40.2 50.6

Output:Average: 30.44Maximum: 50.60Minimum: 10.50Standard Deviation: 14.09

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

float custom_sqrt(float number) {
if (number <= 0) return 0;
float guess = number / 2.0, epsilon = 0.00001;

while ((guess * guess - number) > epsilon || (number - guess * guess) > epsilon) {
guess = (guess + number / guess) / 2.0;
}
return guess;
}

int main() {
int n;
scanf("%d", &n);

if (n < 1 || n > 100) {
return 1;
}

float arr[n], sum = 0.0, max, min, mean, stddev = 0.0;

for (int i = 0; i < n; i++) {
scanf("%f", &arr[i]);
sum += arr[i];
if (i == 0) {
max = min = arr[i];
} else {
if (arr[i] > max) max = arr[i];
if (arr[i] < min) min = arr[i];
}
}

mean = sum / n;

for (int i = 0; i < n; i++) {
float diff = arr[i] - mean;
stddev += diff * diff;
}
stddev = custom_sqrt(stddev / n);

printf("Average: %.2f\n", mean);
printf("Maximum: %.2f\n", max);
printf("Minimum: %.2f\n", min);
printf("Standard Deviation: %.2f\n", stddev);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Average: 2.00
Maximum: 3.00
Minimum: 1.00
Standard Deviation: 0.82
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Average: 10.00
Maximum: 30.00
Minimum: -10.00
Standard Deviation: 14.14
```

Compilation Status: Passed

Execution Time:

0.001s

7. Problem Statement:Write a C program to simulate a simplified banking system that manages transactions for multiple accounts. The program should demonstrate the use of variable scope and lifetime by:

Declaring a global variable to keep track of the total number of transactions across all accounts. Declaring static variables within functions to maintain the balance of each account. Using local variables to handle individual transaction details. The program should perform the following tasks:

Initialize a given number of accounts with starting balances. Allow the user to perform a sequence of transactions (deposit or withdraw) on any account. Display the balance of an account after each transaction. At the end, display the total number of transactions made.

InputAn integer n (number of accounts, where 1 <= n <= 10). n initial account balances as floating-point numbers. An integer t (number of transactions, where 1 <= t <= 50). For each transaction: An integer account_number (1 to n) indicating the target account. A character type ('D' for deposit, 'W' for withdrawal). A floating-point number amount (transaction amount).

OutputThe balance of the specified account after each transaction. The total number of transactions performed. Constraints The number of accounts n must be between 1 and 10. The number of transactions t must be between 1 and 50. Transaction amounts should be in the range of 0.01 to 1,000,000.00. Withdrawals should not exceed the current balance of the account. Accounts are 1-indexed (account numbers start from 1).

Explanation This problem is designed to test understanding of variable scope and lifetime in C, particularly:

Global Variables: Used to maintain data that should be accessible across different functions. Static Local Variables: Used to maintain state within a function, persisting across multiple calls. Local Variables: Used for temporary data that is limited to the function scope.

Example Input: 3 1000.00 2000.00 3000.00 5 1 D 500.00 2 W 250.00 3 D 1000.00 1 W 750.00 3 W 1500.00 Output: Account 1 Balance: 1500.00 Account 2 Balance: 1750.00 Account 3 Balance: 4000.00 Account 1 Balance: 750.00 Account 3 Balance: 2500.00 Total Transactions: 5

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

// Global variable to track the total number of transactions
int total_transactions = 0;

// Function to manage account transactions
void process_transaction(int account_number, char type, float amount, float
balances[],
int num_accounts) {
// Check if the account number is valid
if (account_number < 1 || account_number > num_accounts) {
return; // Ignore invalid account numbers
}

// Index of the account in the balances array
int idx = account_number - 1;

// Process the transaction
if (type == 'D') {
// Deposit
balances[idx] += amount;
total_transactions++;
printf("Account %d Balance: %.2f\n", account_number, balances[idx]);
} else if (type == 'W') {
// Withdrawal
if (amount > balances[idx]) {
return; // Ignore invalid withdrawals
} else {
balances[idx] -= amount;
total_transactions++;
printf("Account %d Balance: %.2f\n", account_number, balances[idx]);
}
}
}

int main() {
int n, t;

// Input the number of accounts
if (scanf("%d", &n) != 1 || n < 1 || n > 10) {
return 1; // Exit for invalid number of accounts
}
float balances[n];

// Input initial balances
for (int i = 0; i < n; i++) {
if (scanf("%f", &balances[i]) != 1) {
return 1; // Exit if invalid input for balances
}
}

// Input the number of transactions
if (scanf("%d", &t) != 1 || t < 1 || t > 50) {
return 1; // Exit for invalid number of transactions
}

// Process each transaction
for (int i = 0; i < t; i++) {
int account_number;
char type;
float amount;

// Input transaction details
if (scanf("%d %c %f", &account_number, &type, &amount) != 3) {
return 1; // Exit if transaction details are invalid
}

if (amount < 0.01 || amount > 1000000.00) {
continue; // Skip invalid transaction amounts
}

// Process the transaction
process_transaction(account_number, type, amount, balances, n);
}

// Output the total number of transactions
printf("Total Transactions: %d\n", total_transactions);

return 0;}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Account 1 Balance: 6500.00
Account 2 Balance: 2000.00
Account 1 Balance: 4500.00
Total Transactions: 3
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Account 3 Balance: 1700.00
Account 4 Balance: 500.00
Account 1 Balance: 4000.00
Account 2 Balance: 2500.00
Total Transactions: 4
```

Compilation Status: Passed

Execution Time:

0.001s

8. Problem StatementYou need to develop a program that processes and analyzes large datasets of integers. Your task is to manage data using arrays and pointers and perform operations like sorting, searching, and finding the median.

RequirementsData Input: There will be an Input of the array of integers dynamically.Sorting: Sort the array using the quicksort algorithm with pointers.Searching: Implementing the binary search methodology using pointers to find a specific element.Median Calculation: Calculate the median of the sorted array using pointers.

Input FormatThe first line will contain an integer n, such that 1 <= n <= 100000, representing the number of elements in the array.The second line contains n space-separated integers representing the array elements.The third line contains an integer x, representing the element to search in the array.

Output FormatPrint the sorted array of integers on a single line, space-separated.Print the median value of the array on the next line.Print the result of the binary search on the next line: the index of the element x if found, otherwise -1.

ConstraintsAll integers are within the range -1000000 to 1000000.Efficient use of pointers for sorting, searching, and calculating the median.Consider both odd and even lengths for median calculation.

ExampleInput:712 4 5 3 8 7 15

Output:1 3 4 5 7 8 125.003

Explanation:Sorting: The sorted array is [1, 3, 4, 5, 7, 8, 12].Median: The median value of this sorted array is 5 since it is the middle element.Binary Search: The element 5 is found at index 3 (0-based index) in the sorted array.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void quicksort(int *arr, int low, int high) {
if (low < high) {
int pivot = arr[high];
int i = low - 1, j, temp;
for (j = low; j < high; j++) {
if (arr[j] < pivot) {
i++;
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
}
temp = arr[i + 1];
arr[i + 1] = arr[high];
arr[high] = temp;
int pi = i + 1;
quicksort(arr, low, pi - 1);
quicksort(arr, pi + 1, high);
}
}

int binary_search(int *arr, int n, int x) {
int left = 0, right = n - 1, mid;
while (left <= right) {
mid = left + (right - left) / 2;
if (arr[mid] == x)
return mid;
else if (arr[mid] < x)
left = mid + 1;
else
right = mid - 1;
}
return -1;
}

float find_median(int *arr, int n) {
if (n % 2 == 0)
return (arr[n / 2 - 1] + arr[n / 2]) / 2.0;
else
return arr[n / 2];
}

int main() {
int n, x, i;
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
scanf("%d", &x);
quicksort(arr, 0, n - 1);
for (i = 0; i < n; i++)
printf("%d ", arr[i]);
printf("\n");
printf("%.2f\n", find_median(arr, n));
printf("%d\n", binary_search(arr, n, x));
free(arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
1 1 3 4 5
3.00
3
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
3 5 7 8 9 10
7.50
3
```

Compilation Status: Passed

Execution Time:

0.001s

9. Problem StatementWrite a C program to manage student records for a class. Each student has a name, roll number, marks in three subjects, and their calculated total and average marks. The program should:

Declare a structure to hold a student’s information.Read student data from the user.Compute the total and average marks for each student.Display the student information along with total and average marks in a sorted order based on the average marks in descending order.

InputAn integer n (number of students, where 1 ≤ n ≤ 100).For each student:A string name (up to 50 characters).An integer roll_number (1 to 1000).Three floating-point numbers representing marks in three subjects.

OutputStudent information (name, roll number, total marks, average marks) sorted by average marks in descending order.ConstraintsThe number of students n must be between 1 and 100.Roll numbers are unique integers between 1 and 1000.Marks should be in the range of 0.0 to 100.0.Use appropriate data types to store student information and calculate totals and averages with precision.

ExplanationThis problem basically emphasizes the understanding of the structures concept in C programming. It requires you to:

Declare and use a structure to represent student data.Manage data entry and computations using the structure.Implement sorting logic based on calculated averages.

ExampleInput:1John Doe 101859278

Output:Name: John Doe , Roll Number: 101, Total Marks: 255.00, Average Marks: 85.00

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
struct Student {
char name[51];
int roll_number;
float marks[3];
float total_marks;
float average_marks;
};

void calculate_total_and_average(struct Student *s) {
s->total_marks = s->marks[0] + s->marks[1] + s->marks[2];
s->average_marks = s->total_marks / 3.0;
}

void sort_students(struct Student *students, int n) {
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (students[j].average_marks < students[j + 1].average_marks) {
struct Student temp = students[j];
students[j] = students[j + 1];
students[j + 1] = temp;
}
}
}
}

int main() {
int n;
scanf("%d", &n);

struct Student students[100];
for (int i = 0; i < n; i++) {
scanf(" %[^\n]", students[i].name);
scanf("%d", &students[i].roll_number);
for (int j = 0; j < 3; j++) {
scanf("%f", &students[i].marks[j]);
}
calculate_total_and_average(&students[i]);
}

sort_students(students, n);
for (int i = 0; i < n; i++) {
printf("Name: %s, Roll Number: %d, Total Marks: %.2f, Average Marks: %.2f\n",
students[i].name, students[i].roll_number, students[i].total_marks,
students[i].average_marks);
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Name: John Doe , Roll Number: 101, Total Marks: 255.00, Average Marks: 85.00
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Name: Alex Johnson, Roll Number: 201, Total Marks: 268.50, Average Marks: 89.50
Name: Betty White , Roll Number: 202, Total Marks: 246.50, Average Marks: 82.17
Name: Charlie Brown , Roll Number: 203, Total Marks: 210.00, Average Marks: 70.00
```

Compilation Status: Passed

Execution Time:

0.001s

10. Problem StatementTitle: Permission Control System for a Sma Home

Scenario-Based Problem:Imagine you're developing a permission control system fo smart home where various devices (like lights, locks, thermostats, and cameras) are controlled based on user permissions. Each device has a specific bit in a control register (an integer) that determines whether a user has permission to control that device.

Given two users, you need to implement a system that:

Grants control over specific devices to a user.Revokes control over specific devices.Checks whether a user has control over a specific device.Combines permissions from two users to create a third user's control register that represents the combined permissions.Displays the control register in binary format to see which devices are controllable by the user.

Devices:

Bit 0: LightsBit 1: LocksBit 2: ThermostatBit 3: CamerasBit 4: Garage DoorBit 5: SprinklersBit 6: Alarm SystemBit 7: Music System

Input:Two integers representing the control registers of two users.Operations that include granting or revoking permissions, checking control, combining permissions, and displaying the control register.

Output:Results of the operations including the updated control registers and binary representation.

Sample Input:170 85Operation: Grant Lights to User AOperation: Revoke Music System from User B

Sample Output:User A Control Register after Granting Lights: 171User B Control Register after Revoking Music System: 85User A has control over Cameras: YesCombined Control Register (User C): 255User A Control Register in Binary: 10101011

Constraints:The control register should be an 8-bit integer (values between 0 and 255).Operations should be performed using bitwise operators.Ensure that the program correctly handles edge cases like no permissions (0) and full permissions (255).

Example:Assume user A has the control register 10101010 (170 in decimal) and user B has the control register 01010101 (85 in decimal).

Grant the Lights control to User A.Revoke the Music System control from User B.Check if User A has control over the Cameras.Combine User A and User B's control registers to create User C's control register.Display User C's control register in binary format.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void display_binary(int control_register) {
for (int i = 7; i >= 0; i--) {
printf("%d", (control_register >> i) & 1);
}
printf("\n");
}

int main() {
int userA, userB, combined;

scanf("%d %d", &userA, &userB);

userA |= (1 << 0);
printf("User A Control Register after Granting Lights: %d\n", userA);

userB &= ~(1 << 7);
printf("User B Control Register after Revoking Music System: %d\n", userB);

if ((userA >> 3) & 1) {
printf("User A has control over Cameras: Yes\n");
} else {
printf("User A has control over Cameras: No\n");
}

combined = userA | userB;
printf("Combined Control Register (User C): %d\n", combined);
printf("User A Control Register in Binary: ");
display_binary(userA);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
User A Control Register after Granting Lights: 171
User B Control Register after Revoking Music System: 85
User A has control over Cameras: Yes
Combined Control Register (User C): 255
User A Control Register in Binary: 10101011
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
User A Control Register after Granting Lights: 35
User B Control Register after Revoking Music System: 12
User A has control over Cameras: No
Combined Control Register (User C): 47
User A Control Register in Binary: 00100011
```

Compilation Status: Passed

Execution Time:

0.001s

11. Problem StatementTitle: Network Packet Manipulation

Scenario-Based Problem:In a specific network communication system, the data is transmitted in the form of packets. Each packet contains metadata in the form of an 8-bit header that encodes various properties such as packet type, priority, encryption status, and error flags. The bits in the header have the following significance:

Bit 0: Packet Type (0 for data, 1 for control)Bit 1: Priority (0 for low, 1 for high)Bit 2: Encryption Status (0 for unencrypted, 1 for encrypted)Bit 3: Error Detected (0 for no error, 1 for error)Bits 4-7: Reserved for future use

You are tasked with writing a program to manipulate this packet header using bitwise operators. The program should perform the following operations based on user input:

Toggle the Packet Type.Set the Priority to High.Clear the Encryption Status (set to unencrypted).Check if an error is detected.Shift the header left by 1 bit and check the new value.

Input:

An integer representing the 8-bit packet header.Operations specified by the user.

Output:

The resulting packet header after performing the specified operations, both in decimal and binary format.

Sample Input:210Operation: Toggle Packet TypeOperation: Set Priority to High

Sample Output:New Header in Decimal: 211New Header in Binary: 11010011New Header after clearing encryption status: 211New Header in Binary: 11010011Error Detected: NoHeader after Left Shift by 1 Bit: 422New Header in Binary: 10100110

Constraints:

The header of the packet is an 8-bit integer (which values between 0 and 255).Operations must be performed using bitwise operators.The program should handle edge cases where all bits are set to 0 or 1.

Example:

Assume the initial packet header is 11010010 (210 in decimal).

Step 1: Toggle the Packet Type.The initial header is 11010010. Toggling the Packet Type (Bit 0) flips the first bit, changing the header to 11010011 (211 in decimal).Step 2: Set the Priority to High.The current header is 11010011. Setting the Priority to High (Bit 1) ensures the second bit is set to 1, but it is already 1, so the header remains 11010011 (211 in decimal).Step 3: Clear the Encryption Status.The current header is 11010011. Clearing the Encryption Status (Bit 2) sets the third bit to 0, changing the header to 11010001 (209 in decimal).Step 4: Check if an error is detected.The current header is 11010001. Checking if an error is detected (Bit 3) shows that the fourth bit is 1, indicating an error is present.Step 5: Shift the header left by 1 bit.The current header is 11010001. Shifting the header left by 1 bit shifts all bits left, resulting in 10100010 (162 in decimal).

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void display_binary(int header) {
for (int i = 7; i >= 0; i--) {
printf("%d", (header >> i) & 1);
}
printf("\n");
}

int main() {
```

```
int header;
scanf("%d", &header);

header ^= (1 << 0);
header |= (1 << 1);

printf("New Header in Decimal: %d\n", header);
printf("New Header in Binary: ");
display_binary(header);

header &= ~(1 << 2);
printf("New Header after clearing encryption status: %d\n", header);
printf("New Header in Binary: ");
display_binary(header);

if ((header >> 3) & 1) {
printf("Error Detected: Yes\n");
} else {
printf("Error Detected: No\n");
}

int shifted_header = header << 1;
printf("Header after Left Shift by 1 Bit: %d\n", shifted_header);
printf("New Header in Binary: ");
display_binary(shifted_header);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
New Header in Decimal: 211
New Header in Binary: 11010011
New Header after clearing encryption status: 211
New Header in Binary: 11010011
Error Detected: No
Header after Left Shift by 1 Bit: 422
New Header in Binary: 10100110
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
New Header in Decimal: 211
New Header in Binary: 11010011
New Header after clearing encryption status: 211
New Header in Binary: 11010011
Error Detected: No
Header after Left Shift by 1 Bit: 422
New Header in Binary: 10100110
```

Compilation Status: Passed

Execution Time:

0.001s

12. Problem StatementTitle: Bank Account Transaction Manager

Scenario-Based Problem:

You are tasked with developing a program for a bank to manage multiple transactions on a customer's account. The bank account holds an integer balance, and transactions are applied sequentially using various assignment operators. Each transaction can be a deposit, withdrawal, interest application, fee deduction, or investment. The operations to be performed on the balance are specified by the customer.

The assignment operators involved include:

=: Assign a new balance.+=: Deposit an amount.-=: Withdraw an amount.*=: Apply a multiplication factor (e.g., interest rate)./=: Apply a division factor (e.g., service fee). %=: Apply a modulo operation (e.g., remainder after dividing by a certain number).&=: Apply a bitwise AND operation.|=: Apply a bitwise OR operation.^=: Apply a bitwise XOR operation.<<=: Apply a left shift operation (e.g., doubling balance).>>=: Apply a right shift operation (e.g., halving balance).The program should read an initial balance and a sequence of transactions, apply the operations accordingly, and output the final balance after all transactions are processed.

Input:

An integer representing the initial balance.A series of operations and values, each specifying a transaction to be applied.

Output:

The final balance after all transactions have been applied.

Sample Input:1000+= 500-= 200*= 1.10/= 2q

Sample Output:Final Balance: 715

Constraints:

The balance should always be a positive integer and can only go to zero if all funds are withdrawn or due to bitwise operations.The operations and values provided must be valid and within the allowable integer range for 32-bit signed integers.The program should handle edge cases like division by zero, large multiplications, and overflow scenarios.

Example:

Assume the initial balance is 1000.

Transaction 1: Deposit 500 (Using +=)1000 += 500 New Balance: 1500Transaction 2: Withdraw 200 (Using -=)1500 -= 200 New Balance: 1300Transaction 3: Apply Interest of 10% (Using *=)1300 *= 1.10 New Balance: 1430Transaction 4: Apply a Service Fee Factor of 2 (Using /=)1430 /= 2 New Balance: 715Transaction 5: Apply a Bitwise AND with 512 (Using &=)715 &= 512 New Balance: 512The final balance after all operations is 512.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
double balance;
char operation[3];
double value;

scanf("%lf", &balance);

while(1) {
scanf("%s", operation);

if(operation[0] == 'q')
break;

switch(operation[0]) {
```

```
case '+':
scanf("%lf", &value);
balance += value;
break;
case '-':
scanf("%lf", &value);
balance -= value;
break;
case '*':
scanf("%lf", &value);
balance *= value;  // Apply multiplication with floating-point value
break;
case '/':
scanf("%lf", &value);
if(value != 0)
balance /= value;
break;
case '%':
scanf("%lf", &value);
balance = (int)balance % (int)value;
break;
case '&':
scanf("%lf", &value);
balance = (int)balance & (int)value;
break;
case '|':
scanf("%lf", &value);
balance = (int)balance | (int)value;
break;
case '^':
scanf("%lf", &value);
balance = (int)balance ^ (int)value;
break;
case '<':
balance = (int)balance << 1;
break;
case '>':
balance = (int)balance >> 1;
break;
}
}

printf("Final Balance: %.0f\n", balance);  // Print balance as integer by rounding
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Final Balance: 715
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Final Balance: 24
```

Compilation Status: Passed

Execution Time:

0.001s

13. Problem StatementTitle: Financial Transaction System with Precise Calculations

Problem Description:You are developing a financial transaction system for a bank. The system must handle transactions involving different types of accounts, which store balances as different data types. The system needs to accurately calculate the total balance across all accounts, considering both implicit and explicit type conversions to avoid any loss of precision.

Scenario:The bank has three types of accounts:

Savings Account: Balance is stored as an int.Current Account: Balance is stored as a float.Fixed Deposit Account: Balance is stored as a double.Your task is to calculate the total balance of a customer by summing the balances from all three types of accounts. Due to the differences in data types, you need to ensure that conversions are handled properly to maintain precision. Use implicit and explicit type casting where necessary.

Input:An integer savings_balance representing the balance in the savings account.A float current_balance representing the balance in the current account.A double fixed_deposit_balance representing the balance in the fixed deposit account.

Output:A double value representing the total balance across all accounts, calculate with the highest precision.Constraints:

The balances are always positive and non-zero.savings_balance is an integer in the range [1, 1,000,000].current_balance is a float in the range [0.01, 1,000,000.00].fixed_deposit_balance is a double in the range [0.01, 10,000,000.00].You must use explicit type casting where necessary to ensure that the total balance is calculated correctly with the highest precision.

Explanation:The savings balance should be implicitly converted to float or double for precise calculation.Explicit type casting might be required to ensure the correct summation of different data types.The output should maintain the precision of the double type.

Sample Input :100000050000.502500000.75

Sample Output :Total Balance: 3550001.25

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int savings_balance;
float current_balance;
double fixed_deposit_balance;

scanf("%d", &savings_balance);
scanf("%f", &current_balance);
scanf("%lf", &fixed_deposit_balance);

double total_balance = savings_balance + (double)current_balance +
fixed_deposit_balance;

printf("Total Balance: %.2lf\n", total_balance);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Total Balance: 3550001.25
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Total Balance: 1151235.34
```

Compilation Status: Passed

Execution Time:

0.001s

14. Problem StatementComplex Budget Calculator for Product Manufacturing

You are tasked with designing a program to assist a manufacturing company in calculating the overall production cost, profit margins, and tax deductions for multiple products. The program should allow the user to input the cost of raw materials, labor, and overhead for each product. The program should then compute the following:

Total Production Cost: Calculated as the sum of raw material cost, labor cost, and overhead cost.Gross Profit: Calculated as 20% of the total production cost.Tax Deduction: If the gross profit is greater than ₹100,000, a 15% tax is applied; otherwise, a 10% tax is applied.Net Profit: Calculated as gross profit minus tax deduction.Product Status:If the net profit is greater than or equal to ₹75,000, the product is considered "Highly Profitable".If the net profit is between ₹50,000 and ₹74,999, it is considered "Moderately Profitable".If the net profit is less than ₹50,000, it is considered "Less Profitable".

Constraints:Input values should be positive integers.The number of products (n) is between 1 and 100.The cost values for raw materials, labor, and overhead must be the range [₹1,000, ₹10,00,000].Use appropriate type casting where necessary to ensure correct calculations, particularly with floating-point arithmetic.

Input:The first input is an integer n representing the number of products.For each product, three integer inputs representing the cost of raw materials, labor, and overhead.

Output:For each product, output the total production cost, gross profit, tax deduction, net profit, and product status.

Example:Sample Input:3250000 150000 50000100000 80000 20000500000 300000 100000

Sample Output:Product 1:Total Production Cost: ₹450000Gross Profit: ₹90000.00Tax Deduction: ₹9000.00Net Profit: ₹81000.00Product Status: Highly ProfitableProduct 2:Total Production Cost: ₹200000Gross Profit: ₹40000.00Tax Deduction: ₹4000.00Net Profit: ₹36000.00Product Status: Less ProfitableProduct 3:Total Production Cost: ₹900000Gross Profit: ₹180000.00Tax Deduction: ₹27000.00Net Profit: ₹153000.00Product Status: Highly Profitable

Explanation:Product 1: The gross profit is ₹90,000 (20% of ₹450,000). A 10% tax is applied (since gross profit is below ₹100,000), resulting in ₹81,000 net profit. It is "Moderately Profitable."Product 2: The gross profit is ₹40,000 (20% of ₹200,000). A 10% tax is applied, resulting in ₹36,000 net profit. It is "Less Profitable."Product 3: The gross profit is ₹180,000 (20% of ₹900,000). A 15% tax is applied, resulting in ₹153,000 net profit. It is "Highly Profitable."

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n;
scanf("%d", &n);

for(int i = 1; i <= n; i++) {
int raw_material_cost, labor_cost, overhead_cost;
scanf("%d %d %d", &raw_material_cost, &labor_cost, &overhead_cost);

int total_cost = raw_material_cost + labor_cost + overhead_cost;
float gross_profit = total_cost * 0.20;
float tax_deduction = (gross_profit > 100000) ? gross_profit * 0.15 : gross_profit *
0.10;
float net_profit = gross_profit - tax_deduction;
printf("Product %d:\n", i);
printf("Total Production Cost: ₹%d\n", total_cost);
printf("Gross Profit: ₹%.2f\n", gross_profit);
printf("Tax Deduction: ₹%.2f\n", tax_deduction);
printf("Net Profit: ₹%.2f\n", net_profit);

if (net_profit >= 75000) {
printf("Product Status: Highly Profitable\n");
} else if (net_profit >= 50000) {
printf("Product Status: Moderately Profitable\n");
} else {
printf("Product Status: Less Profitable\n");
}
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Product 1:
Total Production Cost: ₹1000000
Gross Profit: ₹200000.00
Tax Deduction: ₹30000.00
Net Profit: ₹170000.00
Product Status: Highly Profitable
Product 2:
Total Production Cost: ₹300000
Gross Profit: ₹60000.00
Tax Deduction: ₹6000.00
Net Profit: ₹54000.00
Product Status: Moderately Profitable
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Product 1:
Total Production Cost: ₹450000
Gross Profit: ₹90000.00
Tax Deduction: ₹9000.00
Net Profit: ₹81000.00
Product Status: Highly Profitable
```

Compilation Status: Passed

Execution Time:

0.001s

15. Problem StatementTitle: Emergency Response Traffic Light System

You are required to program a traffic light system that can handle emergency vehicle scenarios at a busy intersection. The traffic light can be: Red, Yellow, or Green. Depending on the state of the light and whether an emergency vehicle is approaching, the system should determine the appropriate action for the vehicles at the intersection.

The rules for determining the action are as follows:

Red Light:

If the light is red and an emergency vehicle is approaching, vehicles must allow it to pass by preparing to go.If no emergency vehicle is approaching, vehicles must stop.Yellow Light:

If the light is yellow, vehicles must prepare to stop.If the light is yellow and an emergency vehicle is approaching, vehicles must continue to allow the emergency vehicle to pass.Green Light:

If the light is green, vehicles should proceed normally.If the light is green and an emergency vehicle is approaching, vehicles must clear the way by pulling over.Based on the above rules, the program should output the action that vehicles should take.

Input:

The first line contains a single character, T, representing the traffic light's current state. T can be 'R' for Red, 'Y' for Yellow, or 'G' for Green.The second line contains a single integer, E, where E is 1 if an emergency vehicle is approaching, and 0 if there is no emergency vehicle.

Output:

Print the action vehicles should take: "ALLOW EMERGENCY", "STOP", "PREPARE TO STOP", "PROCEED", or "CLEAR THE WAY".

Sample Input:R1

Sample Output:ALLOW EMERGENCY

Constraints:

T is always one of the characters 'R', 'Y', 'G'.E is always either 0 or 1.

ExplanationConsider a sample input to illustrate how the system determines the appropriate action:R1

Here, T = 'R' (Red Light) and E = 1 (An emergency vehicle is approaching).

Rule 1 (Red Light): Since an emergency vehicle is approaching, vehicles should prepare to go and allow the emergency vehicle to pass.Result: The output should be "ALLOW EMERGENCY".

ALLOW EMERGENCY

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
char T;
int E;

scanf("%c", &T);
scanf("%d", &E);

if (T == 'R') {
if (E == 1) {
printf("ALLOW EMERGENCY\n");
} else {
printf("STOP\n");
}
} else if (T == 'Y') {
if (E == 1) {
printf("ALLOW EMERGENCY\n");
} else {
printf("PREPARE TO STOP\n");
}
} else if (T == 'G') {
if (E == 1) {
printf("CLEAR THE WAY\n");
} else {
printf("PROCEED\n");
}
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
ALLOW EMERGENCY
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
PREPARE TO STOP
```

Compilation Status: Passed

Execution Time:

0.001s

16. Problem Statement:Scenario:You are tasked with developing a menu-driven program for a small library management system. The program should allow users to perform various operations such as adding a book, searching for a book by its ID, deleting a book by its ID, displaying all books, and exiting the program.

Input:The program will repeatedly display a menu with the following options:

Add a bookSearch for a book by IDDelete a book by IDDisplay all booksExitFor Option 1 (Add a book): The user will input a book ID (integer) and a book title (string).For Option 2 (Search for a book by ID): The user will input the book ID to search for.For Option 3 (Delete a book by ID): The user will input the book ID to delete.For Option 4 (Display all books): The program will display all the books currently in the system.For Option 5 (Exit): The program will terminate.

Output:

For Option 1: The program should confirm that the book was added.For Option 2: The program should display the book details if found or notify the user if the book is not found.For Option 3: The program should confirm that the book was deleted or notify the user if the book ID is not found.For Option 4: The program should list all books in the system.For Option 5: The program will exit with a message.

Sample Input:1101C Programming Language1102Data Structures in C2101431025

Sample Output:Book added successfully.Book added successfully.Book found: ID=101, Title="C Programming Language"Book ID=101, Title="C Programming Language"Book ID=102, Title="Data Structures in C"Book deleted successfully.Exiting the program...

Constraints:

The system can hold a maximum of 100 books.The book ID must be unique.The title should not exceed 50 characters.Input validation should be performed for book ID to ensure it is a positive integer.

Example:Input:1. Add a book2. Search for a book by ID3. Delete a book by ID4. Display all books5. Exit

Choose an option: 1Enter Book ID: 101Enter Book Title: "C Programming Language"

Choose an option: 1Enter Book ID: 102Enter Book Title: "Data Structures in C"

Choose an option: 2Enter Book ID: 101

Choose an option: 4

Choose an option: 3Enter Book ID: 102

Choose an option: 5

Output:Book added successfully.Book added successfully.Book found: ID=101, Title="C Programming Language"Book ID=101, Title="C Programming Language"Book deleted successfully.Exiting the program...

Explanation:

The user adds two books with IDs 101 and 102. Program usually confirms the addition of each book.The user searches for the book with ID 101, and the program displays its details.The user displays all books, and the program lists the available book.The user deletes the book with ID 102, and the program confirms the deletion.The user exits the program.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
#define MAX_BOOKS 100
#define MAX_TITLE_LENGTH 50
typedef struct {
int id;
char title[MAX_TITLE_LENGTH];
} Book;
Book library[MAX_BOOKS];
int bookCount = 0;
void addBook() {
if (bookCount >= MAX_BOOKS) {
return;
}

int id;
char title[MAX_TITLE_LENGTH];

scanf("%d", &id);
getchar();
fgets(title, MAX_TITLE_LENGTH, stdin);
title[strcspn(title, "\n")] = '\0';

library[bookCount].id = id;
strcpy(library[bookCount].title, title);
bookCount++;

printf("Book added successfully.\n");
}

void searchBook() {
int id;
scanf("%d", &id);

for (int i = 0; i < bookCount; i++) {
```

```
if (library[i].id == id) {
printf("Book found: ID=%d, Title=\"%s\"\n", library[i].id, library[i].title);
return;
}
}
printf("Book not found.\n");
}

void deleteBook() {
int id;
scanf("%d", &id);

for (int i = 0; i < bookCount; i++) {
if (library[i].id == id) {
for (int j = i; j < bookCount - 1; j++) {
library[j] = library[j + 1];
}
bookCount--;
printf("Book deleted successfully.\n");
return;
}
}
printf("Book not found.\n");
}

void displayBooks() {
if (bookCount == 0) {
printf("No books in the library.\n");
return;
}
for (int i = 0; i < bookCount; i++) {
printf("Book ID=%d, Title=\"%s\"\n", library[i].id, library[i].title);
}
}

int main() {
int choice;

while (1) {
scanf("%d", &choice);

switch (choice) {
case 1:
addBook();
break;
case 2:
searchBook();
break;
case 3:
deleteBook();
break;
case 4:
displayBooks();
break;
case 5:
printf("Exiting the program...\n");
return 0;
}
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Book added successfully.
Book added successfully.
Book found: ID=101, Title="C Programming Language"
Book ID=101, Title="C Programming Language"
Book ID=102, Title="Data Structures in C"
Book deleted successfully.
Exiting the program...
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Book added successfully.
Book deleted successfully.
No books in the library.
Exiting the program...
```

Compilation Status: Passed

Execution Time:

0.001s

17. Problem StatementYou are developing a system for a hotel to manage room bookings over a month. The hotel has different rooms, each with a different nightly rate. Guests can book a room for a range of dates, and your system needs to calculate the total cost of the stay for each guest. You are tasked with handling the booking process and calculating the total cost using loops and conditional statements.

Each booking request consists of:

The room number (R) the guest wants to book.The start date (S) and end date (E) of their stay.Each room has a rate for the night that depends on the room number.You need to ensure that:

A booking is only valid if the start date is before or equal to the end date.The system should calculate the total cost by looping through each night of the guest's stay and summing the nightly rate for that room.If a guest's booking is invalid, the system should print "Invalid booking" and skip that request.Your program must handle multiple booking requests.

InputThe first line contains an integer N representing the number of booking requests.The next N lines contain three integers each: R (the room number), S (the start date of the stay), and E (the end date of the stay).

OutputFor each valid booking request, output the total cost of the stay for the guest.For each invalid booking request, output "Invalid booking".

Constraints1 ≤ N ≤ 1001 ≤ R ≤ 101 ≤ S, E ≤ 31Nightly rates for rooms:Room 1: $100Room 2: $150Room 3: $200Room 4: $250Room 5: $300Room 6: $350Room 7: $400Room 8: $450Room 9: $500Room 10: $550

Sample Input:31 5 102 12 153 20 18

Sample Output:600600Invalid booking

ExplanationThe first request is to book Room 1 from day 5 to day 10, so the total cost is calculated as 6 * 100 = 600.The second request is to book Room 2 from day 12 to day 15, so the total cost is 4 * 150 = 600.The third request is invalid because the start date is after the end date, so the system outputs "Invalid booking".

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int N, R, S, E;
int roomRates[10] = {100, 150, 200, 250, 300, 350, 400, 450, 500, 550};
scanf("%d", &N);

for (int i = 0; i < N; i++) {
scanf("%d %d %d", &R, &S, &E);

if (R < 1 || R > 10 || S < 1 || E < 1 || S > 31 || E > 31 || S > E) {
printf("Invalid booking\n");
continue;
}

int nights = E - S + 1;
int totalCost = nights * roomRates[R - 1];
printf("%d\n", totalCost);
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
2100
1200
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
200
Invalid booking
1650
```

Compilation Status: Passed

Execution Time:

0.001s

18. Problem StatementYou are tasked with building a reservation system for a conference room in a corporate office. The room can be reserved or freed up based on employee requests. Each request includes an action—either to reserve the room or free it up. Once reserved, no other employee can reserve the room until it is freed.

You need to handle these operations using the goto statement to jump to specific parts of the code based on the type of request and its outcome.

InputThe first line contains an integer N, the number of requests.The next N lines contain either one of two types of commands:"RESERVE X" where X is the employee ID attempting to reserve the room."FREE X" where X is the employee ID freeing up the room.

OutputFor each "RESERVE X" command, output "Room Reserved" if the room was successfully reserved, or "Room Already Reserved by Employee Y" if the room is already reserved by another employee, where Y is the employee ID who holds the reservation.For each "FREE X" command, output "Room Freed" if the room is freed by the employee who reserved it, or "No Reservation to Free" if no reservation exists or the reservation doesn't match the given employee ID.

Constraints1 ≤ N ≤ 1001 ≤ employee ID ≤ 1000There will be at most 100 commands.

Sample Input:5RESERVE 101RESERVE 102FREE 101RESERVE 102FREE 102

Sample Output:Room ReservedRoom Already Reserved by Employee 101Room FreedRoom ReservedRoom Freed

ExplanationThe first request "RESERVE 101" successfully reserves the room for employee 101.The second request "RESERVE 102" fails because the room is already reserved by employee 101.The third request "FREE 101" successfully frees the room.The fourth request "RESERVE 102" successfully reserves the room for employee 102.The fifth request "FREE 102" successfully frees the room.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int N, employeeID, currentReservation = -1;
char command[10];

scanf("%d", &N);

for (int i = 0; i < N; i++) {
scanf("%s %d", command, &employeeID);

if (strcmp(command, "RESERVE") == 0) {
if (currentReservation == -1) {
currentReservation = employeeID;
goto room_reserved;
} else if (currentReservation == employeeID) {
goto already_reserved_by_same;
} else {
goto already_reserved_by_other;
}
} else if (strcmp(command, "FREE") == 0) {
if (currentReservation == employeeID) {
currentReservation = -1;
goto room_freed;
} else if (currentReservation == -1) {
goto no_reservation;
} else {
goto reservation_does_not_match;
}
}

continue;

room_reserved:
printf("Room Reserved\n");
continue;

already_reserved_by_same:
printf("Room Already Reserved by Employee %d\n", currentReservation);
continue;

already_reserved_by_other:
printf("Room Already Reserved by Employee %d\n", currentReservation);
continue;
room_freed:
printf("Room Freed\n");
continue;

no_reservation:
printf("No Reservation to Free\n");
continue;

reservation_does_not_match:
printf("No Reservation to Free\n");
continue;
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Room Reserved
Room Freed
Room Reserved
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Room Reserved
Room Already Reserved by Employee 301
Room Freed
Room Reserved
```

Compilation Status: Passed

Execution Time:

0.001s

19. Problem StatementYou are developing a system for a logistics company that needs to track the shipments of packages. The system will categorize packages based on their weight and determine the cost of shipment based on distance and weight category. The company charges different rates for domestic and international shipments.

You need to implement different types of functions to calculate:

Weight Category: This function will bascially determine the weight category of the package.Cost Calculation: This function calculates the shipment cost based on the weight category and distance.Cost Display: This function displays the final cost along with the details of the shipment.Weight Categories:Lightweight: 0 < weight ≤ 5 kgStandard: 5 < weight ≤ 20 kgHeavy: 20 < weight ≤ 50 kgOverweight: weight > 50 kgRate Table (per km):Domestic:Lightweight: $1.00Standard: $1.50Heavy: $2.00Overweight: $3.00International:Lightweight: $2.00Standard: $3.00Heavy: $4.00Overweight: $6.00

InputThe first line will contain an integer N which represents the number of packages.The next N lines contain:A float W representing the weight of the package.An integer D will represent the distance in kilometers.A string T will represent the type of shipment ("domestic" or "international").

OutputFor each package, output the weight category, shipment type, and the calculated cost, each on a new line.

Constraints1 ≤ N ≤ 500 < W ≤ 1001 ≤ D ≤ 10000T is either "domestic" or "international"

ExampleInput34.5 1000 domestic15.0 2000 international55.0 5000 domestic

OutputLightweightdomestic1000.00Standardinternational6000.00Overweightdomestic15000.00

ExplanationThe first package is categorized as Lightweight and is domestic, so the rate is $1.00 per km. The cost is 1000 * 1.00 = 1000.00.The second package is categorized as Standard and is international, so the rate is $3.00 per km. The cost is 2000 * 3.00 = 6000.00.The third package is categorized as Overweight and is domestic, so the rate is $3.00 per km. The cost is 5000 * 3.00 = 15000.00.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
// Function to determine weight category
const char* determineWeightCategory(float weight) {
if (weight > 0 && weight <= 5) {
return "Lightweight";
} else if (weight > 5 && weight <= 20) {
return "Standard";
} else if (weight > 20 && weight <= 50) {
return "Heavy";
} else if (weight > 50) {
return "Overweight";
} else {
return "Invalid";
}
}

// Function to calculate the cost
float calculateCost(const char* weightCategory, int distance, const char*
shipmentType) {
float rate = 0.0;

if (strcmp(shipmentType, "domestic") == 0) {
if (strcmp(weightCategory, "Lightweight") == 0) rate = 1.00;
else if (strcmp(weightCategory, "Standard") == 0) rate = 1.50;
else if (strcmp(weightCategory, "Heavy") == 0) rate = 2.00;
else if (strcmp(weightCategory, "Overweight") == 0) rate = 3.00;
} else if (strcmp(shipmentType, "international") == 0) {
if (strcmp(weightCategory, "Lightweight") == 0) rate = 2.00;
else if (strcmp(weightCategory, "Standard") == 0) rate = 3.00;
else if (strcmp(weightCategory, "Heavy") == 0) rate = 4.00;
else if (strcmp(weightCategory, "Overweight") == 0) rate = 6.00;
}

return rate * distance;
}

int main() {
int n;
scanf("%d", &n);

for (int i = 0; i < n; i++) {
float weight;
int distance;
char shipmentType[15];

scanf("%f %d %s", &weight, &distance, shipmentType);
const char* weightCategory = determineWeightCategory(weight);

if (strcmp(weightCategory, "Invalid") == 0) {
printf("Invalid weight\n");
continue;
}

float cost = calculateCost(weightCategory, distance, shipmentType);

printf("%s\n", weightCategory);
printf("%s\n", shipmentType);
printf("%.2f\n", cost);
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Standard
domestic
2250.00
Heavy
international
12000.00
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Standard
international
15000.00
```

Compilation Status: Passed

Execution Time:

0.001s

20. Problem StatementYou are developing a system for a food delivery service that needs to handle multiple customer orders. Each order includes a list of items, each with a specific quantity and price. You need to implement functions to:

Calculate the total cost of each order.Apply a discount based on the total cost of the order.Determine the final price after applying the discount.The system should handle multiple orders and output the final price for each order.

Functions RequiredcalculateTotalCost(float prices[], int quantities[], int n): This function calculates the total cost of an order based on the prices and quantities of the items.applyDiscount(float totalCost): This function applies a discount to the total cost. The discount is as follows:If the total cost is less than $50, then there will be no discount.If the total cost is between $50 and $100, a 10% discount.If the total cost is more than $100, a 20% discount.finalPriceAfterDiscount(float totalCost): This function calculates the final price after applying the discount.

InputThe first line contains an integer N representing the number of orders.The next N lines each contain:An integer M representing the number of items in the order.M pairs of floating-point numbers where each pair represents the price and quantity of an item.

OutputFor each order, output the final price after applying the discount, rounded to two decimal places.

Constraints1 ≤ N ≤ 1001 ≤ M ≤ 500 < prices[i] ≤ 5000 < quantities[i] ≤ 100

ExampleInput23 10.0 2 20.0 1 5.0 34 15.0 1 25.0 2 10.0 2

Output49.5076.50

ExplanationFor the first order:

Total cost = (10.0 * 2) + (20.0 * 1) + (5.0 * 3) = 20.0 + 20.0 + 15.0 = 55.0Discount = 10% of 55.0 = 5.50Final price = 55.0 - 5.50 = 49.50For the second order:

Total cost = (15.0 * 1) + (25.0 * 1) + (10.0 * 2) = 15.0 + 25.0 + 20.0 = 60.0Discount = 10% of 60.0 = 6.00Final price = 60.0 - 6.00 = 54.00

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
float calculateTotalCost(float prices[], int quantities[], int n) {
float totalCost = 0.0;
for (int i = 0; i < n; i++) {
totalCost += prices[i] * quantities[i];
}
return totalCost;
}

float applyDiscount(float totalCost) {
if (totalCost < 50) {
return 0.0;
} else if (totalCost >= 50 && totalCost <= 100) {
return 0.10 * totalCost;
} else {
return 0.20 * totalCost;
}
}

float finalPriceAfterDiscount(float totalCost) {
float discount = applyDiscount(totalCost);
return totalCost - discount;
}

int main() {
int N;
scanf("%d", &N);

for (int i = 0; i < N; i++) {
int M;
scanf("%d", &M);

float prices[M];
int quantities[M];

for (int j = 0; j < M; j++) {
scanf("%f %d", &prices[j], &quantities[j]);
}

float totalCost = calculateTotalCost(prices, quantities, M);
float finalPrice = finalPriceAfterDiscount(totalCost);

printf("%.2f\n", finalPrice);
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
88.00
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
81.00
62.10
```

Compilation Status: Passed

Execution Time:

0.001s

21. Problem StatementYou are designing a system for an e-commerce platform where you need to calculate the total cost of items in a shopping cart with multiple discount tiers. Each discount tier provides a different discount rate depending on the total number of items in the cart. You will use recursion to calculate the total cost with discounts applied.

Discount Tiers:

Tier 1: 5% discount for 1 to 5 items.Tier 2: 10% discount for 6 to 10 items.Tier 3: 15% discount for more than 10 items.Your task is to:

Calculate the total cost of items before applying the discount.Apply the discount based on the total number of items.Return the final cost after applying the specific amount of discount.

Functions RequiredcalculateTotalCost(int items[], int n): Recursively calculates the total cost of the items in the cart.applyDiscount(float totalCost, int itemCount): Determines the amount of discount basically based on the number of items and then calculates the final price after applying the discount amount.

InputThe first line contains an integer N representing the number of items in the cart.The second line contains N integers where each integer represents the price of an item.

OutputOutput the final cost of the items in the cart after applying the discount, rounded to two decimal places.

Constraints1 ≤ N ≤ 1000 < items[i] ≤ 500

Sample Input:430 20 50 40

Sample Output:133.00

ExplanationThe total cost of items is 30 + 20 + 50 + 40 = 140.The number of items is 4, which falls under Tier 1 (5% discount).Discount = 5% of 140 = 7.Final cost = 140 - 7 = 133.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

float calculateTotalCost(int items[], int n) {
if (n == 0) {
return 0;
}
return items[n - 1] + calculateTotalCost(items, n - 1);
}

float applyDiscount(float totalCost, int itemCount) {
float discount = 0.0;
if (itemCount >= 1 && itemCount <= 5) {
discount = 0.05 * totalCost;
} else if (itemCount >= 6 && itemCount <= 10) {
discount = 0.10 * totalCost;
} else if (itemCount > 10) {
discount = 0.15 * totalCost;
}
return totalCost - discount;
}

int main() {
int N;
scanf("%d", &N);

int items[N];
for (int i = 0; i < N; i++) {
scanf("%d", &items[i]);
}
float totalCost = calculateTotalCost(items, N);
float finalCost = applyDiscount(totalCost, N);
printf("%.2f\n", finalCost);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
142.50
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
288.00
```

Compilation Status: Passed

Execution Time:

0.001s

22. Problem StatementYou are developing a system to manage employee bonuses based on their performance scores and work hours. The system needs to handle various types of functions to calculate bonuses using different strategies. You will need to implement the functions that:

Calculate the base bonus based on performance scores.Calculate additional bonuses based on work hours.Apply a special adjustment based on employee type (full-time or part-time).Functions Requiredfloat calculateBaseBonus(float performanceScore): Calculates the base bonus based on performance score.float calculateAdditionalBonus(float workHours): Calculates the additional bonus based on work hours.float applyAdjustment(float bonus, char employeeType): Applies a special adjustment based on whether the employee is full-time or part-time.Note: The special adjustment is:

Full-time: No adjustment.Part-time: 5% reduction in the total bonus.

InputThe first line contains an integer N representing the number of employees.The next N lines each contain:A floating-point number representing the performance score.A floating-point number representing the total work hours.A character representing the employee type ('F' for full-time, 'P' for part-time).

OutputFor each employee, output the final bonus amount after applying the adjustment, rounded to two decimal places.

Constraints1 ≤ N ≤ 1000 ≤ performanceScore ≤ 1000 ≤ workHours ≤ 200employeeType ∈ {'F', 'P'}

Sample Input:39.5 40.0 F8.0 35.0 P7.0 30.0 F

Sample Output875.00712.50650.00

ExplanationFor the first employee:

Base bonus = 85.0 * 50 = 4250.00Additional bonus = 40.0 * 10 = 400.00Total bonus before adjustment = 4250.00 + 400.00 = 4650.00Adjustment (full-time) = 0%Final bonus = 4650.00For the second employee:

Base bonus = 75.0 * 50 = 3750.00Additional bonus = 35.0 * 10 = 350.00Total bonus before adjustment = 3750.00 + 350.00 = 4100.00Adjustment (part-time) = 5%Final bonus = 4100.00 * 0.95 = 3895.00For the third employee:

Base bonus = 90.0 * 50 = 4500.00Additional bonus = 50.0 * 10 = 500.00Total bonus before adjustment = 4500.00 + 500.00 = 5000.00Adjustment (full-time) = 0%Final bonus = 5000.00

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

float calculateBaseBonus(float performanceScore) {
return performanceScore * 50;
}

float calculateAdditionalBonus(float workHours) {
return workHours * 10;
}

float applyAdjustment(float bonus, char employeeType) {
if (employeeType == 'P') {
return bonus * 0.95; // 5% reduction for part-time
}
return bonus; // No adjustment for full-time
}

int main() {
int N;
scanf("%d", &N);

for (int i = 0; i < N; i++) {
float performanceScore, workHours;
char employeeType;

scanf("%f %f %c", &performanceScore, &workHours, &employeeType);

float baseBonus = calculateBaseBonus(performanceScore);
float additionalBonus = calculateAdditionalBonus(workHours);
float totalBonus = baseBonus + additionalBonus;
float finalBonus = applyAdjustment(totalBonus, employeeType);

printf("%.2f\n", finalBonus);
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
875.00
712.50
650.00
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
3500.00
4227.50
5350.00
```

Compilation Status: Passed

Execution Time:

0.001s

23. Problem StatementTitle: Array Transformation by Value

Problem Description:

You are given an array arr of size n consisting of integers and two additional integers x and y. Your task is to implement a function transformArray(arr, n, x, y) that performs the following operations:

Element Replacement: Traverse the array, and for every element arr[i], if it is divisible by x, replace it with y.Incremental Sum Update: After the replacement, for each index i from 0 to n-2, set arr[i] to the sum of itself and the next element (arr[i] = arr[i] + arr[i+1]). The last element of the array remains unchanged.Output Maximum Value: Return the maximum value from the modified array.

Input:The first line contains an integer n ($1 \le n \le 10^5$), which is the size of the array.The second line contains n space-separated integers representing the elements of the array arr ($0 \le arr[i] \le 10^9$).The third line contains two integers x and y ($1 \le x, y \le 10^9$).

Output:Print a single integer, which is the maximum value in the array after performing the operations.

Sample Input:512 15 7 18 243 5

Sample Output:12

Constraints:The array size n can be large.Elements of the array can be as large as one billion.The solution should be optimized for both time and space.

Example:Consider an example where arr = [12, 15, 7, 18, 24], x = 3, and y = 5.

Step 1: Element Replacement

12 % 3 == 0 □ Replace 12 with 5.15 % 3 == 0 □ Replace 15 with 5.18 % 3 == 0 □ Replace 18 with 5.24 % 3 == 0 □ Replace 24 with 5.7 is not divisible by 3, so it remains unchanged.Array after replacement: [5, 5, 7, 5, 5].

Step 2: Incremental Sum Update

For i = 0, arr[0] = 5 + 5 = 10.For i = 1, arr[1] = 5 + 7 = 12.For i = 2, arr[2] = 7 + 5 = 12.For i = 3, arr[3] = 5 + 5 = 10.The last element (arr[4]) remains 5.Final array: [10, 12, 12, 10, 5].

Step 3: Output Maximum Value

The maximum value in the array is 12.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int transformArray(int arr[], int n, int x, int y) {
// Step 1: Element Replacement
for (int i = 0; i < n; i++) {
if (arr[i] % x == 0) {
arr[i] = y;
}
}

// Step 2: Incremental Sum Update
for (int i = 0; i < n - 1; i++) {
arr[i] = arr[i] + arr[i + 1];
}

// Step 3: Find and return the maximum value
int maxVal = arr[0];
for (int i = 1; i < n; i++) {
if (arr[i] > maxVal) {
maxVal = arr[i];
}
}

return maxVal;
}

int main() {
int n;
scanf("%d", &n);

int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

int x, y;
scanf("%d %d", &x, &y);

int result = transformArray(arr, n, x, y);
printf("%d\n", result);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
12
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6

Compilation Status: Passed

Execution Time:

0.001s

24. Problem StatementTitle: Advanced Linked List Manipulation

Problem Description:

You are tasked with implementing a function that manipulates a doubly linked list using the concept of call by address. The linked list contains integer nodes, and you need to perform the following operations on the list:

Reverse the List: Reverse the entire linked list.Remove Duplicates: After reversing, remove all duplicate elements from the list, retaining only the first occurrence of each element.Insert Sum at End: After removing duplicates, calculate the sum of all the elements in the linked list and insert this sum as a new node at the end of the list.These operations should be done by passing the head of the list to the function by using a call by address.

Input:The first line contains an integer n (1 ≤ n ≤ 10^5), the number of nodes in the linked list.The next line contains n space-separated integers representing the elements of the linked list.

Output:Print the elements of the modified linked list after performing all the operations.

Sample Input:74 3 2 3 4 5 2

Sample Output:2 5 4 3 14

Constraints:The number of nodes n can be very large, up to 100,000.The values of the nodes can be as large as 1,000,000.The operations must be optimized for both time and space.

Example:Consider a linked list with n = 7 and the elements 4 -> 3 -> 2 -> 3 -> 4 -> 5 -> 2.

Step 1: Reverse the List

The list becomes: 2 -> 5 -> 4 -> 3 -> 2 -> 3 -> 4Step 2: Remove Duplicates

The list after removing duplicates: 2 -> 5 -> 4 -> 3Step 3: Insert Sum at End

The sum of the elements is 2 + 5 + 4 + 3 = 14.The final list is: 2 -> 5 -> 4 -> 3 -> 14.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

struct Node {
int data;
struct Node* next;
struct Node* prev;
};

void reverseList(struct Node** head) {
struct Node* temp = NULL;
struct Node* current = *head;

while (current != NULL) {
temp = current->prev;
current->prev = current->next;
current->next = temp;
current = current->prev;
}

if (temp != NULL) {
*head = temp->prev;
}
}

void removeDuplicates(struct Node** head) {
struct Node* current = *head;
struct Node* prev = NULL;
struct Node* temp = NULL;

while (current != NULL) {
prev = current;
temp = current->next;

while (temp != NULL) {
if (current->data == temp->data) {
prev->next = temp->next;
if (temp->next != NULL) {
temp->next->prev = prev;
}
free(temp);
temp = prev->next;
} else {
prev = temp;
temp = temp->next;
}
}
current = current->next;
}
}

void insertSumAtEnd(struct Node** head) {
struct Node* current = *head;
int sum = 0;

while (current != NULL) {
sum += current->data;
current = current->next;
}

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = sum;
newNode->next = NULL;

current = *head;
while (current->next != NULL) {
current = current->next;
}
current->next = newNode;
newNode->prev = current;
}

void printList(struct Node* head) {
struct Node* temp = head;
while (temp != NULL) {
printf("%d ", temp->data);
temp = temp->next;
}
printf("\n");
}

void append(struct Node** head, int new_data) {
struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
struct Node* last = *head;

new_node->data = new_data;
new_node->next = NULL;

if (*head == NULL) {
new_node->prev = NULL;
*head = new_node;
return;
}

while (last->next != NULL) {
last = last->next;
```

```
}

last->next = new_node;
new_node->prev = last;
}

int main() {
int n;
scanf("%d", &n);

struct Node* head = NULL;

for (int i = 0; i < n; i++) {
int data;
scanf("%d", &data);
append(&head, data);
}

reverseList(&head);
removeDuplicates(&head);
insertSumAtEnd(&head);
printList(head);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
2 5 4 3 14
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5 4 3 2 1 15

Compilation Status: Passed

Execution Time:

0.001s

25. Problem StatementTitle: Maximum Difference in Array

Problem Statement:You are given an array of integers, and you need to find the maximum difference between any two elements in the array. The difference is defined as the absolute difference between the two elements. The array may contain both positive and negative numbers, and your task is to ensure the difference calculation considers all possible pairs within the array.

Input:

The first line contains an integer N (1 ≤ N ≤ 1000) representing the number of elements in the array.The second line contains N space-separated integers representing the elements of the array A[i] (-10^5 ≤ A[i] ≤ 10^5).

Output:

A single integer, the maximum difference between any two elements in the array.

Constraints:

1 ≤ N ≤ 1000-10^5 ≤ A[i] ≤ 10^5

Example:

Example 1:Input:51 -3 2 10 6

Output:13

Explanation:The maximum difference is between -3 and 10, which is |10 - (-3)| = 13.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int N;
scanf("%d", &N);

int arr[N];
int min = INT_MAX, max = INT_MIN;

for (int i = 0; i < N; i++) {
scanf("%d", &arr[i]);
if (arr[i] < min) {
min = arr[i];
}
if (arr[i] > max) {
max = arr[i];
}
}

int max_diff = max - min;
printf("%d\n", max_diff);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
0
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

20

Compilation Status: Passed

Execution Time:

0.001s

26. Problem Statement: Chessboard Pattern Analysis

You are given an n × n chessboard-like matrix where each cell contains either a 1 (indicating a white square) or a 0 (indicating a black square). The matrix follows the alternating pattern of a standard chessboard, but with some cells corrupted, breaking the pattern. Your task is to determine the minimum number of cells that need to be flipped (from 1 to 0 or from 0 to 1) to restore the matrix to a proper chessboard pattern.

Input Format:The first line contains an integer n (where 2 ≤ n ≤ 500), representing the size of the matrix.The next n lines each contain n integers (either 0 or 1), representing the matrix.

Output Format:Output a single integer, the minimum number of cells that need to be flipped to restore the matrix to a proper chessboard pattern.

Constraints:2 ≤ n ≤ 500

Each cell in the matrix is either 0 or 1.The matrix may have at most 50% of its cells corrupted.

Example:Input:41 0 1 00 1 0 11 1 0 00 0 1 1

Output:4

Explanation:The provided 4 × 4 matrix should follow a chessboard pattern:

Pattern 1:1 0 1 00 1 0 11 0 1 00 1 0 1

Pattern 2:0 1 0 11 0 1 00 1 0 11 0 1 0

We compare the given matrix with both Pattern 1 and Pattern 2. For Pattern 1, we need to flip the two bottom-right cells to make the matrix valid. For Pattern 2, we would need to flip more cells, so the optimal solution is to follow Pattern 1 with a total of 2 flips.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n;
scanf("%d", &n);

int matrix[n][n];
for(int i = 0; i < n; i++) {
for(int j = 0; j < n; j++) {
scanf("%d", &matrix[i][j]);
}
}

int pattern1 = 0, pattern2 = 0;

for(int i = 0; i < n; i++) {
for(int j = 0; j < n; j++) {
if((i + j) % 2 == 0) {
if(matrix[i][j] != 1) pattern1++;
if(matrix[i][j] != 0) pattern2++;
} else {
if(matrix[i][j] != 0) pattern1++;
if(matrix[i][j] != 1) pattern2++;
}
}
}

if(pattern1 < pattern2) printf("%d\n", pattern1);
else printf("%d\n", pattern2);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.001s

27. Problem StatementTitle: Dynamic Island Formation in a Multi-dimensional Grid

Problem Description:You have a 2D grid of size n x m consisting of integers, where every cell of the grid represents a part of a map. The grid can contain three types of values:

0 represents water,1 represents land,2 represents a bridge connecting two pieces of land.Your task is to implement a function countIslands(int grid[][m], int n, int m) that finds the number of distinct islands in the grid. An island is formed by connecting adjacent lands horizontally, vertically, or diagonally. Bridges (2) should be considered as land and can connect separate pieces of land, forming a single island.

Input:The first line contains two integers n and m (1 ≤ n, m ≤ 1000), representing the dimensions of the grid.The next n lines each contain m space-separated integers representing the elements of the grid (0 ≤ grid[i][j] ≤ 2).

Output:Print a single integer, the number of distinct islands found in the grid.

Constraints:The grid dimensions n and m can be large, up to 1000x1000.The grid values can be 0, 1, or 2.You should use an efficient algorithm to handle the large input size.

Example:Consider the following grid with n = 5 and m = 6:

Input:5 61 0 0 0 0 00 1 1 0 2 00 0 0 1 0 01 0 0 2 1 10 0 0 0 0 1

Output:2

Step 1: Identifying Islands

There are several pieces of land connected by bridges.Step 2: Counting Islands

There are 3 distinct islands in the grid.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#define MAX_N 1000
#define MAX_M 1000
int grid[MAX_N][MAX_M];
int visited[MAX_N][MAX_M];
int n, m;
int directions[8][2] = {
{1, 0},  {-1, 0},  {0, 1},  {0, -1},
{1, 1},  {1, -1},  {-1, 1}, {-1, -1}
};
void dfs(int x, int y) {
visited[x][y] = 1;

for (int i = 0; i < 8; i++) {
int nx = x + directions[i][0];
int ny = y + directions[i][1];

if (nx >= 0 && ny >= 0 && nx < n && ny < m && !visited[nx][ny] && (grid[nx][ny] == 1 ||
grid[nx][ny] == 2)) {
dfs(nx, ny);
}
}
}

int countIslands(int grid[][MAX_M], int n, int m) {
int islandCount = 0;

for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
if (!visited[i][j] && (grid[i][j] == 1 || grid[i][j] == 2)) {
dfs(i, j);
islandCount++;
}
}
}
return islandCount;
}

int main() {
scanf("%d %d", &n, &m);

for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
scanf("%d", &grid[i][j]);
visited[i][j] = 0;
}
}

int result = countIslands(grid, n, m);
printf("%d\n", result);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.001s

28. Problem StatementTitle: Subarray Maximum Product

Problem Description:

Given an array of n integers, your task is to find the maximum product of any contiguous subarray. The subarray must contain at least one element, and you need to implement the solution by passing the array to functions.

You are required to implement two functions:

maxProductSubarray(int arr[], int n): This function takes the array and its size as arguments and returns the maximum product of any contiguous subarray.getInputArray(int arr[], int n): This function takes the array and its size as arguments and handles the input from the user.

Input:The first line contains an integer n (1 ≤ n ≤ 10^5), the size of the array.The second line contains n space-separated integers representing the elements of the array arr[i] (-10^5 ≤ arr[i] ≤ 10^5).

Output:Print a single integer, the maximum product of any contiguous subarray.

Constraints:The array size n can be large, up to 100,000.The elements of the array can be both positive and negative, including zeros.The solution must optimize both time and space complexity.

Example:Consider the following array with n = 5:

Input:52 3 -2 4 -1

Step 1: Find the Maximum Product Subarray

The maximum product is obtained by considering the subarray [2, 3, -2, 4] which yields a product of 48.

Output:48

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
void getInputArray(int arr[], int n) {
```

```
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
}
int maxProductSubarray(int arr[], int n) {
int currentMax = arr[0];
int currentMin = arr[0];
int result = arr[0];
for (int i = 1; i < n; i++) {
if (arr[i] < 0) {
int temp = currentMax;
currentMax = currentMin;
currentMin = temp;
}

currentMax = arr[i] > arr[i] * currentMax ? arr[i] : arr[i] * currentMax;
currentMin = arr[i] < arr[i] * currentMin ? arr[i] : arr[i] * currentMin;

result = result > currentMax ? result : currentMax;
}

return result;
}

int main() {
int n;
scanf("%d", &n);

int arr[n];
getInputArray(arr, n);

int result = maxProductSubarray(arr, n);
printf("%d\n", result);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
48
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

4

Compilation Status: Passed

Execution Time:

0.001s

29. Problem StatementTitle: Contiguous Memory Allocation in Array for a Warehouse

Problem Description:

A warehouse management system is being developed to store various items in an array where each index of the array represents a specific section of the warehouse. The system needs to manage items by finding a contiguous block of memory (subarray) of a given length k that can accommodate a new batch of items. The challenge is bascially to find the contiguous subarray which has the maximum sum, where your sum represents the availability of space (or capacity) in that subarray.

You are required to implement a function findMaxCapacity(int arr[], int n, int k) that takes an array representing the capacity of sections and returns the starting index of the subarray of length k with the maximum sum. If there are multiple subarrays with the same maximum sum, return the smallest starting index.

Additionally, you need to implement a helper function getInputArray(int arr[], int n) to handle input from the user.

Input:The first line will contain two integers n and k (1 ≤ k ≤ n ≤ 10^6), representing the size of the array and the length of the subarray.The second line will contain n space-separated integers which represents the elements of the array arr[i] (-10^4 ≤ arr[i] ≤ 10^4).

Output:Print a single integer, the starting index (0-based) of the subarray of length k with the maximum sum.

Constraints:The size of the array n can be large, up to 1,000,000.The length k of the subarray is always less than or equal to n.The elements of the array can be both positive and negative.

Example:Consider the following array with n = 6 and k = 3:

Input:6 32 1 -3 4 5 -6

Step 1: Identify the subarray with the maximum sum

Subarray [4, 5, -6] starting at index 3 has the highest sum of 3.

Output:2

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int main() {
int n, k;
scanf("%d %d", &n, &k);

int arr[n];
for(int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

int current_sum = 0;
for(int i = 0; i < k; i++) {
current_sum += arr[i];
}

int max_sum = current_sum;
int start_index = 0;

for(int i = k; i < n; i++) {
current_sum += arr[i] - arr[i - k];
if(current_sum > max_sum) {
max_sum = current_sum;
start_index = i - k + 1;
}
}

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

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

4

Compilation Status: Passed

Execution Time:

0.001s

30. Problem StatementTitle: Matrix Path Finder

Difficulty Level: Hard

Problem Description:

You are given an n x m grid representing a matrix of integers. Your task is to find the maximum sum possible by starting at the top-left corner of the matrix (i.e., cell (0, 0)) and moving to the bottom-right corner of the matrix (i.e., cell (n-1, m-1)). You can only move either to the right or down at each step.

You need to implement this using array declarations and the array index operator to access matrix elements.

The solution must be implemented using a function findMaxPathSum(int matrix[][100], int n, int m) that calculates the maximum sum. The input should be handled by a helper function getInputMatrix(int matrix[][100], int n, int m).

Input:The first line contains two integers n and m (1 ≤ n, m ≤ 100), representing the dimensions of the matrix.The next n lines each contain m space-separated integers representing the matrix elements.

Output:Print a single integer, the maximum sum obtainable by following the path fr the top-left to the bottom-right of the matrix.

Sample Input:3 31 2 34 5 67 8 9

Sample Output:29

Constraints:The matrix size n x m will be at most 100 x 100.The elements of the matrix can be negative, zero, or positive integers.

Example:Consider the following matrix with n = 3 and m = 3:Step 1: Find the maximum path sum

The maximum path sum is 1 + 4 + 7 + 8 + 9 = 29.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void getInputMatrix(int matrix[][100], int n, int m) {
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
scanf("%d", &matrix[i][j]);
}
}
}

int findMaxPathSum(int matrix[][100], int n, int m) {
int dp[100][100];

dp[0][0] = matrix[0][0];

for (int j = 1; j < m; j++) {
dp[0][j] = dp[0][j-1] + matrix[0][j];
}

for (int i = 1; i < n; i++) {
dp[i][0] = dp[i-1][0] + matrix[i][0];
}

for (int i = 1; i < n; i++) {
```

```
for (int j = 1; j < m; j++) {
dp[i][j] = matrix[i][j] + (dp[i-1][j] > dp[i][j-1] ? dp[i-1][j] : dp[i][j-1]);
}
}

return dp[n-1][m-1];
}

int main() {
int n, m;
scanf("%d %d", &n, &m);
int matrix[100][100];
getInputMatrix(matrix, n, m);
int result = findMaxPathSum(matrix, n, m);
printf("%d\n", result);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
29
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
68
```

Compilation Status: Passed

Execution Time:

0.001s

31. Problem StatementTitle: Array Rotation Puzzle

Problem Description:

You are working on a project where you need to manipulate a large array of integers for data processing. One of the operations involves rotating the elements of the array to the left or right by a given number of positions. The goal is to write a C program that performs this operation efficiently.

You are required to implement the function rotateArray(int arr[], int n, int d, char dir) that rotates the array arr[] of size n by d positions. The direction of rotation is determined by the dir parameter, where 'L' indicates a left rotation and 'R' indicates a right rotation.

The solution must include proper looping through the array elements to achieve the rotation without using extra space for another array (in-place rotation).

Input:The first line contains two integers n and d (1 ≤ d ≤ n ≤ 10^6), representing the size of the array and the number of positions to rotate.The second line will contain the n space-separated integers which basically represents the elements of the array arr[i] (-10^4 ≤ arr[i] ≤ 10^4).The third line contains a single character dir which is either 'L' or 'R', representing the direction of rotation.

Output:Print the rotated array as a single line of space-separated integers.

Sample Input:5 21 2 3 4 5L

Sample Output:3 4 5 1 2

Constraints:The size of the array n can be as large as, up to 1,000,000.The number of positions d will always be less than or equal to n.The elements of the array can be both positive and negative integers.

Example:Consider the following array with n = 5, d = 2, and dir = 'L':

Step 1: Rotate the array by 2 positions to the left

The resulting array after the rotation will be: 3 4 5 1 2

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void reverse(int arr[], int start, int end) {
while (start < end) {
int temp = arr[start];
arr[start] = arr[end];
arr[end] = temp;
start++;
end--;
}
}

void rotateArray(int arr[], int n, int d, char dir) {
if (dir == 'L') {
d = d % n;
reverse(arr, 0, n - 1);
reverse(arr, 0, n - d - 1);
reverse(arr, n - d, n - 1);
} else if (dir == 'R') {
d = d % n;
d = n - d;
reverse(arr, 0, n - 1);
reverse(arr, 0, n - d - 1);
reverse(arr, n - d, n - 1);
}
}

int main() {
int n, d;
char dir;

scanf("%d %d", &n, &d);

int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

scanf(" %c", &dir);

rotateArray(arr, n, d, dir);

for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
printf("\n");

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
3 4 5 1 2
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
-4 -5 -6 -1 -2 -3
```

Compilation Status: Passed

Execution Time:

0.001s

32. Problem StatementTitle: Inventory Management using Heap Allocated Arrays

Problem Description:

You are working for a logistics company that manages inventory for different types of products stored in a warehouse. Each product has a unique identifier, a name, a quantity, and a price. You need to write a C program to store and manage the inventory information dynamically using heap-allocated arrays.

Your task:

Add Product: Add a new product to the inventory.Update Product Quantity: Update the quantity of a specific product.Calculate Total Inventory Value: Calculate the total value of the inventory.Remove Product: Remove a product from the inventory.List All Products: List all products in the inventory.You must use dynamic memory allocation to store the product information and ensure that the program efficiently handles memory reallocation as products are added or removed.

Input format:

The first input is an integer N, which represents the number of operations to be performed.Each of the following N lines represents an operation. The operations ca be one of the following:ADD id name quantity priceUPDATE id quantityREMOVE idTOTALLIST

Output format:

For each operation, the output will vary:

For ADD, no output is required.For UPDATE, print "Product updated successfully" if the product exists, otherwise print "Product not found".For REMOVE, print "Product removed successfully" if the product exists, otherwise print "Product not found".For TOTAL, print the total inventory value as a float rounded to 2 decimal places.For LIST, print all products in the format: id name quantity price.

Sample Input:6ADD 101 "Laptop" 10 599.99ADD 102 "Mouse" 50 19.99UPDATE 101 15TOTALREMOVE 102LIST

Sample Output:Product updated successfully9999.35Product removed successfully101 "Laptop" 15 599.99

Constraints:

1 <= N <= 10000 <= quantity <= 100000.0 <= price <= 100000.0The length of the product name will not exceed 50 characters.All product identifiers will be unique and are positive integers.You must use dynamic memory allocation (malloc, realloc, free) to manage the product inventory.

Explanation:

Consider the following example:

Explanation:Two products, Laptop and Mouse, are added to the inventory.The quantity of the Laptop is updated to 15.The total inventory value is calculated as (15 * 599.99) + (50 * 19.99) = 13499.85.The Mouse is removed from the inventory.The remaining product (Laptop) is listed with its details.

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

typedef struct {
```

```
int id;
char name[51];
int quantity;
float price;
} Product;

void addProduct(Product** inventory, int* size, int* capacity, int id, const char* name,
int
quantity, float price) {
if (*size == *capacity) {
*capacity *= 2;
*inventory = realloc(*inventory, (*capacity) * sizeof(Product));
}
(*inventory)[*size].id = id;
strcpy((*inventory)[*size].name, name);
(*inventory)[*size].quantity = quantity;
(*inventory)[*size].price = price;
(*size)++;
}

void updateProduct(Product* inventory, int size, int id, int quantity) {
for (int i = 0; i < size; i++) {
if (inventory[i].id == id) {
inventory[i].quantity = quantity;
printf("Product updated successfully\n");
return;
}
}
printf("Product not found\n");
}

void removeProduct(Product** inventory, int* size, int id) {
for (int i = 0; i < *size; i++) {
if ((*inventory)[i].id == id) {
for (int j = i; j < *size - 1; j++) {
(*inventory)[j] = (*inventory)[j + 1];
}
(*size)--;
printf("Product removed successfully\n");
return;
}
}
printf("Product not found\n");
}

void calculateTotalValue(Product* inventory, int size) {
float total = 0.0;
for (int i = 0; i < size; i++) {
total += inventory[i].quantity * inventory[i].price;
}
printf("%.2f\n", total);
}

void listAllProducts(Product* inventory, int size) {
```

```
for (int i = 0; i < size; i++) {
printf("%d \"%s\" %d %.2f\n", inventory[i].id, inventory[i].name, inventory[i].quantity,
inventory[i].price);
}
}

int main() {
int n;
scanf("%d", &n);

Product* inventory = malloc(2 * sizeof(Product));
int size = 0, capacity = 2;

for (int i = 0; i < n; i++) {
char operation[10];
scanf("%s", operation);

if (strcmp(operation, "ADD") == 0) {
int id, quantity;
float price;
char name[51];
scanf("%d \"%[^\"]\" %d %f", &id, name, &quantity, &price);
addProduct(&inventory, &size, &capacity, id, name, quantity, price);
} else if (strcmp(operation, "UPDATE") == 0) {
int id, quantity;
scanf("%d %d", &id, &quantity);
updateProduct(inventory, size, id, quantity);
} else if (strcmp(operation, "REMOVE") == 0) {
int id;
scanf("%d", &id);
removeProduct(&inventory, &size, id);
} else if (strcmp(operation, "TOTAL") == 0) {
calculateTotalValue(inventory, size);
} else if (strcmp(operation, "LIST") == 0) {
listAllProducts(inventory, size);
}
}

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

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
6999.40
Product updated successfully
101 "Laptop" 10 599.99
102 "Mouse" 45 19.99
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Product not found
Product removed successfully
```

Compilation Status: Passed

Execution Time:

0.001s

33. Reversing Words in a Sentence Stored in a Dynamic ArrayProblem Statement:You are required to write a program that accepts a sentence from the user, dynamically allocates memory for this sentence, and then reverses the order of the words in the sentence. The reversal should happen at the word level, not the character level. The reversed sentence should be printed as the output.

Description:Your task is to handle a string input dynamically using heap memory. The input string will contain multiple words separated by spaces. You need to reverse the order of the words in the sentence while keeping the characters within each word intact. For example, the sentence "Hello World" should become "World Hello". Be careful with the memory allocation and deallocation to avoid memory leaks.

Input Format:

A single line of text containing a sentence. The sentence can have multiple words separated by spaces.Output Format:

A single line of text containing the sentence with the order of words reversed.

Sample Input:Memory management in C

Sample Output:C in management Memory

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void reverseWords(char *sentence) {
char *words[100];
int wordCount = 0;
char *token = strtok(sentence, " ");
while (token != NULL) {
words[wordCount++] = token;
token = strtok(NULL, " ");
}
for (int i = wordCount - 1; i >= 0; i--) {
printf("%s", words[i]);
if (i != 0) {
printf(" ");
}
}
printf("\n");
}

int main() {
char *sentence = (char *)malloc(1000 * sizeof(char));
if (sentence == NULL) {
printf("Memory allocation failed\n");
return 1;
}
fgets(sentence, 1000, stdin);
sentence[strcspn(sentence, "\n")] = '\0';
reverseWords(sentence);
free(sentence);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
C in management Memory
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
fun is C Learning
```

Compilation Status: Passed

Execution Time:

0.001s

34. Frequency of Vowels in a Dynamically Allocated String ArrayProblem Statement:Write a program that accepts multiple strings from the user, stores them in a dynamically allocated array, and then counts and displays the frequency of each vowel (a, e, i, o, u) across all the strings combined.

Description:You are tasked with managing multiple strings using a dynamic array of pointers. Each string will be stored in the heap. After storing the strings, your program should calculate the frequency of each vowel in all the strings combined. Consider both uppercase and lowercase vowels, but treat them as the same (e.g., 'A' and 'a' should be counted together).

Input Format:

The first line contains an integer n, the number of strings.The following n lines each contain a single string.Output Format:

Print the frequency of each vowel (a, e, i, o, u) across all the strings.

Sample Input:3HelloWorldEducation

Sample Output:a: 1, e: 2, i: 1, o: 3, u: 1

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int n;
scanf("%d", &n);
getchar();

char **strings = (char **)malloc(n * sizeof(char *));

if (strings == NULL) {
printf("Memory allocation failed\n");
return 1;
}

for (int i = 0; i < n; i++) {
strings[i] = (char *)malloc(100 * sizeof(char));
if (strings[i] == NULL) {
printf("Memory allocation failed\n");
return 1;
}
fgets(strings[i], 100, stdin);
strings[i][strcspn(strings[i], "\n")] = '\0';
}

int vowels[5] = {0};

for (int i = 0; i < n; i++) {
for (int j = 0; strings[i][j] != '\0'; j++) {
char c = tolower(strings[i][j]);
if (c == 'a') vowels[0]++;
else if (c == 'e') vowels[1]++;
else if (c == 'i') vowels[2]++;
else if (c == 'o') vowels[3]++;
else if (c == 'u') vowels[4]++;
}
}

printf("a: %d, e: %d, i: %d, o: %d, u: %d\n", vowels[0], vowels[1], vowels[2], vowels[3],
vowels[4]);

for (int i = 0; i < n; i++) {
free(strings[i]);
}
free(strings);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
a: 1, e: 2, i: 1, o: 3, u: 1
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
a: 3, e: 1, i: 1, o: 1, u: 1
```

Compilation Status: Passed

Execution Time:

0.001s

35. Character Frequency Distribution in a ParagraphProblem Statement:You are required to write a program that accepts a paragraph from the user and dynamically allocates memory to store it. The program should then count the frequency of each character (ignoring case) and print the character frequency distribution in descending order of frequency. Ignore spaces and punctuation in the count.

Description:Your task is to handle a paragraph input dynamically using heap memory. The input paragraph may contain multiple sentences, spaces, and punctuation marks. You need to count the frequency of each character (ignoring case and excluding spaces/punctuation) and then print the frequencies in descending order. The characters with the same frequency should be printed in alphabetical order.

Input Format:

A single line of text containing a paragraph.Output Format:

A list of characters with their frequencies, sorted by frequency in descending order.

Sample Input:C programming is fun!

Sample Output:g: 2i: 2m: 2n: 2r: 2a: 1c: 1f: 1o: 1p: 1s: 1u: 1

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAX_CHAR 26

typedef struct {
char character;
int frequency;
} CharFreq;

void countFrequency(char *str, CharFreq **freq, int *size) {
int count[MAX_CHAR] = {0};
```

```
for (int i = 0; str[i] != '\0'; i++) {
char c = tolower(str[i]);
if (isalpha(c)) {
count[c - 'a']++;
}
}

*size = 0;
for (int i = 0; i < MAX_CHAR; i++) {
if (count[i] > 0) {
(*size)++;
}
}

*freq = (CharFreq *)malloc(*size * sizeof(CharFreq));

int index = 0;
for (int i = 0; i < MAX_CHAR; i++) {
if (count[i] > 0) {
(*freq)[index].character = i + 'a';
(*freq)[index].frequency = count[i];
index++;
}
}
}

int compare(const void *a, const void *b) {
CharFreq *freqA = (CharFreq *)a;
CharFreq *freqB = (CharFreq *)b;

if (freqB->frequency != freqA->frequency) {
return freqB->frequency - freqA->frequency;
}

return freqA->character - freqB->character;
}

int main() {
char *input = NULL;
size_t len = 0;
ssize_t read;

read = getline(&input, &len, stdin);
if (read == -1) {
printf("Error reading input.\n");
return 1;
}

input[strcspn(input, "\n")] = '\0';

CharFreq *freq = NULL;
int size = 0;
```

```
countFrequency(input, &freq, &size);

qsort(freq, size, sizeof(CharFreq), compare);

for (int i = 0; i < size; i++) {
printf("%c: %d\n", freq[i].character, freq[i].frequency);
}

free(input);
free(freq);

return 0;}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
g: 2
i: 2
m: 2
n: 2
r: 2
a: 1
c: 1
f: 1
o: 1
p: 1
s: 1
u: 1
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
a: 4
t: 4
r: 3
s: 3
d: 2
u: 2
c: 1
e: 1
g: 1
h: 1
i: 1
l: 1
m: 1
n: 1
o: 1
```

Compilation Status: Passed

Execution Time:

0.001s

36. 2D Array Rotation by 90 DegreesProblem Statement:Write a program that accepts a 2D array of integers (a matrix) from the user, dynamically allocates memory for the matrix, and rotates the matrix by 90 degrees clockwise. The rotated matrix should then be printed as the output.

Description:Your task is to handle the matrix input dynamically using heap memory. The input matrix will be square (n x n), and you need to rotate the matrix by 90 degrees clockwise. This operation involves transposing the matrix and then reversing the rows.

Input Format:

The first line contains an integer n, the size of the matrix (n x n).The next n lines contain n integers each, representing the matrix.Output Format:

The output should be the rotated matrix, with each row printed on a new line.

Sample Input:31 2 34 5 67 8 9

Sample Output:7 4 1 8 5 2 9 6 3

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void rotateMatrix(int **matrix, int n) {
for (int i = 0; i < n; i++) {
for (int j = i; j < n; j++) {
int temp = matrix[i][j];
matrix[i][j] = matrix[j][i];
matrix[j][i] = temp;
}
}

for (int i = 0; i < n; i++) {
for (int j = 0; j < n / 2; j++) {
int temp = matrix[i][j];
matrix[i][j] = matrix[i][n - j - 1];
matrix[i][n - j - 1] = temp;
}
}
}

int main() {
int n;
scanf("%d", &n);

int **matrix = (int **)malloc(n * sizeof(int *));
for (int i = 0; i < n; i++) {
matrix[i] = (int *)malloc(n * sizeof(int));
}

for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
scanf("%d", &matrix[i][j]);
}
}

rotateMatrix(matrix, n);

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

for (int i = 0; i < n; i++) {
free(matrix[i]);
}
free(matrix);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
7 4 1
8 5 2
9 6 3
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
3 1
4 2
```

Compilation Status: Passed

Execution Time:

0.001s

37. Analyzing and Replacing Vowels in a StringProblem Statement: You need to implement a C function that replaces all vowels in a string with a specified character while considering the differences between using strings and character arrays. The function should handle cases where the input contains a mix of upper and lower-

case vowels.

Description: Write a C function void replace_vowels(char arr[], char replacement) th replaces all vowels (a, e, i, o, u, A, E, I, O, U) in the string stored in the character array arr with the character replacement. The function should differentiate between character arrays and strings and handle null termination correctly.

Input Format:

The first line contains a string stored in a character array arr (length ≤ 100).The second line contains a character replacement that will replace the vowels in the string.Output Format:

Output the modified string after replacing the vowels.

Sample Input:hello_world\0programming*

Sample Output:h*ll*_w*rld\0pr*gr*mm*ng

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void replace_vowels(char arr[], char replacement) {
for (int i = 0; arr[i] != '\0'; i++) {
char c = arr[i];
if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {
arr[i] = replacement;
}
}
}

int main() {
char arr[101];
char replacement;

fgets(arr, sizeof(arr), stdin);
scanf("%c", &replacement);

replace_vowels(arr, replacement);

printf("%s\n", arr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
h*ll*_w*rld\0pr*gr*mm*ng
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
B###t#f#l_C#d#\0
```

Compilation Status: Passed

Execution Time:

0.001s

38. Counting Words in a String Without Using Standard Library FunctionsProblem Statement: Write a C function that counts the number of words in a string. Words are sequences of characters separated by spaces. The function should handle multiple spaces between words and ensure that it does not use any standard library functions (e.g., strlen, strtok, etc.).

Description: Implement a function int count_words(char arr[]) that counts the numb
of words in the given character array arr. Words are defined as sequences of non-
space characters separated by spaces. The function should correctly handle leadin
trailing, and multiple spaces between words.

Input Format:

The input is a single line containing a string stored in a character array arr (length ≤
100).Output Format:

Output the number of words in the string.

Sample Input:hello world

Sample Output:2

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

int count_words(char arr[]) {
int count = 0, i = 0;
int in_word = 0;

while (arr[i] != '\0') {
if (arr[i] != ' ' && !in_word) {
in_word = 1;
count++;
} else if (arr[i] == ' ') {
in_word = 0;
}
i++;
}

return count;
}

int main() {
char arr[101];
fgets(arr, 101, stdin);

int word_count = count_words(arr);
printf("%d\n", word_count);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
2
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
4
```

Compilation Status: Passed

Execution Time:

0.001s

39. Manipulating Arrays Using Pointer ArithmeticProblem Statement:Write a C program that takes an array of integers and modifies it in the following way: For each element in the array, if the element is even, multiply it by the next element in the array using pointer arithmetic. If the element is odd, divide it by the previous element using pointer arithmetic. The last element should remain unchanged.

Description:Your task is to manipulate the elements of an integer array using pointer arithmetic. The program should accept an array of integers and process it as per the given rules. The operations should be performed directly on the memory addresses of

the elements using pointer arithmetic, and the modified array should be printed as t
output.

Input Format:The first line contains an integer n, representing the number of elemen
in the array.The second line contains n space-separated integers, representing the elements of the array.Output Format:A single line containing n space-separated integers, representing the modified array.

Sample Input:54 7 2 9 6

Sample Output:28 0 18 0 6

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void manipulate_array(int arr[], int n) {
for (int i = 0; i < n - 1; i++) {
if (*(arr + i) % 2 == 0) {
*(arr + i) = *(arr + i) * *(arr + i + 1);
} else {
*(arr + i) = *(arr + i) / *(arr + i - 1);
}
}
}

int main() {
int n;
scanf("%d", &n);
int arr[n];

for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

manipulate_array(arr, n);

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

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
main.c: In function ‘main’:
main.c:27:8: warning: missing terminating " character
printf("
^
main.c:27:8: error: missing terminating " character
main.c:28:1: warning: missing terminating " character
");
^
main.c:28:1: error: missing terminating " character
");
^~~
main.c:29:1: error: expected expression before ‘return’
return 0;
^~~~~~
main.c:30:1: error: expected ‘;’ before ‘}’ token
}
^
```

Compilation Error

Compilation Status: Failed

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
main.c: In function ‘main’:
main.c:27:8: warning: missing terminating " character
printf("
^
main.c:27:8: error: missing terminating " character
main.c:28:1: warning: missing terminating " character
```

```
");
^
main.c:28:1: error: missing terminating " character
");
^~~
main.c:29:1: error: expected expression before ‘return’
return 0;
^~~~~~
main.c:30:1: error: expected ‘;’ before ‘}’ token
}
^
```

Compilation Error

Compilation Status: Failed

40. Complex String Manipulation Using Pointer to PointerProblem Statement:Write a C program that reads a list of strings from the user and then performs the following operations using pointer-to-pointer (char **):

Reverse each string.Swap the first and last string in the list.Print the modified list.Description:The program should first read the number of strings and then the strings themselves. Use a pointer-to-pointer (char **) to manage the list of strings. After reversing each string, swap the first and last strings in the list. Finally, print the modified list of strings. The implementation should handle variable string lengths and ensure memory safety by properly allocating and freeing memory.

Input Format:

The first line contains an integer n, the number of strings.The next n lines each contain a single string.Output Format:

Print the modified list of strings, one per line.

Sample Input:3algorithmcodelanguage

Sample Output:egaugnaledocmhtirogla

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

```
#include <stdlib.h>
#include <string.h>

void reverseString(char *str) {
int start = 0, end = strlen(str) - 1;
while (start < end) {
char temp = str[start];
str[start] = str[end];
str[end] = temp;
start++;
end--;
}
}

int main() {
int n;
scanf("%d", &n);

char **arr = (char **)malloc(n * sizeof(char *));
for (int i = 0; i < n; i++) {
arr[i] = (char *)malloc(100 * sizeof(char));
scanf("%s", arr[i]);
}

for (int i = 0; i < n; i++) {
reverseString(arr[i]);
}

char *temp = arr[0];
arr[0] = arr[n - 1];
arr[n - 1] = temp;

for (int i = 0; i < n; i++) {
printf("%s\n", arr[i]);
}

for (int i = 0; i < n; i++) {
free(arr[i]);
}
free(arr);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
main.c: In function ‘main’:
main.c:35:16: warning: missing terminating " character
printf("%s
^
main.c:35:16: error: missing terminating " character
printf("%s
^~~
main.c:36:1: warning: missing terminating " character
", arr[i]);
^
main.c:36:1: error: missing terminating " character
", arr[i]);
^~~~~~~~~~~
main.c:37:5: error: expected expression before ‘}’ token
}
^
main.c:37:5: error: expected ‘;’ before ‘}’ token


Compilation Error
```

Compilation Status: Failed

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
main.c: In function ‘main’:
main.c:35:16: warning: missing terminating " character
printf("%s
^
main.c:35:16: error: missing terminating " character
printf("%s
^~~
main.c:36:1: warning: missing terminating " character
", arr[i]);
^
main.c:36:1: error: missing terminating " character
", arr[i]);
^~~~~~~~~~~
main.c:37:5: error: expected expression before ‘}’ token
```

```
}
^
main.c:37:5: error: expected ‘;’ before ‘}’ token
```

Compilation Error

Compilation Status: Failed

41. String Manipulation Using Pointer Arithmetic Problem Statement: Write a C program that takes a string as input and reorders its characters using pointer arithmetic. The program should alternate between moving the first character to the end and the last character to the start until the string is fully reordered. Description: You need to reorder the characters of a string based on a specific pattern using pointer arithmetic. Specifically, the program should start with the first character and move it to the end, then move the last character to the start, and repeat this pattern until all characters are reordered. The output should be the modified string. Input Format: A single line containing a string s of lowercase letters. Output Format: A single line containing the reordered string. Sample Input: abcdef Sample Output: bdfeca

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <string.h>
void reorderString(char *s) {
int len = strlen(s);
char oddChars[(len + 1) / 2];
char evenChars[len / 2];
int oddIndex = 0, evenIndex = 0;


for (int i = 0; i < len; i++) {
if (i % 2 == 0) {
evenChars[evenIndex++] = s[i];
} else {
oddChars[oddIndex++] = s[i];
}
}
int index = 0;
for (int j = 0; j < oddIndex; j++) {
s[index++] = oddChars[j];
}
for (int j = evenIndex - 1; j >= 0; j--) {
s[index++] = evenChars[j];
}
s[index] = '\0';
}
int main() {
char s[101];
scanf("%100s", s);
reorderString(s);
printf("%s\n", s);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
bdfeca
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
pniaeo
```

Compilation Status: Passed

Execution Time:

0.001s

42. Pointer Arithmetic with String ManipulationProblem Statement:Implement a C program that reads a string from the user and removes every third character from the string using pointer arithmetic. The program should then print the modified string.

Description:The program should take a single string input and manipulate it by removing every third character (starting from the third character, then the sixth, and so on). You should utilize pointer arithmetic to traverse and modify the string. Ensure that the string is null-terminated after modification. The program must handle strings of various lengths and should dynamically allocate memory as needed.

Input Format:

A single string s (maximum length of 100 characters).Output Format:

Print the modified string after removing every third character.

Sample Input:abcdefghijklmnopq

Sample Output:abdeghjkmnpq

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void removeEveryThirdChar(char *str) {
int len = strlen(str);
int count = 0;

for (int i = 2; i < len; i++) {
str[i - count] = str[i];
if ((i + 1) % 3 == 0) {
count++;
}
}
str[len - count] = '\0';
}

int main() {
char str[101];
fgets(str, 101, stdin);
str[strcspn(str, "\n")] = '\0';

removeEveryThirdChar(str);

printf("%s\n", str);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
abdeghjkmnpq
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
OpnAismain
```

Compilation Status: Passed

Execution Time:

0.001s

43. Understanding L-Value ModificationProblem Statement:Write C program that demonstrates the concept of L-value by creating a integer variable, assigning its address to a pointer, and modifying its value using the pointer. Then, dynamically allocate memory for another integer using malloc and modify its value, finally printing both integers.

Description:The program should start by declaring an integer variable and assigning it a value. Next, assign the address of this variable to a pointer and use the pointer to modify the variable's value. The program should then dynamically allocate memory in the heap for another integer, modify its value using a pointer, and print both the original integer (modified through the pointer) and the dynamically allocated integer.

Input Format:

Two integers: The first integer is the initial value for the integer variable, and the second integer is the value to be assigned to the dynamically allocated integer.Output Format:

Print the modified value of the original integer.Print the value stored in the dynamically allocated memory.

Sample Input:10 25

Sample Output:Original Integer: 15Heap Integer: 25

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int original;
int *ptr1;
int *heapPtr;
int initialValue, heapValue;

scanf("%d %d", &initialValue, &heapValue);

original = initialValue;

ptr1 = &original;
*ptr1 += 5;

heapPtr = (int *)malloc(sizeof(int));

if (heapPtr == NULL) {
return 1;
}

*heapPtr = heapValue;

printf("Original Integer: %d\n", original);
printf("Heap Integer: %d\n", *heapPtr);
free(heapPtr);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Original Integer: 5
Heap Integer: 100
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Original Integer: -5
Heap Integer: 50
```

Compilation Status: Passed

Execution Time:

0.001s

44. Dynamic Memory Allocation and L-Value ExpressionsProblem Statement:Create a C program that dynamically allocates memory for an array of integers on the heap. The program should read the array's size and elements from the user. Then, calculate the sum of all even-indexed elements and store the result in the first element of the array using L-value expressions.

Description:The program should prompt the user to enter the number of elements and then read these elements into a dynamically allocated array. It should then compute the sum of all elements located at even indices (0, 2, 4, etc.) and store theresult in the first element of the array using L-value expressions. Finally, the program should print the modified array.

Input Format:First, an integer representing the size of the array.Then, n integers representing the elements of the array.

Output Format:Print the modified array, with the first element now holding the sum of all even-indexed elements.

Sample Input:51 2 3 4 5

Sample Output:9 2 3 4 5

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int n;
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));

for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

int sum = 0;
for (int i = 0; i < n; i += 2) {
sum += arr[i];
}
arr[0] = sum;

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

free(arr);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
9 2 3 4 5
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
8 3 5 7
```

Compilation Status: Passed

Execution Time:

0.001s

45. Memory Leak Detection with Dynamic ArraysProblem Statement:Write a C program that reads a series of integers into a dynamically allocated array. After reading the integers, the prograı should create a copy of the array, double the size of the original array, and then replace the original array with this new array. The program should print both the original and copied arrays before deallocating the memory. Ensure that you avoid memory leaks by freeing allocated memory appropriately.

Description:The program should:

Dynamically allocate memory for an array to store integers.Copy the contents of the original array into a new, larger array.Print both the original and the new array.Ensure proper memory management by freeing all dynamically allocated memory.Input Format:

An integer n, the number of elements in the array.n integers to populate the array.Output Format:

Print the original array.Print the copied array.

Sample Input:31 2 3

Sample Output:Original array: 1 2 3 Copied array: 1 2 3 0 0 0

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

int main() {
int n;
scanf("%d", &n);

int *original = (int *)malloc(n * sizeof(int));
if (original == NULL) {
printf("Memory allocation failed\n");
return 1;
}

for (int i = 0; i < n; i++) {
scanf("%d", &original[i]);
}
int *copied = (int *)malloc(2 * n * sizeof(int));
if (copied == NULL) {
printf("Memory allocation failed\n");
free(original);
return 1;
}

for (int i = 0; i < n; i++) {
copied[i] = original[i];
}

for (int i = n; i < 2 * n; i++) {
copied[i] = 0;
}

printf("Original array: ");
for (int i = 0; i < n; i++) {
printf("%d ", original[i]);
}
printf("\n");

printf("Copied array: ");
for (int i = 0; i < 2 * n; i++) {
printf("%d ", copied[i]);
}
printf("\n");

free(original);
free(copied);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Original array: 1 2 3
Copied array: 1 2 3 0 0 0
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Original array: 4 5 6 7
Copied array: 4 5 6 7 0 0 0 0
```

Compilation Status: Passed

Execution Time:

0.001s

46. Pointer Arithmetic and Dynamic Array ModificationProblem Statement:Write a C program that uses pointers to manipulate a dynamically allocated array. The program should:

Allocate memory for an array of n integers.Initialize the array with values such that each element is the square of its index.Create a function that takes a pointer to the array and performs the following:Increase each element by the value of the next element, except for the last element.Swap the first and last elements of the array.Print the modified array.Description:Use pointers to manage and modify the array. Ensure that you use the address-of operator (&) to pass pointers to functions and the value-at operator (*) to manipulate array elements. Handle memory allocation and deallocation properly.

Input Format:

An integer n indicating the number of elements in the array.Output Format:

Print the modified array, with each element separated by a space.

Sample Input:5

Sample Output:16 5 13 25 1

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void modifyArray(int *arr, int n) {
for (int i = 0; i < n - 1; i++) {
arr[i] += arr[i + 1];
}

int temp = arr[0];
arr[0] = arr[n - 1];
arr[n - 1] = temp;
}

int main() {
int n;
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));

for (int i = 0; i < n; i++) {
arr[i] = i * i;
}

modifyArray(arr, n);

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

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
16 5 13 25 1
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
4 5 1
```

Compilation Status: Passed

Execution Time:

0.001s

47. Pointer to Pointer and Multi-Dimensional ArrayProblem Statement:Write a C program that creates a dynamically allocated 2D array (matrix) using pointers to pointers. The program should:

Allocate memory for a matrix of size m x n.Initialize the matrix such that each element matrix[i][j] is the product of its indices (i * j).Create a function that takes a pointer to the matrix and:Transposes the matrix (swap rows and columns).Prints the transposed matrix.Description:Use pointer-to-pointer (int **) to manage the 2D array. Implement the transpose function by correctly accessing and modifying the matrix elements using the value-at (*) and address-of (&) operators.

Input Format:

Two integers m and n, indicating the number of rows and columns in the matrix.Output Format:

Print the transposed matrix, with each row on a new line.

Sample Input:2 3

Sample Output:0 0 0 1 0 2

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void transposeMatrix(int **matrix, int m, int n) {
int **transposed = (int **)malloc(n * sizeof(int *));
for (int i = 0; i < n; i++) {
transposed[i] = (int *)malloc(m * sizeof(int));
}

for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) {
transposed[j][i] = matrix[i][j];
}
}

for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
printf("%d ", transposed[i][j]);
}
printf("\n");
}

for (int i = 0; i < n; i++) {
free(transposed[i]);
}
free(transposed);
}

int main() {
int m, n;
scanf("%d %d", &m, &n);

int **matrix = (int **)malloc(m * sizeof(int *));
for (int i = 0; i < m; i++) {
matrix[i] = (int *)malloc(n * sizeof(int));
}

for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) {
matrix[i][j] = i * j;
}
}

transposeMatrix(matrix, m, n);

for (int i = 0; i < m; i++) {
free(matrix[i]);
}
free(matrix);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
0 0
0 1
0 2
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
0 0 0
0 1 2
```

Compilation Status: Passed

Execution Time:

0.001s

48. L-value with Arrays and PointersProblem Statement:Write a C program to illustrate the behavior of L-values with arrays and pointers. The program should:

Define an array of integers and use a pointer to traverse and modify it.Use a function to modify the array by passing the array name (which is a pointer) and demonstrate how array elements can be modified using pointer arithmetic.Description:This question examines how arrays (which are L-values) and pointers interact. You will demonstrate how changes made through a pointer affect the original array and how array names can be used as pointers.

Input Format:

An integer n for the size of the array, followed by n integers.Output Format:

Print the array elements before and after modification.

Sample Input:4 1 2 3 4

Sample Output:Array before modification:1 2 3 4 Array after modification:5 6 7 8

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

void modify_array(int *arr, int n) {
for (int i = 0; i < n; i++) {
arr[i] += 4;
}
}

int main() {
int n;
scanf("%d", &n);

int arr[n];
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}

printf("Array before modification:\n");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
printf("\n");

modify_array(arr, n);
printf("Array after modification:\n");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
printf("\n");
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Array before modification:
1 2 3 4
Array after modification:
5 6 7 8
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Array before modification:
7 1 2 3 4
Array after modification:
11 5 6 7 8
```

Compilation Status: Passed

Execution Time:

0.001s

49. Matrix Multiplication with Dynamic AllocationProblem Statement:

Write a program that dynamically allocates memory for two matrices, performs matrix multiplication, and then frees the allocated memory. The matrices' sizes should be taken from user input, and you must handle errors such as mismatched dimensions.

Description:

Dynamically allocate memory for two matrices based on user input.Perform matrix multiplication.Ensure proper memory management using malloc, calloc, realloc, an free.Handle cases where the matrices cannot be multiplied due to incompatible dimensions.Input Format:

The dimensions of the first matrix R1 x C1.The dimensions of the second matrix R2 x C2.R1 must be equal to R2 for matrix multiplication.Followed by the elements of both matrices.Output Format:

Print the resulting matrix from the multiplication.

Sample Input:2 33 21 2 34 5 67 89 10

Sample Output:25 28 73 82

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

void multiplyMatrices(int **matrix1, int **matrix2, int **result, int R1, int C1, int R2, int
C2) {
for (int i = 0; i < R1; i++) {
for (int j = 0; j < C2; j++) {
result[i][j] = 0;
for (int k = 0; k < C1; k++) {
result[i][j] += matrix1[i][k] * matrix2[k][j];
}
}
}
}

int main() {
int R1, C1, R2, C2;
scanf("%d %d", &R1, &C1);
scanf("%d %d", &R2, &C2);

if (C1 != R2) {
printf("Matrix multiplication is not possible due to incompatible dimensions.\n");
return 1;
}

int **matrix1 = (int **)malloc(R1 * sizeof(int *));
for (int i = 0; i < R1; i++) {
matrix1[i] = (int *)malloc(C1 * sizeof(int));
}

int **matrix2 = (int **)malloc(R2 * sizeof(int *));
for (int i = 0; i < R2; i++) {
matrix2[i] = (int *)malloc(C2 * sizeof(int));
}

int **result = (int **)malloc(R1 * sizeof(int *));
for (int i = 0; i < R1; i++) {
result[i] = (int *)malloc(C2 * sizeof(int));
}

for (int i = 0; i < R1; i++) {
for (int j = 0; j < C1; j++) {
scanf("%d", &matrix1[i][j]);
}
}

for (int i = 0; i < R2; i++) {
for (int j = 0; j < C2; j++) {
scanf("%d", &matrix2[i][j]);
}
}

multiplyMatrices(matrix1, matrix2, result, R1, C1, R2, C2);

for (int i = 0; i < R1; i++) {
for (int j = 0; j < C2; j++) {
printf("%d ", result[i][j]);
}
printf("\n");
}

for (int i = 0; i < R1; i++) {
free(matrix1[i]);
}
free(matrix1);

for (int i = 0; i < R2; i++) {
free(matrix2[i]);
}
free(matrix2);

for (int i = 0; i < R1; i++) {
free(result[i]);
}
free(result);

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
25 28
73 82
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
27 30 33
61 68 75
95 106 117
```

Compilation Status: Passed

Execution Time:

0.001s

50. Dynamic Memory Allocation for Circular BufferProblem Statement:

Create a circular buffer using dynamic memory allocation. The buffer should support adding elements, removing elements, and checking if it is full or empty. You must handle the dynamic resizing of the buffer when it is full or if it is resized.

Description:

Implement a circular buffer that can grow and shrink dynamically.Handle cases where

the buffer needs resizing using realloc.Properly manage memory using malloc, call
realloc, and free.Input Format:

An integer n indicating the number of operations.For each operation, you will receive:An operation type (ADD, REMOVE, RESIZE).Depending on the operation, additional parameters such as the value to add, or the new size for resizing.Output Format:

After each operation, print the current state of the buffer.

Sample Input:6ADD 5ADD 10RESIZE 5ADD 15REMOVEADD 20

Sample Output:5 5 10 5 10 5 10 15 10 15 10 15 20

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
int *buffer;
int size;
int front;
int rear;
int count;
} CircularBuffer;
void initBuffer(CircularBuffer *cb, int size) {
cb->size = size;
cb->buffer = (int *)malloc(size * sizeof(int));
cb->front = 0;
cb->rear = 0;
cb->count = 0;
}
void freeBuffer(CircularBuffer *cb) {
free(cb->buffer);
}
int isFull(CircularBuffer *cb) {
return cb->count == cb->size;
}
int isEmpty(CircularBuffer *cb) {
return cb->count == 0;
}
void addElement(CircularBuffer *cb, int element) {
if (isFull(cb)) {
printf("Buffer is full. Unable to add element.\n");
```

```
return;
}
cb->buffer[cb->rear] = element;
cb->rear = (cb->rear + 1) % cb->size;
cb->count++;
}
void removeElement(CircularBuffer *cb) {
if (isEmpty(cb)) {
printf("Buffer is empty. Unable to remove element.\n");
return;
}
cb->front = (cb->front + 1) % cb->size;
cb->count--;
}
void resizeBuffer(CircularBuffer *cb, int newSize) {
int *newBuffer = (int *)malloc(newSize * sizeof(int));
int i, j;
for (i = cb->front, j = 0; j < cb->count; j++) {
newBuffer[j] = cb->buffer[i];
i = (i + 1) % cb->size;
}
free(cb->buffer);
cb->buffer = newBuffer;
cb->size = newSize;
cb->front = 0;
cb->rear = cb->count;
}
void printBuffer(CircularBuffer *cb) {
if (isEmpty(cb)) {
printf("Buffer is empty.\n");
return;
}
int i = cb->front;
for (int j = 0; j < cb->count; j++) {
printf("%d ", cb->buffer[i]);
i = (i + 1) % cb->size;
}
printf("\n");
}
int main() {
int n, newSize, value;
char operation[10];
scanf("%d", &n);
CircularBuffer cb;
initBuffer(&cb, 10);
for (int i = 0; i < n; i++) {
scanf("%s", operation);
if (strcmp(operation, "ADD") == 0) {
scanf("%d", &value);
addElement(&cb, value);
} else if (strcmp(operation, "REMOVE") == 0) {
removeElement(&cb);
} else if (strcmp(operation, "RESIZE") == 0) {
scanf("%d", &newSize);
resizeBuffer(&cb, newSize);
}
printBuffer(&cb);
}
freeBuffer(&cb);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
5
5 10
5 10
5 10 15
10 15
10 15 20
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
1
1 2
1 2
2
```

Compilation Status: Passed

Execution Time:

0.001s

51. Dynamic Sparse Matrix RepresentationQuestion: Implement a dynamic sparse matrix using linked lists. The matrix should support insertion, deletion, and retrieval of elements at specified positions. Each non-zero element should be stored with its row, column, and value.

Description: Create a sparse matrix where only non-zero elements are stored to save memory. Implement functions to:

Insert a new element.Delete an existing element.Retrieve an element by its row and column.Input Format:

The first line contains an integer n, the number of operations.Each of the next n lines contains one of the following operations:INSERT row col valueDELETE row colRETRIEVE row colOutput Format:

For RETRIEVE, output the value of the element at the specified position or "0" if the element does not exist.

Sample Input:5INSERT 2 3 5INSERT 1 1 10RETRIEVE 2 3DELETE 2 3RETRIEVE 2 3

Sample Output:50

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
int row, col, value;
struct Node *next;
} Node;
Node *head = NULL;
void insert(int row, int col, int value) {
Node *newNode = (Node *)malloc(sizeof(Node));
newNode->row = row;
newNode->col = col;
newNode->value = value;
newNode->next = NULL;
if (!head) {
head = newNode;
return;
```

```
}
Node *temp = head;
while (temp->next) {
if (temp->row == row && temp->col == col) {
temp->value = value;
free(newNode);
return;
}
temp = temp->next;
}
temp->next = newNode;
}
void delete(int row, int col) {
Node *temp = head, *prev = NULL;
while (temp) {
if (temp->row == row && temp->col == col) {
if (prev) {
prev->next = temp->next;
} else {
head = temp->next;
}
free(temp);
return;
}
prev = temp;
temp = temp->next;
}
}
int retrieve(int row, int col) {
Node *temp = head;
while (temp) {
if (temp->row == row && temp->col == col) {
return temp->value;
}
temp = temp->next;
}
return 0;
}
int main() {
int n;
scanf("%d", &n);
for (int i = 0; i < n; i++) {
char operation[10];
int row, col, value;
scanf("%s", operation);
if (operation[0] == 'I') {
scanf("%d %d %d", &row, &col, &value);
insert(row, col, value);
} else if (operation[0] == 'D') {
scanf("%d %d", &row, &col);
delete(row, col);
} else if (operation[0] == 'R') {
scanf("%d %d", &row, &col);
printf("%d\n", retrieve(row, col));
}
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
5
0
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
1
0
```

Compilation Status: Passed

Execution Time:

0.001s

52. Dynamic Memory Allocation for GraphsQuestion: Implement a dynamic graph representation using adjacency lists. The graph should support adding vertices, adding edges, and printing the adjacency list of each vertex.

Description: Create a graph with dynamic memory allocation for both vertices and edges. Implement functions to:

Add a vertex.Add an edge between two vertices.Print the adjacency list of the graph.Input Format:

The first line contains an integer n, the number of operations.Each of the next n lines contains one of the following operations:ADD_VERTEX vertexADD_EDGE vertex1 vertex2PRINT_ADJ_LISTOutput Format:

For PRINT_ADJ_LIST, output the adjacency list for each vertex in the format: vertex: [adj1, adj2, ...].

Sample Input:5ADD_VERTEX 0ADD_VERTEX 1ADD_EDGE 0 1PRINT_ADJ_LIST

Sample Output:0: [1]0: [1]

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

typedef struct Node {
int vertex;
struct Node* next;
} Node;

typedef struct Graph {
int numVertices;
Node** adjLists;
} Graph;

Graph* createGraph(int vertices) {
Graph* graph = malloc(sizeof(Graph));
graph->numVertices = vertices;
graph->adjLists = malloc(vertices * sizeof(Node*));

for (int i = 0; i < vertices; i++) {
graph->adjLists[i] = NULL;
}
return graph;
}

void addVertex(Graph* graph, int vertex) {
```

```
if (vertex >= graph->numVertices) {
graph->adjLists = realloc(graph->adjLists, (vertex + 1) * sizeof(Node*));
for (int i = graph->numVertices; i <= vertex; i++) {
graph->adjLists[i] = NULL;
}
graph->numVertices = vertex + 1;
}
}

void addEdge(Graph* graph, int src, int dest) {
Node* newNode = malloc(sizeof(Node));
newNode->vertex = dest;
newNode->next = graph->adjLists[src];
graph->adjLists[src] = newNode;
}

void printAdjList(Graph* graph) {
for (int i = 0; i < graph->numVertices; i++) {
Node* temp = graph->adjLists[i];
if (temp) {
printf("%d: [", i);
while (temp) {
printf("%d", temp->vertex);
temp = temp->next;
if (temp) printf(" ");
}
printf("]\n");
}
}
}

void processOperations(int numOperations) {
Graph* graph = createGraph(0);

for (int i = 0; i < numOperations; i++) {
char operation[20];
scanf("%s", operation);

if (strcmp(operation, "ADD_VERTEX") == 0) {
int vertex;
scanf("%d", &vertex);
addVertex(graph, vertex);
} else if (strcmp(operation, "ADD_EDGE") == 0) {
int src, dest;
scanf("%d %d", &src, &dest);
addEdge(graph, src, dest);
} else if (strcmp(operation, "PRINT_ADJ_LIST") == 0) {
printAdjList(graph);
}
}

for (int i = 0; i < graph->numVertices; i++) {
Node* temp = graph->adjLists[i];
while (temp) {
```

```
Node* toFree = temp;
temp = temp->next;
free(toFree);
}
}
free(graph->adjLists);
free(graph);
}

int main() {
int numOperations;
scanf("%d", &numOperations);
processOperations(numOperations);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
0: [1]
0: [1]
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
2: [3]
3: [2]
2: [3]
3: [2]
```

Compilation Status: Passed

Execution Time:

0.001s

53. Structure-Based Sparse Matrix RepresentationProblem Statement:You are required to implement a structure-based sparse matrix representation. A sparse matrix is one in which the majority of the elements are zero. Your task is to store only the non-zero elements using an array of structures and then perform basic matrix operations (addition and subtraction) on the sparse matrices.

Description:Define a structure SparseMatrixElement that contains three fields: row, col, and value, representing the row index, column index, and the value of the non-zero element in the matrix, respectively.Implement functions to:Create a sparse matrix by reading user inputs and storing only non-zero elements.Add two sparse matrices and return the result as a new sparse matrix.Subtract one sparse matrix from another and return the result as a new sparse matrix.Input Format:First line: An integer n, the number of non-zero elements in the first matrix.Next n lines: Three integers representing row, col, and value for each non-zero element of the first matrix.Followed by the same format for the second matrix.Output Format:Print the resultant sparse matrix after performing addition and subtraction operations.

Sample Input:30 1 51 2 82 0 630 1 31 0 72 2 4

Sample Output:Addition Result:0 1 81 0 71 2 82 0 62 2 4

Subtraction Result:0 1 21 0 -71 2 82 0 6

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>

#define MAX 100

// Define a structure for non-zero matrix elements
typedef struct {
int row;
int col;
int value;
```

```
} SparseMatrixElement;

// Function to read a sparse matrix from input
void readMatrix(SparseMatrixElement matrix[], int *n) {
scanf("%d", n); // Read number of non-zero elements
for (int i = 0; i < *n; i++) {
scanf("%d %d %d", &matrix[i].row, &matrix[i].col, &matrix[i].value);
}
}

// Function to add two sparse matrices
int addMatrices(SparseMatrixElement matrix1[], int n1, SparseMatrixElement
matrix2[], int n2, SparseMatrixElement result[]) {
int i = 0, j = 0, k = 0;

// Iterate through both matrices
while (i < n1 && j < n2) {
if (matrix1[i].row == matrix2[j].row && matrix1[i].col == matrix2[j].col) {
// Add the values of the matching elements
int sum = matrix1[i].value + matrix2[j].value;
if (sum != 0) { // Only store non-zero results
result[k].row = matrix1[i].row;
result[k].col = matrix1[i].col;
result[k].value = sum;
k++;
}
i++;
j++;
} else if (matrix1[i].row < matrix2[j].row || (matrix1[i].row == matrix2[j].row &&
matrix1[i].col < matrix2[j].col)) {
result[k++] = matrix1[i++];
} else {
result[k++] = matrix2[j++];
}
}

// Add remaining elements from matrix1
while (i < n1) {
result[k++] = matrix1[i++];
}
// Add remaining elements from matrix2
while (j < n2) {
result[k++] = matrix2[j++];
}

return k; // Return the number of non-zero elements in the result
}

// Function to subtract two sparse matrices
int subtractMatrices(SparseMatrixElement matrix1[], int n1, SparseMatrixElement
matrix2[], int n2, SparseMatrixElement result[]) {
int i = 0, j = 0, k = 0;

// Iterate through both matrices
```

```
while (i < n1 && j < n2) {
if (matrix1[i].row == matrix2[j].row && matrix1[i].col == matrix2[j].col) {
// Subtract values when both matrices have the same element
int diff = matrix1[i].value - matrix2[j].value;
if (diff != 0) { // Only store non-zero results
result[k].row = matrix1[i].row;
result[k].col = matrix1[i].col;
result[k].value = diff;
k++;
}
i++;
j++;
} else if (matrix1[i].row < matrix2[j].row || (matrix1[i].row == matrix2[j].row &&
matrix1[i].col < matrix2[j].col)) {
// If matrix1 has a smaller element, treat missing element in matrix 2 as 0
result[k++] = matrix1[i++];
} else {
// If matrix2 has a smaller element, treat missing element in matrix 1 as 0 and
subtract
result[k].row = matrix2[j].row;
result[k].col = matrix2[j].col;
result[k].value = -matrix2[j].value; // Subtract element from matrix2 (i.e., treat matrix1
as 0)
k++;
j++;
}
}

// Add remaining elements from matrix1
while (i < n1) {
result[k++] = matrix1[i++];
}
return k; // Return the number of non-zero elements in the result
}

// Function to print a sparse matrix
void printMatrix(SparseMatrixElement matrix[], int n) {
for (int i = 0; i < n; i++) {
printf("%d %d %d", matrix[i].row, matrix[i].col, matrix[i].value);
if (i < n - 1) {
printf("\n"); // Print newline for all but the last element
}
}
}
int main() {
SparseMatrixElement matrix1[MAX], matrix2[MAX], result[MAX];
int n1, n2, nResult;

// Read the first matrix
readMatrix(matrix1, &n1);

// Read the second matrix
readMatrix(matrix2, &n2);
```

```
// Add the matrices
nResult = addMatrices(matrix1, n1, matrix2, n2, result);

// Print the addition result
printf("Addition Result:\n");
printMatrix(result, nResult);

// Subtract the matrices
nResult = subtractMatrices(matrix1, n1, matrix2, n2, result);

// Print the subtraction result
printf("\n\nSubtraction Result:\n");
printMatrix(result, nResult);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Addition Result:
0 1 8
1 0 7
1 2 8
2 0 6
2 2 4

Subtraction Result:
0 1 2
1 0 -7
1 2 8
2 0 6
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

```
< hidden >
```

Output:

```
Addition Result:
0 2 15
2 1 20

Subtraction Result:
0 2 5
2 1 10
```

Compilation Status: Passed

Execution Time:

0.001s

54. Complex Number Operations Using StructuresProblem Statement:Implement a program that handles operations on complex numbers using structures. The program should allow for addition, subtraction, multiplication, and division of complex numbers.

Description:Define a structure ComplexNumber with two fields real and imaginary representing the real and imaginary parts of a complex number.Implement functions to:Add two complex numbers.Subtract one complex number from another.Multiply two complex numbers.Divide one complex number by another.Input Format:First line: Two integers representing the real and imaginary parts of the first complex number.Second line: Two integers representing the real and imaginary parts of the second complex number.Output Format:Print the results of the addition, subtraction, multiplication, and division operations.

Sample Input:3 41 2

Sample Output:Addition: 4.00 + 6.00iSubtraction: 2.00 + 2.00iMultiplication: -5.00 + 10.00iDivision: 2.20 - 0.40i

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

```
struct ComplexNumber {
float real;
float imaginary;
};
struct ComplexNumber add(struct ComplexNumber c1,
struct ComplexNumber c2) {
struct ComplexNumber result;
result.real = c1.real + c2.real;
result.imaginary = c1.imaginary + c2.imaginary;
return result;
}
struct ComplexNumber subtract(struct ComplexNumber c1,
struct ComplexNumber c2) {
struct ComplexNumber result;
result.real = c1.real - c2.real;
result.imaginary = c1.imaginary - c2.imaginary;
return result;
}
struct ComplexNumber multiply(struct ComplexNumber c1,
struct ComplexNumber c2) {
struct ComplexNumber result;
result.real = c1.real * c2.real - c1.imaginary * c2.imaginary;
result.imaginary = c1.real * c2.imaginary + c1.imaginary *
c2.real;
return result;
}
struct ComplexNumber divide(struct ComplexNumber c1,
struct ComplexNumber c2) {
struct ComplexNumber result;
float denominator = c2.real * c2.real + c2.imaginary *
c2.imaginary;
if (denominator == 0) {
printf("Division by zero is not possible.\n");
result.real = 0;
result.imaginary = 0;
return result;
}
result.real = (c1.real * c2.real + c1.imaginary *
c2.imaginary) / denominator;
result.imaginary = (c1.imaginary * c2.real - c1.real *
c2.imaginary) / denominator;
return result;
}
void printComplex(struct ComplexNumber c) {
if (c.imaginary >= 0) {
printf("%.2f + %.2fi\n", c.real, c.imaginary);
} else {
printf("%.2f - %.2fi\n", c.real, -c.imaginary);
}
}
int main() {
struct ComplexNumber c1, c2, result;
scanf("%f %f", &c1.real, &c1.imaginary);
scanf("%f %f", &c2.real, &c2.imaginary);
result = add(c1, c2);
printf("Addition: ");
printComplex(result);
result = subtract(c1, c2);
printf("Subtraction: ");
printComplex(result);
result = multiply(c1, c2);
printf("Multiplication: ");
printComplex(result);
result = divide(c1, c2);
printf("Division: ");
printComplex(result);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Addition: 4.00 + 6.00i
Subtraction: 2.00 + 2.00i
Multiplication: -5.00 + 10.00i
Division: 2.20 - 0.40i
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Addition: 7.00 + 9.00i
Subtraction: 3.00 + 3.00i
Multiplication: -8.00 + 27.00i
```

Division: 2.15 - 0.23i

Compilation Status: Passed

Execution Time:

0.001s

55. Nested Structures and Dynamic Memory AllocationProblem Statement:Write a C program to manage a list of students, each with multiple courses, and each course with multiple assessments. The structure must support dynamic memory allocation for courses and assessments. The program should allow the user to input data and then calculate and display the average score for each course and the overall average for each student.

Description:You are required to create a program that utilizes nested structures to manage student data. Each student can be enrolled in multiple courses, and each course can have several assessments. The number of courses and assessments are determined at runtime by user input. After the data is input, calculate the average score for each course and the overall average score for each student. Use dynamic memory allocation to handle the variable number of courses and assessments.

Input Format:The first input is an integer n representing the number of students.For each student:An integer c representing the number of courses.For each course:An integer a representing the number of assessments.a integers representing the scores of the assessments.Output Format:For each student, output the average score for each course.Output the overall average score for each student.

Sample Input:22380 90 85275 801288 92

Sample Output:Student 1:Course 1 Average: 85.00Course 2 Average: 77.50Overall Average: 81.25Student 2:Course 1 Average: 90.00Overall Average: 90.00

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
int score;
} Assessment;
typedef struct {
```

```
int numAssessments;
Assessment *assessments;
} Course;
typedef struct {
int numCourses;
Course *courses;
} Student;
float calculateCourseAverage(Course course) {
int totalScore = 0;
for (int i = 0; i < course.numAssessments; i++) {
totalScore += course.assessments[i].score;
}
return (float)totalScore / course.numAssessments;
}
float calculateOverallAverage(Student student) {
float totalCourseAverage = 0;
for (int i = 0; i < student.numCourses; i++) {
totalCourseAverage +=
calculateCourseAverage(student.courses[i]);
}
return totalCourseAverage / student.numCourses;
}
int main() {
int n;
scanf("%d", &n);
Student *students = (Student *)malloc(n * sizeof(Student));
for (int i = 0; i < n; i++) {
scanf("%d", &students[i].numCourses);
students[i].courses = (Course
*)malloc(students[i].numCourses * sizeof(Course));
for (int j = 0; j < students[i].numCourses; j++) {
scanf("%d",
&students[i].courses[j].numAssessments);
students[i].courses[j].assessments = (Assessment
*)malloc(students[i].courses[j].numAssessments *
sizeof(Assessment));
for (int k = 0; k <
students[i].courses[j].numAssessments; k++) {
scanf("%d",
&students[i].courses[j].assessments[k].score);
}
}
}
for (int i = 0; i < n; i++) {
printf("Student %d:\n", i + 1);
float overallAverage = 0;
for (int j = 0; j < students[i].numCourses; j++) {
float courseAverage =
calculateCourseAverage(students[i].courses[j]);
printf("Course %d Average: %.2f\n", j + 1,
courseAverage);
overallAverage += courseAverage;
}
overallAverage /= students[i].numCourses;
printf("Overall Average: %.2f\n", overallAverage);
}
for (int i = 0; i < n; i++) {
for (int j = 0; j < students[i].numCourses; j++) {
free(students[i].courses[j].assessments);
}
free(students[i].courses);
}
free(students);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Student 1:
Course 1 Average: 85.00
Course 2 Average: 77.50
Overall Average: 81.25
Student 2:
Course 1 Average: 90.00
Overall Average: 90.00
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Student 1:
Course 1 Average: 92.50
Overall Average: 92.50
```

Compilation Status: Passed

Execution Time:

0.001s

56. Handling Complex Data in Structures with PointersProblem Statement:Design a C program that simulates an inventory management system for a store. Each product in the store can have multiple suppliers, and each supplier can offer different prices for the product. The program must calculate the average price of each product across all suppliers and determine the cheapest supplier for each product.

Description:You need to create a program that manages a list of products, each with multiple suppliers. For each product, the user will input the number of suppliers and the price offered by each supplier. The program should calculate and display the average price for each product and identify the cheapest supplier. Use structures to represent the products and suppliers, and handle the input dynamically.

Input Format:The first input is an integer p representing the number of products.For each product:A string product_name representing the name of the product.An integer s representing the number of suppliers.For each supplier:A string supplier_name representing the name of the supplier.A float price representing the price offered by the supplier.Output Format:For each product, output the average price across all suppliers.Output the name of the cheapest supplier for each product.

Sample Input:2Laptop3SupplierA 1000.50SupplierB 980.75SupplierC 1025.00Phone2SupplierX 500.00SupplierY 495.50

Sample Output:Product: LaptopAverage Price: 1002.08Cheapest Supplier: SupplierBProduct: PhoneAverage Price: 497.75Cheapest Supplier: SupplierY

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

struct Supplier {
char name[50];
float price;
```

```
};

struct Product {
char name[50];
int numSuppliers;
struct Supplier *suppliers;
};

float calculateAveragePrice(struct Product *product) {
float total = 0.0;
for (int i = 0; i < product->numSuppliers; i++) {
total += product->suppliers[i].price;
}
return total / product->numSuppliers;
}

void findCheapestSupplier(struct Product *product) {
int cheapestIndex = 0;
for (int i = 1; i < product->numSuppliers; i++) {
if (product->suppliers[i].price < product->suppliers[cheapestIndex].price) {
cheapestIndex = i;
}
}
printf("Cheapest Supplier: %s\n", product->suppliers[cheapestIndex].name);
}

int main() {
int p;
scanf("%d", &p);

struct Product *products = (struct Product *)malloc(p * sizeof(struct Product));

for (int i = 0; i < p; i++) {
scanf("%s", products[i].name);
scanf("%d", &products[i].numSuppliers);
products[i].suppliers = (struct Supplier *)malloc(products[i].numSuppliers *
sizeof(struct Supplier));

for (int j = 0; j < products[i].numSuppliers; j++) {
scanf("%s %f", products[i].suppliers[j].name, &products[i].suppliers[j].price);
}
}

for (int i = 0; i < p; i++) {
printf("Product: %s\n", products[i].name);
float avgPrice = calculateAveragePrice(&products[i]);
printf("Average Price: %.2f\n", avgPrice);
findCheapestSupplier(&products[i]);
}

for (int i = 0; i < p; i++) {
free(products[i].suppliers);
}
free(products);
```

```
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Product: Laptop
Average Price: 1002.08
Cheapest Supplier: SupplierB
Product: Phone
Average Price: 497.75
Cheapest Supplier: SupplierY
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Product: Printer
Average Price: 147.75
Cheapest Supplier: Supplier2
```

Compilation Status: Passed

Execution Time:

0.001s

57. Complex Macro with Multiple ParametersProblem Statement:

Write a C program that uses a complex macro with multiple parameters to perform series of operations on two integers. The macro should compute the result of a mathematical expression and print both intermediate and final results. Use the mac to compute the following expression: (a * b + c) / d, where a, b, c, and d are provide by the user.

Description:

Define a macro CALC(a, b, c, d) that calculates the result of (a * b + c) / d.Prompt the user for values of a, b, c, and d.Use the macro to perform the calculation and print the intermediate and final results.Input Format:

Four integers: a, b, c, and d.Output Format:

Intermediate result of a * b.Intermediate result of a * b + c.Final result of (a * b + c) / d.

Sample Input: 2342

Sample Output:Intermediate result of a * b: 6Intermediate result of a * b + c: 10Final result of (a * b + c) / d: 5

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#define CALC(a, b, c, d) (((a) * (b) + (c)) / (d))
int main() {
int a, b, c, d;
scanf("%d %d %d %d", &a, &b, &c, &d);
int intermediate1 = a * b;
int intermediate2 = intermediate1 + c;
int finalResult = CALC(a, b, c, d);
printf("Intermediate result of a * b: %d\n", intermediate1);
printf("Intermediate result of a * b + c: %d\n",
intermediate2);
printf("Final result of (a * b + c) / d: %d\n", finalResult);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Intermediate result of a * b: 6
Intermediate result of a * b + c: 10
Final result of (a * b + c) / d: 5
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Intermediate result of a * b: 30
Intermediate result of a * b + c: 37
Final result of (a * b + c) / d: 12
```

Compilation Status: Passed

Execution Time:

0.001s

58. Bank Account Management SystemProblem Statement:Design a program to manage bank accounts using structures and unions. The program should handle two types of accounts: Savings Account and Checking Account. Each account type has different attributes. The program should calculate and display the account balance based on the type of account.

Description:Define a union AccountDetails that contains:

savings structure with:

balance (float) for the current balance.interestRate (float) for the annual interest rate.checking structure with:

balance (float) for the current balance.overdraftLimit (float) for the overdraft limit.Define a structure BankAccount that includes:

A string accountHolderName for the name of the account holder.An integer accountType (1 for Savings Account, 2 for Checking Account).An AccountDetails union.The program should:

For Savings Account: Display the balance and interest rate.For Checking Account: Display the balance and overdraft limit.Input Format:The first line contains the name of the account holder.The second line contains an integer representing the account type (1 for Savings Account, 2 for Checking Account).If the account type is 1, the next line contains two floats: balance and interest rate.If the account type is 2, the next line contains two floats: balance and overdraft limit.Output Format:Display the account details based on the account type.

Sample Input:Alice Johnson11500.75 2.5

Sample Output:Savings Account Balance: 1500.75Annual Interest Rate: 2.50%

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

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

union AccountDetails {
struct {
float balance;
float interestRate;
} savings;
struct {
float balance;
float overdraftLimit;
} checking;
};

struct BankAccount {
char accountHolderName[100];
int accountType;
union AccountDetails accountDetails;
};

int main() {
struct BankAccount account;

// Corrected fgets to read the account holder's name
fgets(account.accountHolderName, sizeof(account.accountHolderName), stdin);

// Corrected strcspn to remove newline character after fgets
account.accountHolderName[strcspn(account.accountHolderName, "\n")] = 0;

scanf("%d", &account.accountType);

if (account.accountType == 1) {
scanf("%f %f", &account.accountDetails.savings.balance,
&account.accountDetails.savings.interestRate);
printf("Savings Account Balance: %.2f\n", account.accountDetails.savings.balance);
printf("Annual Interest Rate: %.2f%%\n", account.accountDetails.savings.interestRate);
} else if (account.accountType == 2) {
scanf("%f %f", &account.accountDetails.checking.balance,
&account.accountDetails.checking.overdraftLimit);
printf("Checking Account Balance: %.2f\n",
account.accountDetails.checking.balance);
printf("Overdraft Limit: %.2f\n", account.accountDetails.checking.overdraftLimit);
} else {
printf("Invalid account type.\n");
}

return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Savings Account Balance: 1500.75
Annual Interest Rate: 2.50%
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Checking Account Balance: 2000.50
Overdraft Limit: 500.00
```

Compilation Status: Passed

Execution Time:

0.001s

59. Conditional Compilation for DebuggingProblem Statement: Create a C program that uses conditional compilation to include or exclude debug information based on a macro definition. You need to handle the scenario where debug information is enabled only when a macro DEBUG is defined. The program should include debug messages if DEBUG is defined and exclude them otherwise.

Description:

Define a macro DEBUG to toggle debug messages.Use conditional compilation to print debug information about the execution of your program.The program should prompt the user for an integer and then print whether the integer is even or odd.Input Format:

An integer from the user.Output Format:

Print whether the integer is even or odd.Include debug information if DEBUG is defined.

Sample Input:10

Sample Output:Debug: Input value is 10The number is even

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#define DEBUG
int main() {
```

```
int number;
scanf("%d", &number);
#ifdef DEBUG
printf("Debug: Input value is %d\n", number);
#endif
if (number % 2 == 0) {
printf("The number is even\n");
} else {
printf("The number is odd\n");
}
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Debug: Input value is 10
The number is even
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Debug: Input value is 4
The number is even
```

Compilation Status: Passed

Execution Time:

0.001s

60. Recursive Array Manipulation with PointersProblem Statement:Write a C program that uses recursion to manipulate a 2D array of integers. Your task is to implement a recursive function that performs the following operations:

Traverse the 2D array and compute the sum of all elements.Compute the product of all elements.Count the number of elements greater than a given threshold.Description:You need to implement a recursive function that takes a pointer to the 2D array and its dimensions (number of rows and columns).The function should return a structure containing the sum, product, and count of elements greater than a threshold.Input Format:The first line contains two integers: rows and cols (dimensions of the 2D array).The next rows lines each contain cols integers representing the 2D array elements.The last line contains an integer threshold.Output Format:Print the sum, product, and count of elements greater than the threshold.

Sample Input:2 31 2 34 5 64

Sample Output:Sum: 21Product: 720Count greater than 4: 2

Completion Status: Completed

Concepts Included:

gu 28 1st semester c programming

Language Used: C

Source Code:

```
#include <stdio.h>
#define MAX_ROWS 100
#define MAX_COLS 100
typedef struct {
int sum;
long long product;
int countGreaterThanThreshold;
} ArrayResult;
ArrayResult processArray(int arr[MAX_ROWS][MAX_COLS],
int row, int col, int rows, int cols, int threshold) {
ArrayResult result = {0, 1, 0};
if (row >= rows) {
return result;
}

if (col >= cols) {
return processArray(arr, row + 1, 0, rows, cols, threshold);
}
int currentElement = arr[row][col];
result.sum += currentElement;
result.product *= currentElement;
if (currentElement > threshold) {
result.countGreaterThanThreshold++;
}
ArrayResult nextResult = processArray(arr, row, col + 1,
rows, cols, threshold);
result.sum += nextResult.sum;
result.product *= nextResult.product;
result.countGreaterThanThreshold +=
nextResult.countGreaterThanThreshold;
return result;
}
int main() {
int rows, cols, threshold;
int arr[MAX_ROWS][MAX_COLS];
scanf("%d %d", &rows, &cols);
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
scanf("%d", &arr[i][j]);
}
}
scanf("%d", &threshold);
ArrayResult result = processArray(arr, 0, 0, rows, cols,
threshold);
printf("Sum: %d\n", result.sum);
printf("Product: %lld\n", result.product);
printf("Count greater than %d: %d\n", threshold,
result.countGreaterThanThreshold);
return 0;
}
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Sum: 21
Product: 720
Count greater than 4: 2
```

Compilation Status: Passed

Execution Time:

0.001s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Sum: 31
Product: 2160
Count greater than 1: 5
```

Compilation Status: Passed

Execution Time:

0.001s