Program

```c
#include <stdio.h>

int isPossible(int pages[], int n, int s, int maxPages){
        int students = 1, sum = 0;
        for(int i= 0; i < n ; i++){
                if(pages[i] > maxPages)
                        return 0;
                if(sum + pages[i] > maxPages){
                        students++;
                        sum = pages[i];
                        if(students > s)
                                return 0;
                }
                else {
                        sum += pages[i];
                }
        }
        return 1;
}

int main(){
        int n , s;
        printf("Enter number of books: ");
        scanf("%d", &n);
        int pages[n];
        printf("Enter number of pages in each book (sorted): \n");
        ↪
        for(int i = 0; i < n;i++)
                scanf("%d", &pages[i]);
        printf("Enter number of students: ");
        scanf("%d", &s);
        int low = 0, high = 0;
        for(int i = 0; i < n; i++){
                high += pages[i];
        }
        int ans = -1;
        while(low <= high){
                int mid = (low+high)/2;
                if(isPossible(pages, n, s, mid)){
                        ans = mid;
                        high = mid - 1;
                }else {
                        low = mid + 1;
                }
        }
        int sum = 0; int students = 1;
        printf("Students %d: [", students);
```

```
        for(int i= 0; i < n;i++){
                if(sum + pages[i] > ans){
                        printf("] -->  %d pages\n", sum);
                        sum = pages[i];
                        students++;
                        printf("Student %d: [%d", students, pages[i]);
                }
                else{
                        if(sum == 0)
                                printf("%d", pages[i]);
                        else
                                printf(", %d", pages[i]);
                        sum += pages[i];
                }
        }
        printf("] --> %d pages \n", sum);
        return 0;

}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P1_Book_Arrangement$ ./a.out
Enter number of books: 6
Enter number of pages in each book (sorted):
10
20
30
40
50
60
Enter number of students: 4
Students 1: [10, 20, 30] -->  60 pages
Student 2: [40] -->  40 pages
Student 3: [50] -->  50 pages
Student 4: [60] --> 60 pages
```

Program

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

#define MAX_DEGREE 7

typedef struct {
    int coeffs[MAX_DEGREE];
} Polynomial;

void initPolynomial(Polynomial* p) {
    int i;
    for (i = 0; i < MAX_DEGREE; i++)
        (*p).coeffs[i] = 0;
}

int parseNumber(const char* str, int* idx) {
    int val = 0;
    while (isdigit(str[*idx])) {
        val = val * 10 + (str[*idx] - '0');
        (*idx)++;
    }
    return val;
}

void parseTerm(char* term, Polynomial* p) {
    int coeff = 0, degree = 0, i = 0;

    // skip leading spaces
    while (term[i] == ' ') i++;

    if (isdigit(term[i]))
        coeff = parseNumber(term, &i);
    else if (term[i] == 'x' || term[i] == 'X') {
        coeff = 1;
    }

    if (term[i] == 'x' || term[i] == 'X') {
        i++;
        if (isdigit(term[i]))
            degree = parseNumber(term, &i);
        else
            degree = 1;
    } else {
        degree = 0;
    }
```

```c
    if (degree >= 0 && degree < MAX_DEGREE)
        (*p).coeffs[degree] += coeff;
}

void readPolynomial(Polynomial* p) {
    char input[256];
    fgets(input, sizeof(input), stdin);
    char* token = strtok(input, "+\n");
    while (token != NULL) {
        parseTerm(token, p);
        token = strtok(NULL, "+\n");
    }
}

void addPolynomials(Polynomial* raw, Polynomial* mod3, Polynomial a,
↪  Polynomial b) {
    int i;
    for (i = 0; i < MAX_DEGREE; i++) {
        int sum = a.coeffs[i] + b.coeffs[i];
        (*raw).coeffs[i] = sum;
        if (sum != 0 && sum % 3 == 0)
            (*mod3).coeffs[i] = 0;
        else
            (*mod3).coeffs[i] = sum;
    }
}

void printPolynomial(Polynomial p) {
    int i, first = 1;
    for (i = MAX_DEGREE - 1; i >= 0; i--) {
        int c = p.coeffs[i];
        if (c == 0) continue;
        if (!first) printf(" + ");
        if (i == 0)
            printf("%d", c);
        else {
            if (c != 1)
                printf("%d", c);
            printf("x");
            if (i > 1)
                printf("%d", i);
        }
        first = 0;
    }
    if (first) printf("0");
    printf("\n");
}

int main() {
    Polynomial m, k, raw_sum, mod_result;
```

```
    initPolynomial(&m);
    initPolynomial(&k);
    initPolynomial(&raw_sum);
    initPolynomial(&mod_result);

    printf("k(x) = ");
    readPolynomial(&k);

    printf("m(x) = ");
    readPolynomial(&m);

    addPolynomials(&raw_sum, &mod_result, m, k);

    printf("Raw sum: ");
    printPolynomial(raw_sum);

    printf("Modulo-3 result: ");
    printPolynomial(mod_result);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P2_Cryptography$ ./a.out
k(x) = 2x2 + 1x1 + 3
m(x) = 1x2 + 2x + 4
Raw sum: 3x2 + 3x + 7
Modulo-3 result: 7
```

Program

```c
#include <stdio.h>

void inputSparseMatrix(int rows, int cols, int mat[100][100], int
↪  nonZero) {
    int r, c, val;
    for (int i = 0; i < nonZero; i++) {
        printf("Enter row, col, value for object %d: ", i + 1);
        scanf("%d %d %d", &r, &c, &val);
        if (r < rows && c < cols)
            mat[r][c] = val;
        else
            printf("Invalid position. Skipped.\n");
    }
}

void addMatrices(int rows, int cols, int A[100][100], int B[100][100],
↪  int result[100][100]) {
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            result[i][j] = A[i][j] + B[i][j];
}

void transposeMatrix(int rows, int cols, int mat[100][100], int
↪  trans[100][100]) {
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            trans[j][i] = mat[i][j];
}

void printMatrix(int rows, int cols, int mat[100][100]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
            printf("%3d ", mat[i][j]);
        printf("\n");
    }
}

int main() {
    int rows, cols;
    int buildings[100][100] = {0}, vehicles[100][100] = {0},
    ↪  characters[100][100] = {0};
    int scene[100][100] = {0}, temp[100][100] = {0},
    ↪  finalFrame[100][100] = {0};

    printf("Enter scene size (rows cols): ");
    scanf("%d %d", &rows, &cols);
```

```
    int nz_buildings, nz_vehicles, nz_characters;
    printf("Enter number of objects in Layer 1 (Buildings): ");
    scanf("%d", &nz_buildings);
    inputSparseMatrix(rows, cols, buildings, nz_buildings);

    printf("Enter number of objects in Layer 2 (Vehicles): ");
    scanf("%d", &nz_vehicles);
    inputSparseMatrix(rows, cols, vehicles, nz_vehicles);

    printf("Enter number of objects in Layer 3 (Characters): ");
    scanf("%d", &nz_characters);
    inputSparseMatrix(rows, cols, characters, nz_characters);

    addMatrices(rows, cols, buildings, vehicles, temp);
    addMatrices(rows, cols, temp, characters, scene);

    printf("\nScene Matrix (Before Transpose):\n");
    printMatrix(rows, cols, scene);

    transposeMatrix(rows, cols, scene, finalFrame);

    printf("\nFinal Transposed Frame sent to GPU:\n");
    printMatrix(cols, rows, finalFrame);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P3_Game_Dev$ ./a.out
Enter scene size (rows cols): 5 5
Enter number of objects in Layer 1 (Buildings): 2
Enter row, col, value for object 1: 0 1 2
Enter row, col, value for object 2: 2 1 1
Enter number of objects in Layer 2 (Vehicles): 2
Enter row, col, value for object 1: 1 1 1
Enter row, col, value for object 2: 0 0 2
Enter number of objects in Layer 3 (Characters): 3
Enter row, col, value for object 1: 1 0 2
Enter row, col, value for object 2: 2 1 2
Enter row, col, value for object 3: 2 2 3

Scene Matrix (Before Transpose):
  2   2   0   0   0
  2   1   0   0   0
  0   3   3   0   0
  0   0   0   0   0
  0   0   0   0   0

Final Transposed Frame sent to GPU:
  2   2   0   0   0
  2   1   3   0   0
  0   0   3   0   0
  0   0   0   0   0
  0   0   0   0   0
```

Program

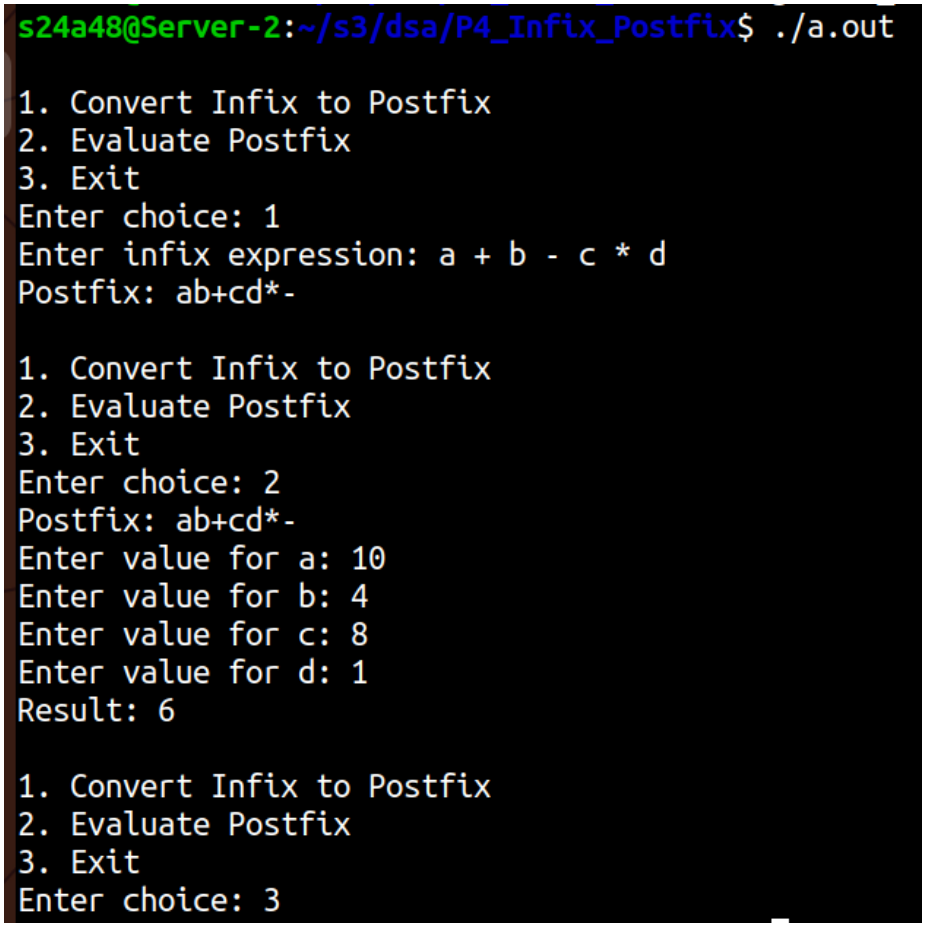
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

char opStack[100];
int opTop = -1;

void pushOp(char ch) {
    opStack[++opTop] = ch;
}

char popOp() {
    return opStack[opTop--];
}

char peekOp() {
    return (opTop >= 0) ? opStack[opTop] : '\0';
}

int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}

int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^';
}

int isVariable(char ch) {
    return isalpha(ch);
}

void infixToPostfix(char* infix, char* postfix) {
    int i = 0, j = 0;
    char ch;
    while ((ch = infix[i++]) != '\0') {
        if (isVariable(ch)) {
            postfix[j++] = ch;
        } else if (ch == '(') {
            pushOp(ch);
        } else if (ch == ')') {
            while (peekOp() != '(')
                postfix[j++] = popOp();
```

```
            popOp(); // remove '('
        } else if (isOperator(ch)) {
            while (opTop != -1 && precedence(peekOp()) >= precedence(ch))
                postfix[j++] = popOp();
            pushOp(ch);
        }
    }
    while (opTop != -1)
        postfix[j++] = popOp();
    postfix[j] = '\0';
}

int valStack[100];
int valTop = -1;

void pushVal(int val) {
    valStack[++valTop] = val;
}

int popVal() {
    return valStack[valTop--];
}

int evaluatePostfix(char* postfix) {
    int varValues[26] = {0};
    int usedVars[26] = {0};
    int i = 0;
    char ch;
    while ((ch = postfix[i++]) != '\0') {
        if (isVariable(ch))
            usedVars[ch - 'a'] = 1;
    }
    for (int j = 0; j < 26; j++) {
        if (usedVars[j]) {
            printf("Enter value for %c: ", 'a' + j);
            scanf("%d", &varValues[j]);
        }
    }
    i = 0;
    while ((ch = postfix[i++]) != '\0') {
        if (isVariable(ch)) {
            pushVal(varValues[ch - 'a']);
        } else if (isOperator(ch)) {
            int b = popVal();
            int a = popVal();
            int result = 0;
            switch (ch) {
                case '+': result = a + b; break;
                case '-': result = a - b; break;
                case '*': result = a * b; break;
```

```
                case '/': result = a / b; break;
                case '^': result = (int)pow(a, b); break;
            }
            pushVal(result);
        }
    }
    return popVal();
}

int main() {
    int choice;
    char infix[100] = "", postfix[100] = "";
    int expressionAvailable = 0;

    while (1) {
        printf("\n1. Convert Infix to Postfix\n");
        printf("2. Evaluate Postfix\n");
        printf("3. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        getchar();

        switch (choice) {
            case 1:
                printf("Enter infix expression: ");
                fgets(infix, sizeof(infix), stdin);
                infix[strcspn(infix, "\n")] = '\0';
                opTop = -1;
                infixToPostfix(infix, postfix);
                printf("Postfix: %s\n", postfix);
                expressionAvailable = 1;
                break;

            case 2:
                if (!expressionAvailable) {
                    printf("No postfix expression available. Convert
                    ↪  infix first.\n");
                } else {
                    printf("Postfix: %s\n", postfix);
                    valTop = -1;
                    printf("Result: %d\n", evaluatePostfix(postfix));
                }
                break;

            case 3:
                exit(0);

            default:
                printf("Invalid choice.\n");
        }
```

```
    }

    return 0;
}
```

Sample run of the program



Program

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


struct PrintJob {
    char teacher[30];
    char document[50];
    int pages;
};


struct PrintJob queue[10];
int job_count = 0;
int current_job = 0;


void addJob(char teacher[], char document[], int pages) {
    if (job_count < 10) {
        strcpy(queue[job_count].teacher, teacher);
        strcpy(queue[job_count].document, document);
        queue[job_count].pages = pages;
        job_count++;
        printf("Added: %s by %s\n", document, teacher);
    } else {
        printf("Print queue is full!\n");
    }
}


void showCurrentJob() {
    printf("\n--- Currently Printing ---\n");
    if (job_count == 0) {
        printf("No jobs in queue\n");
    } else if (current_job < job_count) {
        printf("Teacher: %s\n", queue[current_job].teacher);
        printf("Document: %s\n", queue[current_job].document);
        printf("Pages: %d\n", queue[current_job].pages);
        printf("Print time: %.1f minutes\n", queue[current_job].pages /
        ↪  30.0);
    } else {
        printf("All jobs completed\n");
    }
}


void showWaitingTime(char document[]) {
    int found = 0;
```

```
    float wait_time = 0;

    printf("\n--- Waiting Time ---\n");

    for (int i = 0; i < job_count; i++) {
        if (strcmp(queue[i].document, document) == 0) {
            found = 1;
            if (i < current_job) {
                printf("Document already printed\n");
            } else if (i == current_job) {
                printf("Document is currently printing\n");
            } else {
                // Calculate waiting time
                for (int j = current_job; j < i; j++) {
                    wait_time += queue[j].pages / 30.0;
                }
                printf("Document: %s\n", document);
                printf("Waiting time: %.1f minutes\n", wait_time);
            }
            break;
        }
    }

    if (!found) {
        printf("Document not found in queue\n");
    }
}


void showMaxWaitTeacher() {
    printf("\n--- Teacher with Maximum Wait ---\n");

    if (job_count == 0) {
        printf("No jobs in queue\n");
        return;
    }

    float max_wait = 0;
    int max_index = current_job;

    for (int i = current_job; i < job_count; i++) {
        float wait_time = 0;
        for (int j = current_job; j < i; j++) {
            wait_time += queue[j].pages / 30.0;
        }
        if (wait_time > max_wait) {
            max_wait = wait_time;
            max_index = i;
        }
    }
```

```
    if (max_index < job_count) {
        printf("Teacher: %s\n", queue[max_index].teacher);
        printf("Document: %s\n", queue[max_index].document);
        printf("Wait time: %.1f minutes\n", max_wait);
    }
}


void nextJob() {
    if (current_job < job_count) {
        printf("\nFinished printing: %s\n", queue[current_job].document);
        current_job++;
        if (current_job < job_count) {
            printf("Now printing: %s\n", queue[current_job].document);
        } else {
            printf("All jobs completed!\n");
        }
    } else {
        printf("No more jobs to print\n");
    }
}


void showAllJobs() {
    printf("\n--- Print Queue ---\n");
    if (job_count == 0) {
        printf("No jobs in queue\n");
        return;
    }

    for (int i = 0; i < job_count; i++) {
        if (i == current_job) {
            printf("* %d. %s - %s (%d pages) [PRINTING]\n",
                   i+1, queue[i].teacher, queue[i].document,
                   ↪  queue[i].pages);
        } else if (i < current_job) {
            printf("  %d. %s - %s (%d pages) [DONE]\n",
                   i+1, queue[i].teacher, queue[i].document,
                   ↪  queue[i].pages);
        } else {
            printf("  %d. %s - %s (%d pages) [WAITING]\n",
                   i+1, queue[i].teacher, queue[i].document,
                   ↪  queue[i].pages);
        }
    }
}

int main() {
```

```c
int choice;
char teacher[30], document[50];
int pages;

printf("=== WiFi Printer Simulation ===\n");

do {
    printf("\n1. Add new document\n");
    printf("2. Check waiting time for document\n");
    printf("3. Show teacher with max wait\n");
    printf("4. Show currently printing\n");
    printf("5. Complete current job\n");
    printf("6. Show all jobs\n");
    printf("7. Exit\n");
    printf("Choice: ");

    scanf("%d", &choice);

    switch(choice) {
        case 4:
            showCurrentJob();
            break;

        case 2:
            printf("Enter document name: ");
            scanf("%s", document);
            showWaitingTime(document);
            break;

        case 3:
            showMaxWaitTeacher();
            break;

        case 1:
            printf("Teacher name: ");
            scanf("%s", teacher);
            printf("Document name: ");
            scanf("%s", document);
            printf("Number of pages: ");
            scanf("%d", &pages);
            addJob(teacher, document, pages);
            break;

        case 5:
            nextJob();
            break;

        case 6:
```

```
                showAllJobs();
                break;

            case 7:

                break;

            default:
                printf("Invalid choice!\n");
        }

    } while(choice != 7);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P5_Wifi_Printer$ ./a.out
=== WiFi Printer Simulation ===

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 1
Teacher name: Prof.Akhil
Document name: Deld.pptx
Number of pages: 17
Added: Deld.pptx by Prof.Akhil

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 1
Teacher name: Prof.Leena
Document name: OOP_Concepts.pdf
Number of pages: 32
Added: OOP_Concepts.pdf by Prof.Leena
```

```
1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 1
Teacher name: Prof.Ajeesh
Document name: TOC_Tutorial.pdf
Number of pages: 44
Added: TOC_Tutorial.pdf by Prof.Ajeesh

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 2
Enter document name: Deld.pptx

--- Waiting Time ---
Document is currently printing
```

```
1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 2
Enter document name: OOP_Concepts.pdf

--- Waiting Time ---
Document: OOP_Concepts.pdf
Waiting time: 0.6 minutes

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 3

--- Teacher with Maximum Wait ---
Teacher: Prof.Ajeesh
Document: TOC_Tutorial.pdf
Wait time: 1.6 minutes
```

```
1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 4

--- Currently Printing ---
Teacher: Prof.Akhil
Document: Deld.pptx
Pages: 17
Print time: 0.6 minutes

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 5

Finished printing: Deld.pptx
Now printing: OOP_Concepts.pdf
```

```
1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 4

--- Currently Printing ---
Teacher: Prof.Leena
Document: OOP_Concepts.pdf
Pages: 32
Print time: 1.1 minutes

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 6

--- Print Queue ---
  1. Prof.Akhil - Deld.pptx (17 pages) [DONE]
* 2. Prof.Leena - OOP_Concepts.pdf (32 pages) [PRINTING]
  3. Prof.Ajeesh - TOC_Tutorial.pdf (44 pages) [WAITING]
```

```
1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 5

Finished printing: OOP_Concepts.pdf
Now printing: TOC_Tutorial.pdf

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 6

--- Print Queue ---
  1. Prof.Akhil - Deld.pptx (17 pages) [DONE]
  2. Prof.Leena - OOP_Concepts.pdf (32 pages) [DONE]
* 3. Prof.Ajeesh - TOC_Tutorial.pdf (44 pages) [PRINTING]

1. Add new document
2. Check waiting time for document
3. Show teacher with max wait
4. Show currently printing
5. Complete current job
6. Show all jobs
7. Exit
Choice: 7
```

Program

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

struct Customer {
    int tokenNumber;
    int forms;
};

int main() {
    int front = -1, rear = -1;
    int maxSize = 10;
    struct Customer queue[10];
    int tokenCounter = 1;

    while (1) {
        int choice;
        printf("\n--- Train Ticket Reservation ---\n");
        printf("1. Add Customer to Queue\n");
        printf("2. Serve Current Customer\n");
        printf("3. Display Current Customer\n");
        printf("4. Display Number of Customers Waiting\n");
        printf("5. Display All Customers in Queue\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            if ((front == 0 && rear == maxSize - 1) || (rear + 1) %
            ↪  maxSize == front) {
                printf("Queue is full! Cannot add more customers.\n");
            } else {
                struct Customer c;
                c.tokenNumber = tokenCounter++;

                printf("Enter number of tickets (forms) for customer: ");
                scanf("%d", &c.forms);

                if (front == -1) {
                    front = rear = 0;
                } else {
                    rear = (rear + 1) % maxSize;
                }
                queue[rear] = c;
                printf("Customer with token %d added to the queue.\n",
                ↪  c.tokenNumber);
            }
```

```
} else if (choice == 2) {
    if (front == -1) {
        printf("No customers in queue.\n");
    } else {
        struct Customer c = queue[front];
        printf("Serving customer with token %d, tickets: %d\n",
        ↪  c.tokenNumber, 1);

        c.forms--;
        if (c.forms > 0) {

            rear = (rear + 1) % maxSize;
            queue[rear] = c;
            printf("Customer with token %d goes to the end of
            ↪  the queue with %d tickets left.\n",
            ↪  c.tokenNumber, c.forms);
        }


        if (c.forms == 0) {
            if (front == rear) {
                front = rear = -1;
            } else {
                front = (front + 1) % maxSize;
            }
        } else {
            if (front == rear) {
                front = rear = -1;
            } else {
                front = (front + 1) % maxSize;
            }
        }
    }

} else if (choice == 3) {
    if (front == -1) {
        printf("No customer is being served.\n");
    } else {
        printf("Current customer: Token %d, Tickets left:
        ↪  %d\n", queue[front].tokenNumber,
        ↪  queue[front].forms);
    }

} else if (choice == 4) {
    if (front == -1) {
        printf("Number of customers waiting: 0\n");
    } else if (rear >= front) {
        printf("Number of customers waiting: %d\n", rear -
        ↪  front + 1);
    } else {
```

```
                printf("Number of customers waiting: %d\n", (maxSize -
                ↪  front) + (rear + 1));
            }

        } else if (choice == 5) {
            if (front == -1) {
                printf("No customers in queue.\n");
            } else {
                printf("Customers in queue:\n");
                int i = front;
                while (1) {
                    printf("Token %d - Tickets: %d\n",
                    ↪  queue[i].tokenNumber, queue[i].forms);
                    if (i == rear) break;
                    i = (i + 1) % maxSize;
                }
            }

        } else if (choice == 6) {
            printf("Exiting...\n");
            break;
        } else {
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P6_Railway_Booking$ ./a.out

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 1
Enter number of tickets (forms) for customer: 1
Customer with token 1 added to the queue.

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 1
Enter number of tickets (forms) for customer: 3
Customer with token 2 added to the queue.

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 1
Enter number of tickets (forms) for customer: 2
Customer with token 3 added to the queue.
```

```
--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 5
Customers in queue:
Token 1 - Tickets: 1
Token 2 - Tickets: 3
Token 3 - Tickets: 2

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 2
Serving customer with token 1, tickets: 1

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 3
Current customer: Token 2, Tickets left: 3
```

```
--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 4
Number of customers waiting: 2

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 2
Serving customer with token 2, tickets: 1
Customer with token 2 goes to the end of the queue with 2 tickets left.

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 2
Serving customer with token 3, tickets: 1
Customer with token 3 goes to the end of the queue with 1 tickets left.
```

```
--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 5
Customers in queue:
Token 2 - Tickets: 2
Token 3 - Tickets: 1

--- Train Ticket Reservation ---
1. Add Customer to Queue
2. Serve Current Customer
3. Display Current Customer
4. Display Number of Customers Waiting
5. Display All Customers in Queue
6. Exit
Enter choice: 6
Exiting...
```

Program

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

typedef struct {
    char urls[100][100];
    int top;
} Stack;

void init(Stack *s) {
    s->top = -1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

int isFull(Stack *s) {
    return s->top == 99;
}

void push(Stack *s, char url[]) {
    if (!isFull(s)) {
        strcpy(s->urls[++s->top], url);
    }
}

void pop(Stack *s, char url[]) {
    if (!isEmpty(s)) {
        strcpy(url, s->urls[s->top--]);
    }
}

int main() {
    Stack backStack, forwardStack;
    init(&backStack);
    init(&forwardStack);

    char currentPage[100] = "https://ktu.edu.in/";
    char url[100];
    int choice;

    printf("Starting at: %s\n", currentPage);

    do {
        printf("\n1. Visit New Page");
        printf("\n2. Back");
        printf("\n3. Forward");
```

```
        printf("\n4. Exit");
        printf("\nEnter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter new URL: ");
                scanf("%s", url);
                push(&backStack, currentPage);
                strcpy(currentPage, url);
                init(&forwardStack);
                break;

            case 2:
                if (!isEmpty(&backStack)) {
                    push(&forwardStack, currentPage);
                    pop(&backStack, currentPage);
                } else {
                    printf("No page to go back to.\n");
                }
                break;

            case 3:
                if (!isEmpty(&forwardStack)) {
                    push(&backStack, currentPage);
                    pop(&forwardStack, currentPage);
                } else {
                    printf("No page to go forward to.\n");
                }
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice.\n");
        }

        printf("Current Page: %s\n", currentPage);

    } while(choice != 4);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P7_Browser_Navigation$ ./a.out
Starting at: https://ktu.edu.in/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 2
No page to go back to.
Current Page: https://ktu.edu.in/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 1
Enter new URL: https://youtube.com/
Current Page: https://youtube.com/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 1
Enter new URL: https://cet.etlab.in/
Current Page: https://cet.etlab.in/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 2
Current Page: https://youtube.com/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 2
Current Page: https://ktu.edu.in/
```

```
1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 3
Current Page: https://youtube.com/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 3
Current Page: https://cet.etlab.in/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 3
No page to go forward to.
Current Page: https://cet.etlab.in/

1. Visit New Page
2. Back
3. Forward
4. Exit
Enter choice: 4
Exiting...
Current Page: https://cet.etlab.in/
```

Program

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

struct Node {
    char word[50];
    struct Node *next;
};

struct Node* createNode(char w[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->word, w);
    newNode->next = NULL;
    return newNode;
}

void append(struct Node** head, char w[]) {
    struct Node* newNode = createNode(w);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

int findReplace(struct Node* head, char find[], char replace[]) {
    int count = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        if (strcmp(temp->word, find) == 0) {
            strcpy(temp->word, replace);
            count++;
        }
        temp = temp->next;
    }
    return count;
}

void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%s ", temp->word);
        temp = temp->next;
    }
```

```
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    char text[1024];
    char *token;
    char find[50], replace[50];

    printf("Enter text: ");
    scanf(" %[^\n]", text);

    token = strtok(text, " ");
    while (token != NULL) {
        append(&head, token);
        token = strtok(NULL, " ");
    }

    printf("--- Entered Text ---\n");
    display(head);

    printf("Enter word to find: ");
    scanf("%s", find);
    printf("Enter replacement word: ");
    scanf("%s", replace);

    int occ = findReplace(head, find, replace);

    printf("%d replacements made\n", occ);
    printf("---- Replaced Text ----\n");
    display(head);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P8_Find_and_Replace$ ./a.out
Enter text:This program finds and replaces word text from this text to some other inputted word
--- Entered Text ---
This program finds and replaces word text from this text to some other inputted word
Enter find word(to replace):text
Enter replacing(new) word:INPUT
2 replacements made
---- Replaced Text ----
This program finds and replaces word INPUT from this INPUT to some other inputted word
```

Program

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

struct Node {
    int coeff;
    int exp;
    struct Node* next;
};

struct Node* createNode(int c, int e) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coeff = c;
    newNode->exp = e;
    newNode->next = NULL;
    return newNode;
}

void insertTerm(struct Node** head, int c, int e) {
    if (c == 0) return;
    struct Node* newNode = createNode(c, e);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    struct Node* prev = NULL;

    while (temp != NULL && temp->exp > e) {
        prev = temp;
        temp = temp->next;
    }
    if (temp != NULL && temp->exp == e) {
        temp->coeff += c;
        free(newNode);
        return;
    }
    if (prev == NULL) {
        newNode->next = *head;
        *head = newNode;
    } else {
        newNode->next = temp;
        prev->next = newNode;
    }
}

struct Node* readPoly() {
    int n, c, e;
```

```
    struct Node* head = NULL;
    printf("Enter number of terms: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter term %d (coeff exp): ", i + 1);
        scanf("%d %d", &c, &e);
        insertTerm(&head, c, e);
    }
    return head;
}

void printPoly(struct Node* head) {
    if (head == NULL) {
        printf("0\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->coeff > 0 && temp != head) printf(" + ");
        if (temp->exp == 0) printf("%d", temp->coeff);
        else if (temp->exp == 1) printf("%dx", temp->coeff);
        else printf("%dx^%d", temp->coeff, temp->exp);
        temp = temp->next;
    }
    printf("\n");
}

struct Node* multiply(struct Node* p1, struct Node* p2) {
    struct Node* result = NULL;
    for (struct Node* t1 = p1; t1 != NULL; t1 = t1->next) {
        for (struct Node* t2 = p2; t2 != NULL; t2 = t2->next) {
            insertTerm(&result, t1->coeff * t2->coeff, t1->exp +
            ↪  t2->exp);
        }
    }
    return result;
}

int main() {
    printf("-- Reading P1 --\n");
    struct Node* p1 = readPoly();
    printf("P1 = ");
    printPoly(p1);

    printf("-- Reading P2 --\n");
    struct Node* p2 = readPoly();
    printf("P2 = ");
    printPoly(p2);

    struct Node* result = multiply(p1, p2);
```

```
    printf("Polynomial Multiplication Output: ");
    printPoly(result);

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P9_Poly_Product$ ./a.out
-- Reading P1 ---
Enter number of terms: 3
Enter term 1 (coeff exp): 2 2
Enter term 2 (coeff exp): 3 1
Enter term 3 (coeff exp): 1 0
2x^2 + 3x + 1
-- Reading P2 ---
Enter number of terms: 2
Enter term 1 (coeff exp): 4 1
Enter term 2 (coeff exp): 2 0
4x + 2
Polynomial Multiplication Output: 8x^3 + 16x^2 + 10x + 2
```

Program

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

struct Node {
    char name[50];
    int freq;
    struct Node* next;
};

struct Node* head = NULL;

struct Node* createNode(char* name) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->name, name);
    newNode->freq = 0;
    newNode->next = NULL;
    return newNode;
}

void addApp(char* name) {
    struct Node* newNode = createNode(name);
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            if (strcmp(temp->name, name) == 0) {
                printf("App already exists.\n");
                free(newNode);
                return;
            }
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("App '%s' added successfully.\n", name);
}

void openApp(char* name) {
    struct Node* temp = head;
    while (temp != NULL) {
        if (strcmp(temp->name, name) == 0) {
            temp->freq++;
            printf("Opened %s (frequency = %d)\n", name, temp->freq);
            return;
        }
```

```
        temp = temp->next;
    }
    printf("App not found: %s\n", name);
}

void showApps() {
    if (head == NULL) {
        printf("No apps available.\n");
        return;
    }
    struct Node* temp = head;
    printf("Apps and their usage:\n");
    while (temp != NULL) {
        printf("%s (freq = %d)\n", temp->name, temp->freq);
        temp = temp->next;
    }
}

void cleanApps() {
    if (head == NULL) {
        printf("No apps to clean.\n");
        return;
    }

    struct Node* temp = head;
    struct Node* minNode = head;
    struct Node* prev = NULL;
    struct Node* minPrev = NULL;

    while (temp != NULL) {
        if (temp->freq < minNode->freq) {
            minNode = temp;
            minPrev = prev;
        }
        prev = temp;
        temp = temp->next;
    }

    if (minPrev == NULL) {
        head = head->next;
    } else {
        minPrev->next = minNode->next;
    }
    printf("Cleaning (removing): %s (freq = %d)\n", minNode->name,
    ↪  minNode->freq);
    free(minNode);
}

int main() {
    int choice;
```

```
    char appName[50];

    while (1) {
        printf("\n--- App Cleaner Menu ---\n");
        printf("1. Add App\n");
        printf("2. Open App\n");
        printf("3. Show All Apps\n");
        printf("4. Clean Least Used App\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter app name: ");
            scanf("%s", appName);
            addApp(appName);
        }
        else if (choice == 2) {
            printf("Enter app name to open: ");
            scanf("%s", appName);
            openApp(appName);
        }
        else if (choice == 3) {
            showApps();
        }
        else if (choice == 4) {
            cleanApps();
        }
        else if (choice == 5) {
            printf("Exiting...\n");
            break;
        }
        else {
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

Sample run of the program

```
s24a48@Server-2:~/s3/dsa/P10_App_Cleaner$ ./a.out

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 1
Enter app name: Instagram
App 'Instagram' added successfully.

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 1
Enter app name: Whatsapp
App 'Whatsapp' added successfully.

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 1
Enter app name: Facebook
App 'Facebook' added successfully.

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 2
Enter app name to open: Whatsapp
Opened Whatsapp (frequency = 1)
```

```
--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 2
Enter app name to open: Instagram
Opened Instagram (frequency = 1)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 2
Enter app name to open: Whatsapp
Opened Whatsapp (frequency = 2)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 3
Apps and their usage:
Instagram (freq = 1)
Whatsapp (freq = 2)
Facebook (freq = 0)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 4
Cleaning (removing): Facebook (freq = 0)
```

```
--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 3
Apps and their usage:
Instagram (freq = 1)
Whatsapp (freq = 2)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 4
Cleaning (removing): Instagram (freq = 1)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 3
Apps and their usage:
Whatsapp (freq = 2)

--- App Cleaner Menu ---
1. Add App
2. Open App
3. Show All Apps
4. Clean Least Used App
5. Exit
Enter choice: 5
Exiting...
```