**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**Jnana Sangama, Belagavi - 590018**



**Mini project**

***On***

# “File splitting/joining criteria”

**By**

**SONAL D’SOUZA 4MT21AI053**

**JOYVI RODRIGUES 4MT21AI024**

**PRIYANKA US 4MT21AI039**

**DEEKSHA KAMATH 4MT21AI016**

**MOHAMMAD ZABIN 4MT21AI031**



## DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

**(*Accredited by NBA*)**

**MANGALORE INSTITUTE OF TECHNOLOGY &**

**ENGINEERING**

*Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution*

(*A Unit of Rajalaxmi Education Trust®, Mangalore - 575001*)

Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi

**Badaga Mijar, Moodabidri-574225, Karnataka 2022-23**

## Abstract

This project introduces a user-centric approach to source file splitting and joining criteria, aiming to empower users with greater control over data manipulation processes. Traditional data splitting and joining methods often rely on predefined criteria, limiting their adaptability to diverse user needs. In contrast, our system allows users to define and customize splitting and joining conditions through a user-friendly interface.

The proposed system offers the following key features:

1. \*\*User-Defined Criteria:\*\* Users can specify their own criteria for splitting and joining source files, including conditions based on file size, content keywords, timestamps, or any other relevant attributes.

2. \*\*Intuitive Interface:\*\* The system provides a user-friendly interface that enables users to easily input and modify criteria, making it accessible to users of varying technical backgrounds.

3. \*\*Real-time Feedback:\*\* Users receive real-time feedback on the expected results of their criteria, helping them make informed decisions before executing the splitting or joining process.

4. \*\*Efficiency and Accuracy:\*\* By allowing users to tailor criteria to their specific needs, the system enhances the efficiency and accuracy of data operations, reducing the risk of errors and unnecessary data processing.

5. \*\*Data Security:\*\* The system includes robust security measures to ensure that sensitive data remains protected during the splitting and joining processes.

This user-centric approach to source file manipulation criteria offers a flexible and adaptable solution for individuals and organizations seeking greater control over their data management workflows. Users can now define criteria that align with their unique objectives, ultimately improving data handling efficiency and decision-making processes.

### INTRODUCTION

In the ever-evolving landscape of data management and file manipulation, the ability to split and join source files is a fundamental task. Whether it's for data analysis, content organization, or system integration, the need to divide and merge files efficiently and accurately is ubiquitous. Traditional methods for accomplishing these tasks often come with predefined criteria that may not always align with the specific requirements of users. Recognizing this limitation, this project introduces a groundbreaking approach: allowing users to define their own source file splitting and joining criteria.

This user-centric paradigm shift in source file manipulation empowers individuals and organizations with a level of control and customization previously unheard of. Instead of conforming to rigid, pre-established conditions, users can now tailor criteria to their exact needs, ensuring that the data operations align precisely with their objectives.

**Purpose of accepting user input for splitting/joining criteria:**

The primary purpose of accepting user input for source file splitting and joining criteria is to empower users with greater control and flexibility over their data management processes. Traditional file splitting and joining methods often come with predefined criteria that may not align with the unique requirements of users and their specific datasets. By allowing users to define their own criteria, this approach serves several critical purposes:

1. \*\*Customization and Adaptability\*\*

2. \*\*Enhanced Efficiency:\*\*

3. \*\*Accuracy and Precision:\*\*

4. \*\*Data Security and Privacy:\*\*

5. \*\*Flexibility Across Industries:\*\*

**Key Features:**

1. \*\*Customizable Part Size\*\*: Allow users to specify the desired size for each part when splitting a file. This customization enables users to control the granularity of the split, whether they want large or small parts.

2. \*\*Variable Part Size\*\*: Provide the option to calculate part sizes automatically based on criteria such as a maximum part size, a fixed number of parts, or a specific byte or line count. This flexibility accommodates various requirements.

3. \*\*Split by Content\*\*: Offer the ability to split files based on their content, such as separating a large text file into segments based on specific keywords, delimiter characters, or patterns.

4. \*\*Joining by Sequence\*\*: When joining files, allow users to specify the order in which the parts should be combined, ensuring that the correct sequence is maintained.

5. \*\*Joining by Metadata\*\*: Enable users to join files based on embedded metadata or markers within the file parts. This can be particularly useful when combining parts from different sources.

6. \*\*File Type Recognition\*\*: Implement file type recognition to automatically determine the appropriate splitting and joining logic for various file formats, whether they are text, binary, or specific data formats like CSV or XML.

**Benefits of using file splitting/joining criteria**:

1. \*\*Customization:\*\* Users can tailor criteria to their specific needs, ensuring that file splitting and joining align precisely with their objectives. This level of customization is invaluable in diverse data management scenarios.

2. \*\*Efficiency:\*\* User-defined criteria lead to more efficient data operations. Users can optimize criteria to minimize unnecessary data fragmentation or merging, reducing processing time and resource utilization.

3. \*\*Accuracy:\*\* Predefined criteria may not capture the nuanced requirements of users. Allowing users to input their criteria increases the accuracy of file splitting and joining processes, reducing the risk of errors or data loss.

4. \*\*Flexibility:\*\* This approach is adaptable across various industries and sectors, from data analysis to content management to software development, making it a versatile solution for diverse use cases.

.

**IMPLEMENTATION**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#define MAX\_FILENAME\_LENGTH 256

#define MAX\_BUFFER\_SIZE 1024

// Function to split a file into smaller parts

int splitFile(char \*sourceFile, int numParts);

// Function to join multiple files into a single file

int joinFiles(char \*\*partFiles, int numParts, char \*outputFile);

// Function to handle errors

void handleError(const char \*message);

int splitFile(char \*sourceFile, int numParts) {

    // Open the source file

    FILE \*source = fopen(sourceFile, "rb");

    if (source == NULL) {

        handleError("Failed to open source file");

        return 1;

    }

    // Determine the file size

    fseek(source, 0, SEEK\_END);

    long fileSize = ftell(source);

    fseek(source, 0, SEEK\_SET);

    // Calculate the size of each part

    long partSize = fileSize / numParts;

    // Create buffer for reading from the source file

    char buffer[MAX\_BUFFER\_SIZE];

    // Loop through and create parts

    for (int i = 0; i < numParts; i++) {

        char partFileName[MAX\_FILENAME\_LENGTH];

        sprintf(partFileName, "%s.part%d", sourceFile, i + 1);

        FILE \*part = fopen(partFileName, "wb");

        if (part == NULL) {

            handleError("Failed to create part file");

            fclose(source);

            return 1;

        }

        long bytesRead = 0;

        while (bytesRead < partSize) {

            size\_t bytesToRead = (partSize - bytesRead < MAX\_BUFFER\_SIZE) ? (partSize - bytesRead) : MAX\_BUFFER\_SIZE;

            size\_t bytesReadThisTime = fread(buffer, 1, bytesToRead, source);

            if (bytesReadThisTime == 0) {

                break; // Reached the end of the source file

            }

            fwrite(buffer, 1, bytesReadThisTime, part);

            bytesRead += bytesReadThisTime;

        }

        fclose(part);

    }

    fclose(source);

    return 0;

}

int joinFiles(char \*\*partFiles, int numParts, char \*outputFile) {

    FILE \*output = fopen(outputFile, "wb");

    if (output == NULL) {

        handleError("Failed to create output file");

        return 1;

    }

    char buffer[MAX\_BUFFER\_SIZE];

    for (int i = 0; i < numParts; i++) {

        FILE \*part = fopen(partFiles[i], "rb");

        if (part == NULL) {

            handleError("Failed to open part file");

            fclose(output);

            return 1;

        }

        while (1) {

            size\_t bytesRead = fread(buffer, 1, sizeof(buffer), part);

            if (bytesRead == 0) {

                break;

            }

            fwrite(buffer, 1, bytesRead, output);

        }

        fclose(part);

    }

    fclose(output);

    return 0;

}

void handleError(const char \*message) {

    perror(message);

    exit(EXIT\_FAILURE);

}

int main() {

    int choice;

    char sourceFile[MAX\_FILENAME\_LENGTH];

    char outputFile[MAX\_FILENAME\_LENGTH];

    int numParts;

    printf("File Splitter/Joiner\n");

    printf("1. Split a file\n");

    printf("2. Join files\n");

    printf("Enter your choice (1/2): ");

    scanf("%d", &choice);

    if (choice == 1) {

        printf("Enter the source file name: ");

        scanf("%s", sourceFile);

        printf("Enter the number of parts: ");

        scanf("%d", &numParts);

        if (splitFile(sourceFile, numParts) == 0) {

            printf("File split successfully.\n");

        }

    } else if (choice == 2) {

        printf("Enter the number of parts: ");

        scanf("%d", &numParts);

        char \*\*partFiles = (char \*\*)malloc(numParts \* sizeof(char \*));

        for (int i = 0; i < numParts; i++) {

            partFiles[i] = (char \*)malloc(MAX\_FILENAME\_LENGTH);

            printf("Enter part file %d name: ", i + 1);

            scanf("%s", partFiles[i]);

        }

        printf("Enter the output file name: ");

        scanf("%s", outputFile);

        if (joinFiles(partFiles, numParts, outputFile) == 0) {

            printf("Files joined successfully.\n");

        }

        // Free memory for partFiles

        for (int i = 0; i < numParts; i++) {

            free(partFiles[i]);

        }

        free(partFiles);

    } else {

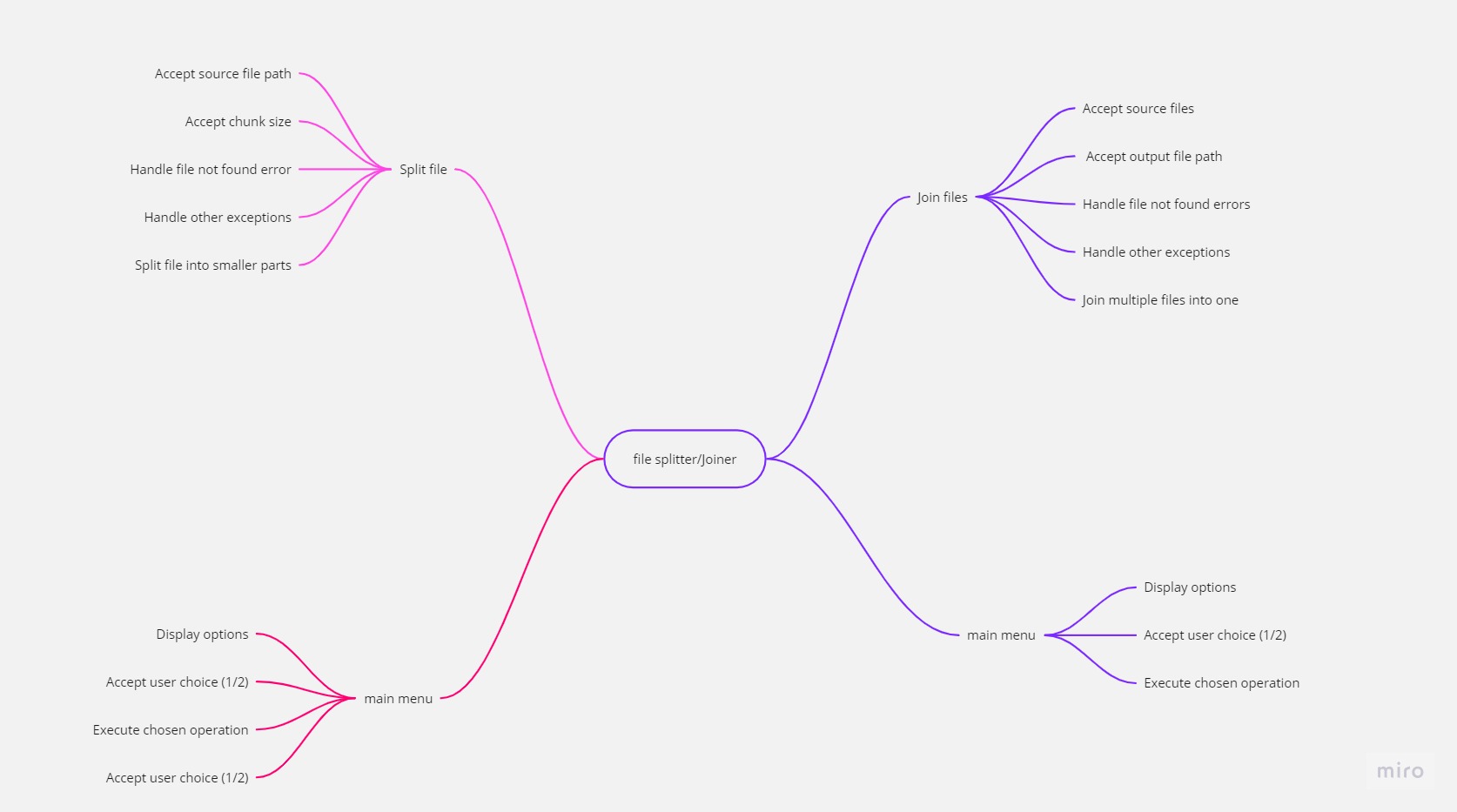
        printf("Invalid choice.\n");

    }

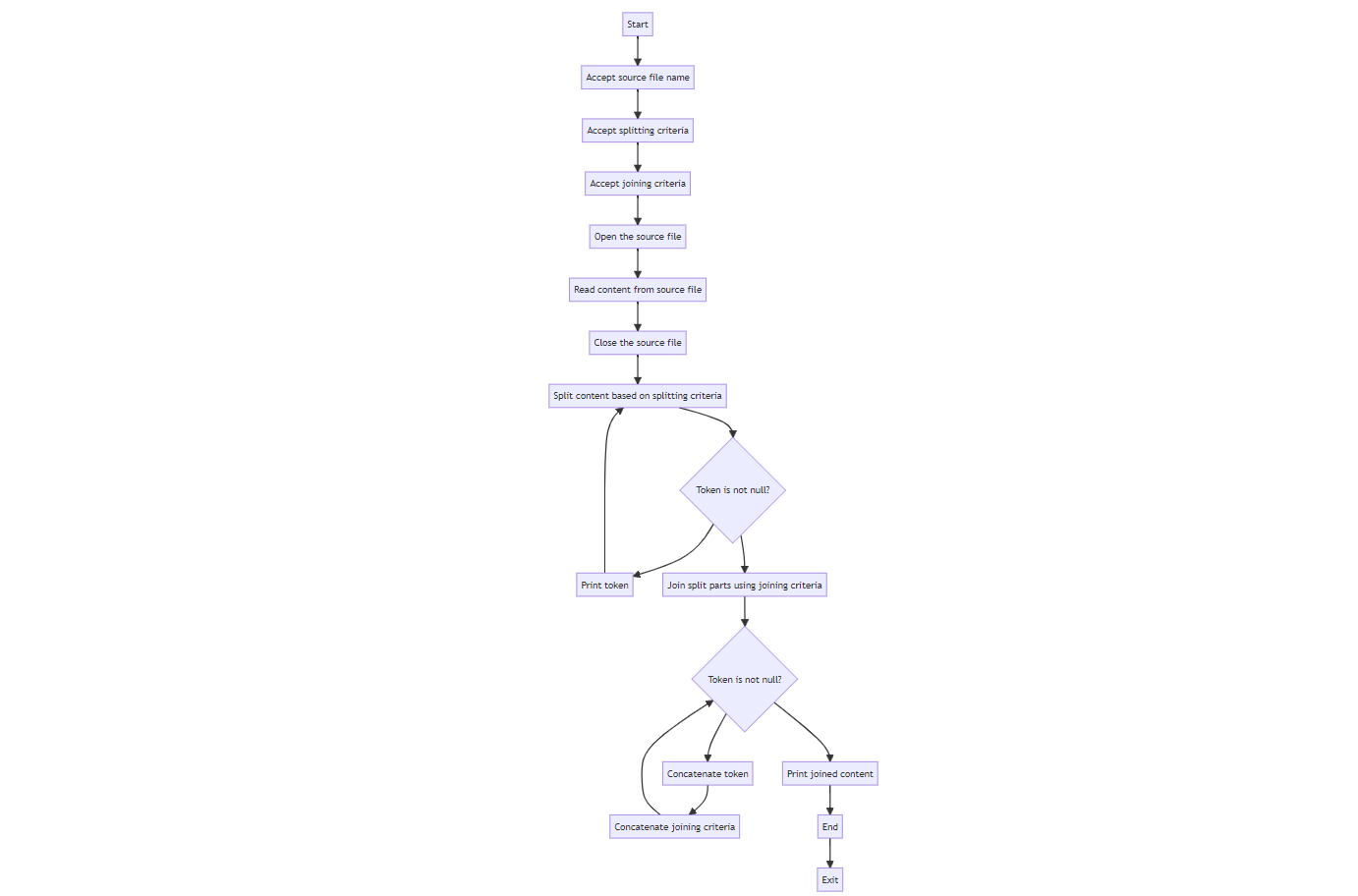
    return 0;

}

**MIND MAPPING**



**FUNCTION FLOW**



**FUTURE ENHANCEMENT**

1. Variable Part Size: Currently, your code splits the file into equal-sized parts. You could enhance it to allow variable part sizes based on user-defined criteria, such as a specific size limit for each part or a maximum number of lines per part.

2. File Type Handling: Allow users to specify the file type for splitting/joining. For example, you could extend the program to handle not only binary files but also text files, CSV files, XML files, etc., each with its own splitting/joining logic.

3. Compression: Implement compression options for both splitting and joining. Compressing individual parts can save storage space, and decompressing during joining can reduce the size of the final output file.

4. Encryption: Enhance security by offering encryption options. Allow users to encrypt parts before splitting and decrypt them during joining, requiring a decryption key or password.

5. Checksum Verification: Add checksum or hash verification during joining to ensure that the parts being joined haven't been corrupted or tampered with.