CCP-Report

CodeMasters

Group Members:

Ali Hasan Farooqui (CT-25098)
Hasan Shahir (CT-25095)
Muneeb Rehman (CT-25092)

Discipline: BCIT

Teachers:
Sir Muhammad Abdullah
and
Sir Furqan Hussain

CCP Project

Custom C library with Urdu Keywords

Abstract

This report presents the design and implementation of a custom C library named 'Urdu.h'. The purpose of this project is to make programming in C more accessible and relatable to native Urdu speakers by providing Roman Urdu equivalents for standard C functions, macros, and syntax. This library allows programmers to use familiar Urdu-based function names while maintaining C language standards.

1. Introduction

The `urdu.h` library is a creative framework that localizes the C programming language by introducing Urdu-based macros and functions. It allows users to write and understand code in familiar linguistic terms, making programming more accessible to Urdu-speaking learners. This project bridges the gap between language and technology, simplifying programming education through cultural and linguistic adaptation.

2. Technical Terms and Concepts

Macro: A preprocessor directive that replaces code before compilation, improving readability and reducing repetition.
Function: A reusable code block that performs a specific task and supports parameterization.
Header File (`.h`): A file containing function declarations and macros to promote modular programming.
Standard I/O: Core library for handling input and output operations (`stdio.h`).
String Handling: Operations involving text manipulation, such as concatenation, copying, and comparison.
Control Flow: Logical structures such as loops and conditionals that determine program execution paths.
Compiler Directive: Instructions processed by the compiler before actual code execution begins.
Character Encoding: Representation of text characters in a computer system (e.g., ASCII).
Doxygen: A documentation generation tool that converts comments into structured technical documents.

3. System Design and Implementation

3.1 System Overview

The urdu.h system was designed to enhance code readability and accessibility for beginners and native Urdu speakers. It achieves this by combining C programming fundamentals with Urdu inspired syntax through macros and modular functions. The design emphasizes simplicity, modularity, and integration with existing C compilers.S ystem Architecture

3.2 System Architecture

The overall structure of the urdu.h library follows a modular design, ensuring scalability and reusability. The core architecture consists of several independent modules that handle different aspects of program execution.

- **Input/Output Module** : Simplifies user interaction through readable Urdu equivalents for input (`parhle_int`, `parhle_float`) and output (`likhde`, `likhde_line`).
- **String Handling Module** : Implements essential string operations such as copying, concatenation, comparison, and length determination.
- **Character Handling Module** : Provides case conversion and alphanumeric validation utilities.
- **File Handling Module** : Enables file operations (open, write, read, append) using macros like `file_kholo` and `file_likho`.
- **Control Flow Module** : Replaces traditional control structures (`if`, `while`, `switch`) with Urdu style equivalents for improved readability.
- **Utility Functions** : Includes helper routines to manage input buffers, streamline user interaction, and ensure clean data handling.

3.3 Functional Flow

The functional flow begins with importing the `urdu.h` header file into a C program. Once included, the programmer can use all Urdu inspired macros and functions as direct replacements for their C counterparts.

- **Input Phase:** The program prompts the user for input through Urdu macros (`parhle`, `parhle_int`, etc.).
- **Processing Phase:** Logical conditions and loops use macros like `agr`, `warna`, and `jab_tak`.
- **Output Phase:** Results are displayed via Urdu print functions such as `likhde` and `likhde_line`.

- **File Operations:** Data is stored, retrieved, or appended through macros (`file_kholo`, `file_likho`, `file_band`).
- **Termination:** The program ends after executing control flow and output routines, demonstrating complete Urdu-based syntax flow.

3.4 File Structure

urdu.h	**→ Core library header file containing all macros and functions**
demo.c	**→ Demonstration program showcasing all features**
Doxyfile	**→ Configuration file for generating Doxygen documentation**

3.5 Integration in Dev C++

To make UrduC globally accessible in the IDE:

- Locate the Dev-C++ installation directory (e.g., `C:\Program Files (x86)\Dev-Cpp\MinGW64\include`).
- Copy the `urdu.h` file into the `include` folder.
- Restart Dev-C++.
- Include it in any program using:

```
#include <urdu.h>
```

3.6 Challenges

During development, several issues were identified and resolved:

- Macro Conflicts: Care was taken to avoid naming conflicts with existing C keywords or standard functions.
- Input Buffering: Extra steps were added to handle leftover characters from input streams.
- Compatibility: Ensured the library works with all C compilers supporting the ANSI C standard.
- Testing and Debugging: Each function was validated for edge cases like empty inputs and invalid file paths**.**

3.7 Outcome

The implementation successfully created a functional and educational C library that blends linguistic familiarity with programming structure.
Through modular functions, descriptive macros, and file integration, urdu.h promotes both learning and innovation in localized programming environments.

4. Header File

```
#ifndef URDU_H
#define URDU_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <stddef.h> // For size_t


int parhle_int(const char *prompt) {
  int x;
  printf("%s", prompt);
  scanf("%d", &x);
  int c;
  while ((c = getchar()) != '\n' && c != EOF); // Clear buffer
  return x;
}

void parhle(const char *prompt, char *buf, int size) {
  printf("%s", prompt);
  if (fgets(buf, size, stdin) != NULL) {
    size_t len = strlen(buf);
    if (len > 0 && buf[len - 1] == '\n')
      buf[len - 1] = '\0';
  }
}

float parhle_float(const char *prompt) {
  float x;
  printf("%s", prompt);
  scanf("%f", &x);
  int c;
  while ((c = getchar()) != '\n' && c != EOF); // Clear buffer
  return x;
}

double parhle_double(const char *prompt) {
  double x;
  printf("%s", prompt);
  scanf("%lf", &x);
  int c;
  while ((c = getchar()) != '\n' && c != EOF); // Clear buffer
  return x;
}
```

```
void likhde(const char *text) {
   printf("%s", text);
}

void likhde_int(int n) {
   printf("%d", n);
}

void likhde_float(float n) {
   printf("%.2f", n);
}

void likhde_double(double n) {
   printf("%.2lf", n);
}

void likhde_line(const char *text) {
   printf("%s\n", text);
}

void likhde_int_line(int n) {
   printf("%d\n", n);
}

void likhde_float_line(float n) {
   printf("%.2f\n", n);
}

void likhde_harf(char ch) { printf("%c", ch); }
void likhde_harf_line(char ch) { printf("%c\n", ch); }


int lambai(const char* str) {
   int length = 0;
   while (*str != '\0') {
      length++;
      str++;
   }
   return length;
}

char* nakal(char* dest, const char* src) {
   char* ptr = dest;
   while (*src != '\0') {
      *ptr = *src;
      ptr++;
      src++;
   }
   *ptr = '\0';
   return dest;
}
```

```
char* jodo(char* dest, const char* src) {
    char* ptr = dest;
    while (*ptr != '\0') {
    }
    while (*src != '\0') {
        *ptr = *src;
        ptr++;
        src++;
    }
    *ptr = '\0';
    return dest;
}

int muqabla(const char* s1, const char* s2) {
    while (*s1 && (*s1 == *s2)) {
        s1++;
        s2++;
    }
    return (unsigned char)*s1 - (unsigned char)*s2;
}

char* ulta_karde(char* str) {
    int len = lambai(str);
    char* start = str;
    char* end = str + len - 1;
    char temp;

    while (end > start) {
        temp = *start;
        *start = *end;
        *end = temp;

        start++;
        end--;
    }
    return str;
}

char* harf_dhundo(const char* str, int ch) {
    while (*str != '\0') {
        if (*str == (char)ch) {
            return (char*)str;
        }
        str++;
    }
    if (*str == (char)ch) {
        return (char*)str;
    }
    return NULL;
}
```

```
char* hissa_nakal(char* dest, const char* src, size_t n) {
    char* ptr = dest;
    size_t count = 0;

    while (count < n && *src != '\0') {
        *ptr = *src;
        ptr++;
        src++;
        count++;
    }

    while (count < n) {
        *ptr = '\0';
        ptr++;
        count++;
    }

    return dest;
}

char* lafz_dhundo(const char* haystack, const char* needle) {
    if (*needle == '\0') {
        return (char*)haystack;
    }

    while (*haystack != '\0') {
        const char* h = haystack;
        const char* n = needle;

        while (*n != '\0' && *h == *n) {
            h++;
            n++;
        }

        if (*n == '\0') {
            return (char*)haystack;
        }

        haystack++;
    }

    return NULL;
}

char* bara_karde(char* str) {
    char* ptr = str;
    while (*ptr != '\0') {
        *ptr = toupper(*ptr);
        ptr++;
    }
    return str;
}
```

```
char* chhota_karde(char* str) {
    char* ptr = str;
    while (*ptr != '\0') {
        *ptr = tolower(*ptr);
        ptr++;
    }
    return str;
}


void abhi_ka_time(char *buffer, int size) {
    time_t abhi = time(NULL);
    struct tm *local = localtime(&abhi);
    strftime(buffer, size, "%c", local);
}

void aaj_ki_tareekh(char *buffer, int size) {
    time_t abhi = time(NULL);
    struct tm *local = localtime(&abhi);
    strftime(buffer, size, "%d-%m-%Y", local);
}

void abhi_ka_wakt(char *buffer, int size) {
    time_t abhi = time(NULL);
    struct tm *local = localtime(&abhi);
    strftime(buffer, size, "%H:%M:%S", local);
}

void ruk_jana(int seconds) {
    time_t start = time(NULL);
    while (difftime(time(NULL), start) < seconds);
}

void likhde_abhi_ka_time() {
    time_t abhi = time(NULL);
    printf("%s", ctime(&abhi)); // ctime already includes a newline
}

#define bara_karo(ch) toupper(ch)
#define chhota_karo(ch) tolower(ch)
#define kya_adad(ch) isdigit(ch)
#define kya_harf(ch) isalpha(ch)
#define kya_harfya_adad(ch) isalnum(ch)

#define agr if
#define warna else
#define warna_agr else if

#define badal switch
#define surat case
#define warna_sab default
```

```
#define jab_tak(condition) while (condition)
#define karo do
#define jab_(condition) while (condition)

#define agay_chalo continue
#define ruk_jao break


#define file_kholo(filename, mode) fopen(filename, mode)
#define file_band(stream) fclose(stream)
#define file_likho(stream, ...) fprintf(stream, __VA_ARGS__)
#define file_lo(stream, ...) fscanf(stream, __VA_ARGS__)
#define line_lo(str, size, stream) fgets(str, size, stream)
#define line_likho(str, stream) fputs(str, stream)

#endif // URDU_H
```

5. Demo Code

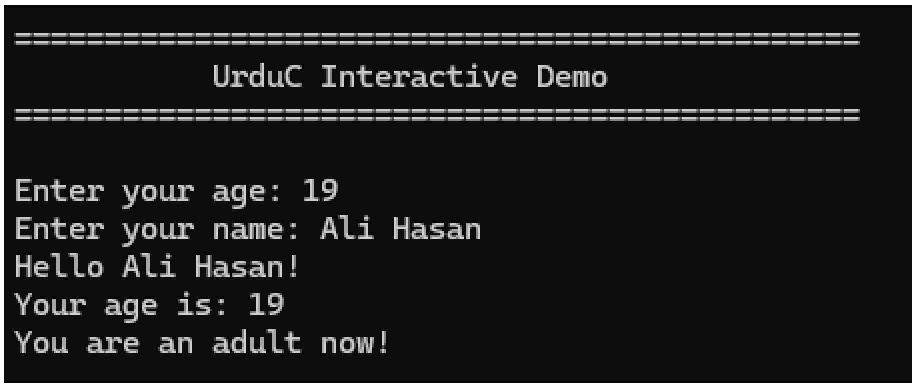
```
#include <urdu.h>


int main(void)
{
   likhde_line("============================================");
   likhde_line("          UrduC Interactive Demo");
   likhde_line("============================================\n");

   likhde_line("Assalam-o-Alaikum! \n");

   char naam[100];
   int umar = parhle_int("Apni umar likho: ");
   parhle("Apna naam likho: ", naam, sizeof(naam));

   likhde("\kese ho  ");
   likhde(naam);
   likhde_line("!");
   likhde("Aap ki umar hai: ");
   likhde_int_line(umar);

   agr (umar < 18)
      likhde_line("Aap abhi chhotay hain!");
   warna_agr (umar < 50)
      likhde_line("Aap ab barey hogai ho!");
   warna
      likhde_line("Aap to bohat barey hain!");
likhde_line("\n--- STRING FUNCTIONS DEMO ---");

char s1[100], s2[100], result[200];
parhle("Pehli string likho: ", s1, sizeof(s1));
parhle("Dusri string likho: ", s2, sizeof(s2));

likhde_line("\nNakal (Copy) demo:");
nakal(result, s1);
likhde("humne pehli string ko dusri mein daldiya: ");
likhde_line(result);

likhde_line("\nJodo (Concatenate) demo:");
jodo(result, " ");
jodo(result, s2);
likhde("Dono strings mil kar ban gayi: ");
likhde_line(result);

likhde_line("\nMuqabla (Compare) demo:");
int cmp = muqabla(s1, s2);
agr (cmp == 0)
   likhde_line("Dono strings barabar hain!");
warna_agr (cmp < 0)
   likhde_line("Pehli string chhoti hai.");
warna
   likhde_line("Pehli string badi hai.");

likhde_line("\nLambai (Length) demo:");
likhde("Pehli string ki lambai: ");
likhde_int_line(lambai(s1));
likhde("Dusri string ki lambai: ");
likhde_int_line(lambai(s2));

// --- CHARACTER FUNCTIONS DEMO ---
likhde_line("\n--- CHARACTER FUNCTIONS DEMO ---");
char ch;
likhde("Koi ek character likho: ");
scanf(" %c", &ch);

likhde("Bara karo (toupper): ");
likhde_harf_line(bara_karo(ch));
likhde("Chhota karo (tolower): ");
likhde_harf_line(chhota_karo(ch));
```

```
agr (kya_harf(ch))
    likhde_line("Yeh ek huroof hai.");
warna_agr (kya_adad(ch))
    likhde_line("Yeh ek number hai.");
warna
    likhde_line("Yeh na hi huroof hai na hi koi number.");

getchar();

likhde_line("\n--- FILE HANDLING DEMO ---");

char filename[100];
parhle("File ka naam likho (e.g., mera_data.txt): ", filename, sizeof(filename));

FILE *meraFile = file_kholo(filename, "w");
agr (meraFile == NULL) {
    likhde_line("File likhne mein masla aya!");
    return 1;
}

likhde_line("\nFile mein kuch likho:");
char fileContent[300];
parhle("", fileContent, sizeof(fileContent));

line_likho(fileContent, meraFile);
file_band(meraFile);

likhde_line("\nFile likh di gayi!");

likhde_line("\nKya aap file mein aur data jorna chahtay hain? (Y/N): ");
char choice;
scanf(" %c", &choice);
getchar();

agr (choice == 'Y' || choice == 'y') {
    FILE *appendFile = file_kholo(filename, "a");
    agr (appendFile == NULL) {
        likhde_line("File append karte waqt masla aya!");
        return 1;
    }

    likhde_line("File mein aur likho:");
    char extraContent[300];
    parhle("", extraContent, sizeof(extraContent));

    line_likho("\n", appendFile);
    line_likho(extraContent, appendFile);
    file_band(appendFile);
```

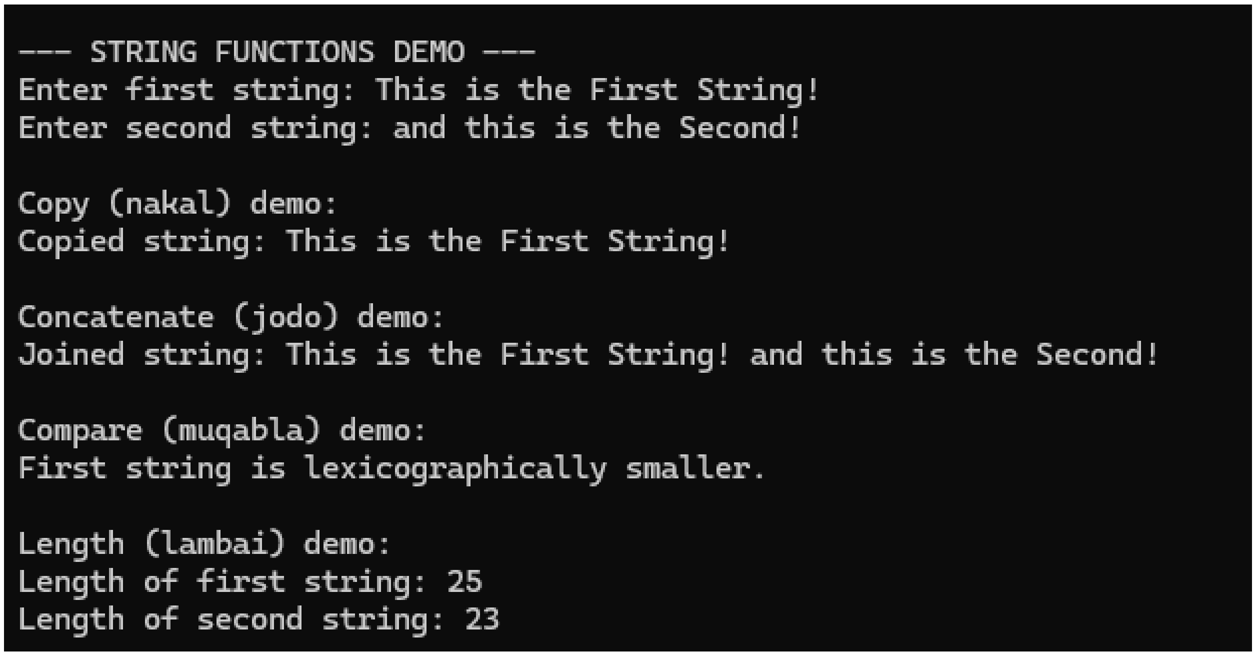
```
        likhde_line("Naya data successfully jor diya gaya!");
    }

    likhde_line("\nFile ka final content padhte hain:");
    FILE *readFile = file_kholo(filename, "r");
    agr (readFile == NULL) {
        likhde_line("File padhne mein masla aya!");
        return 1;
    }

    char line[300];
    jab_tak (line_lo(line, sizeof(line), readFile) != NULL) {
        likhde(line);
    }
    file_band(readFile);

    likhde_line("\n\n--- LOOP DEMO ---");
    int count = parhle_int("Kitni martaba likhna hai 'Pakistan Zindabad'? : ");
    int i = 0;

    jab_tak (i < count) {
        likhde_line("Pakistan Zindabad!");
        i++;
    }

    likhde_line("\nDemo mukammal hogaya!");
    likhde_line("Shukriya! Allah Hafiz.");

    return 0;
}
```

6. Result

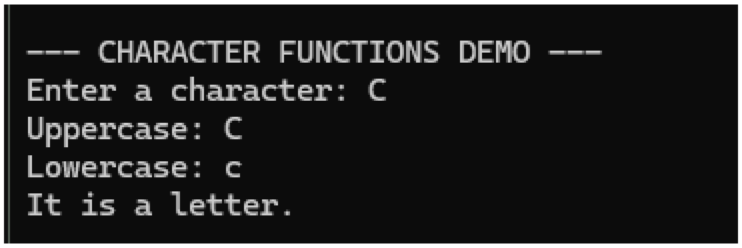
Screenshots:



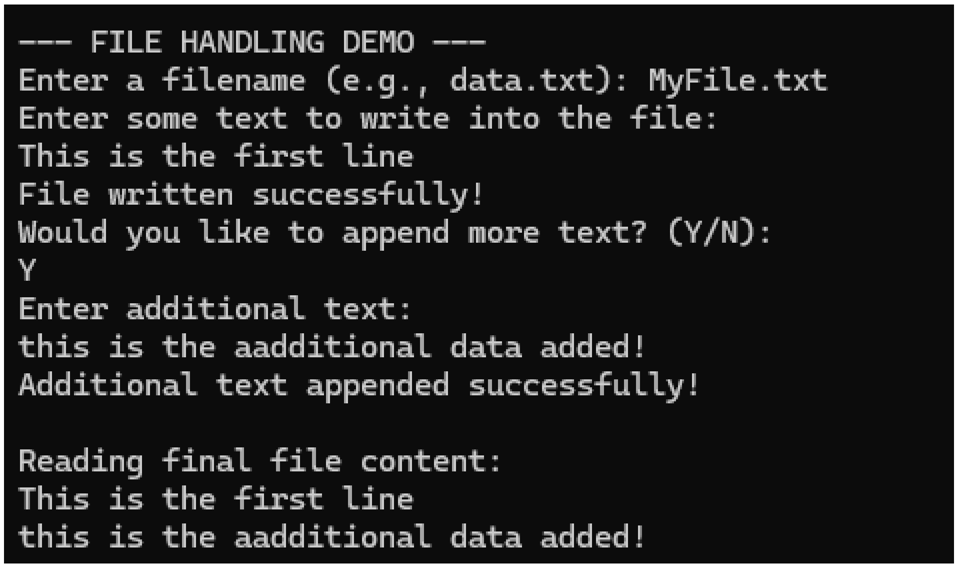
```
==================================================
           UrduC Interactive Demo
==================================================

Enter your age: 19
Enter your name: Ali Hasan
Hello Ali Hasan!
Your age is: 19
You are an adult now!
```



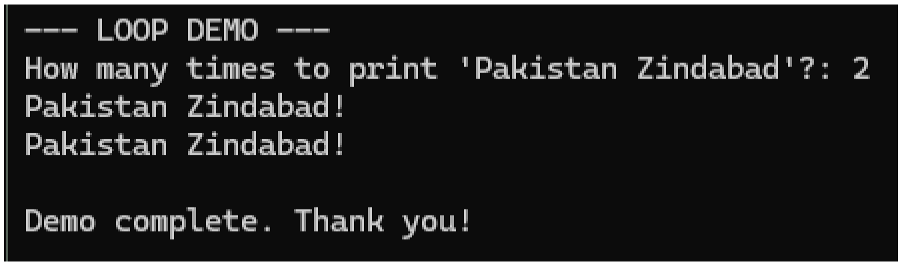
```
--- STRING FUNCTIONS DEMO ---
Enter first string: This is the First String!
Enter second string: and this is the Second!

Copy (nakal) demo:
Copied string: This is the First String!

Concatenate (jodo) demo:
Joined string: This is the First String! and this is the Second!

Compare (muqabla) demo:
First string is lexicographically smaller.

Length (lambai) demo:
Length of first string: 25
Length of second string: 23
```



```
--- CHARACTER FUNCTIONS DEMO ---
Enter a character: C
Uppercase: C
Lowercase: c
It is a letter.
```

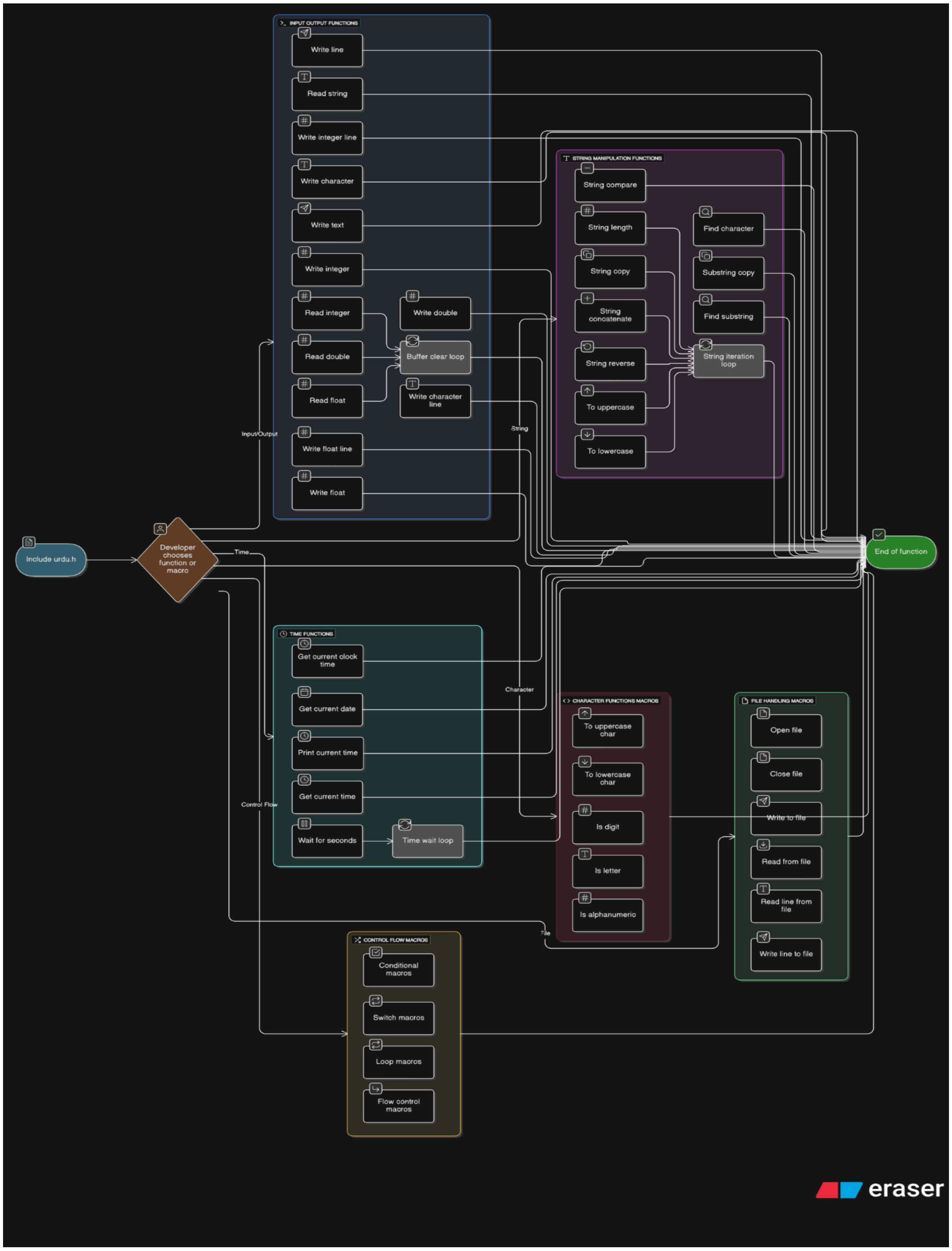


```
--- FILE HANDLING DEMO ---
Enter a filename (e.g., data.txt): MyFile.txt
Enter some text to write into the file:
This is the first line
File written successfully!
Would you like to append more text? (Y/N):
Y
Enter additional text:
this is the aadditional data added!
Additional text appended successfully!

Reading final file content:
This is the first line
this is the aadditional data added!
```

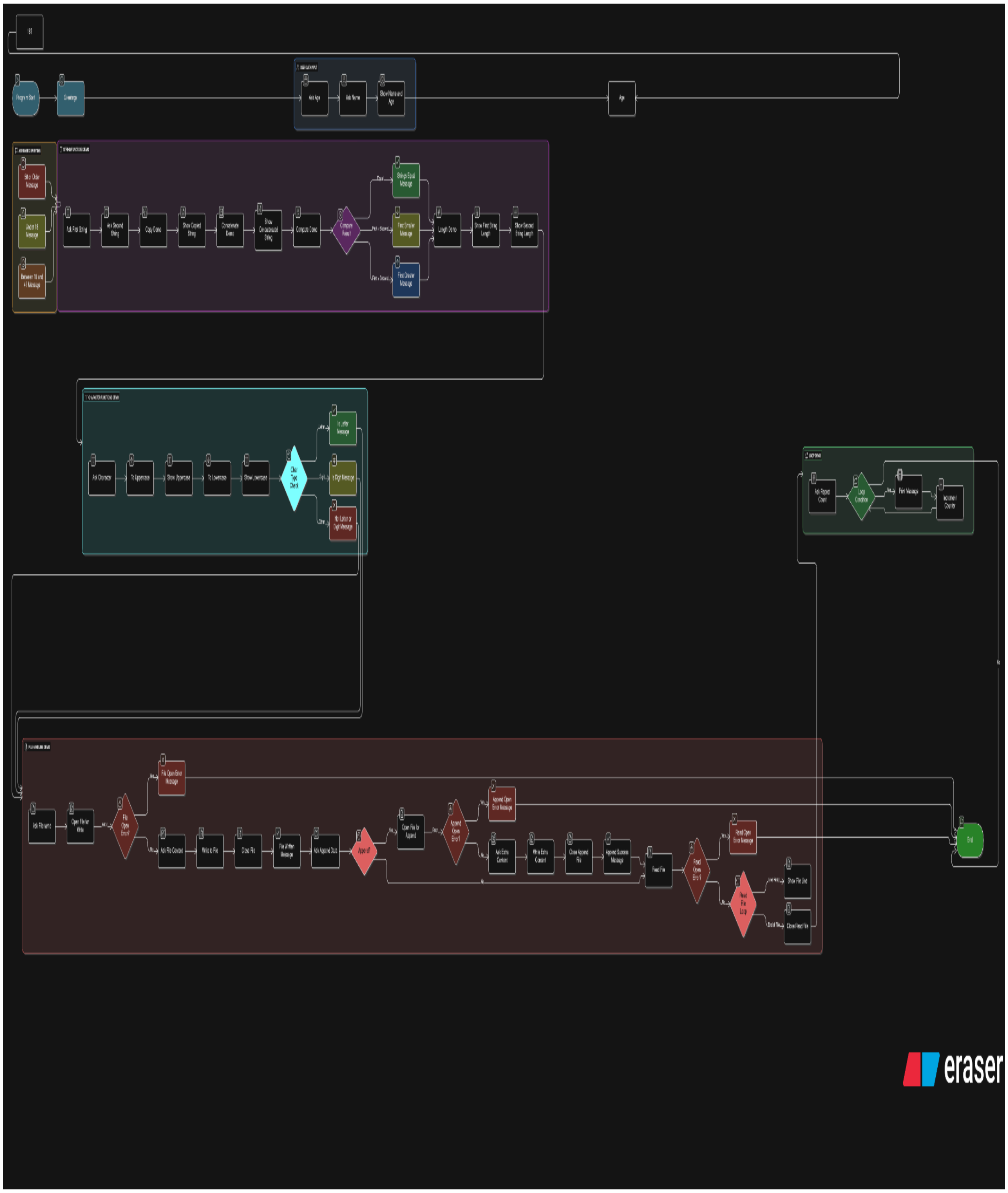


```
--- LOOP DEMO ---
How many times to print 'Pakistan Zindabad'?: 2
Pakistan Zindabad!
Pakistan Zindabad!

Demo complete. Thank you!
```

FLOWCHART (urdu.h):

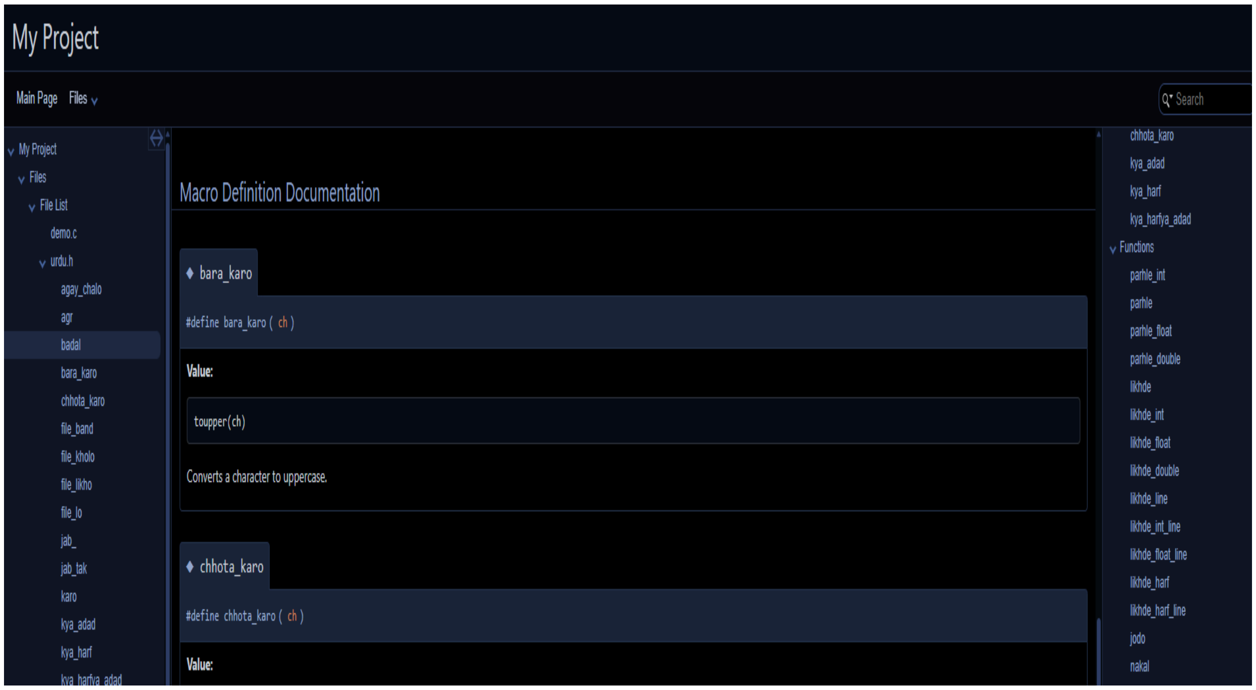


FLOWCHART (Demo File)



7. Doxygen

Used doxygen to create a professional documentation for the header file itself and the demo file



My Project

Main Page Files

Search

- My Project
  - Files
    - File List
      - demo.c
      - urdu.h
        - agay_chalo
        - agr
        - badal
        - bara_karo
        - chhota_karo
        - file_band
        - file_kholo
        - file_likho
        - file_lo
        - jab_
        - jab_tak
        - karo
        - kya_adad
        - kya_harf
        - kya_harfya_adad

Macro Definition Documentation

◆ bara_karo

```
#define bara_karo ( ch )
```

Value:

```
toupper(ch)
```

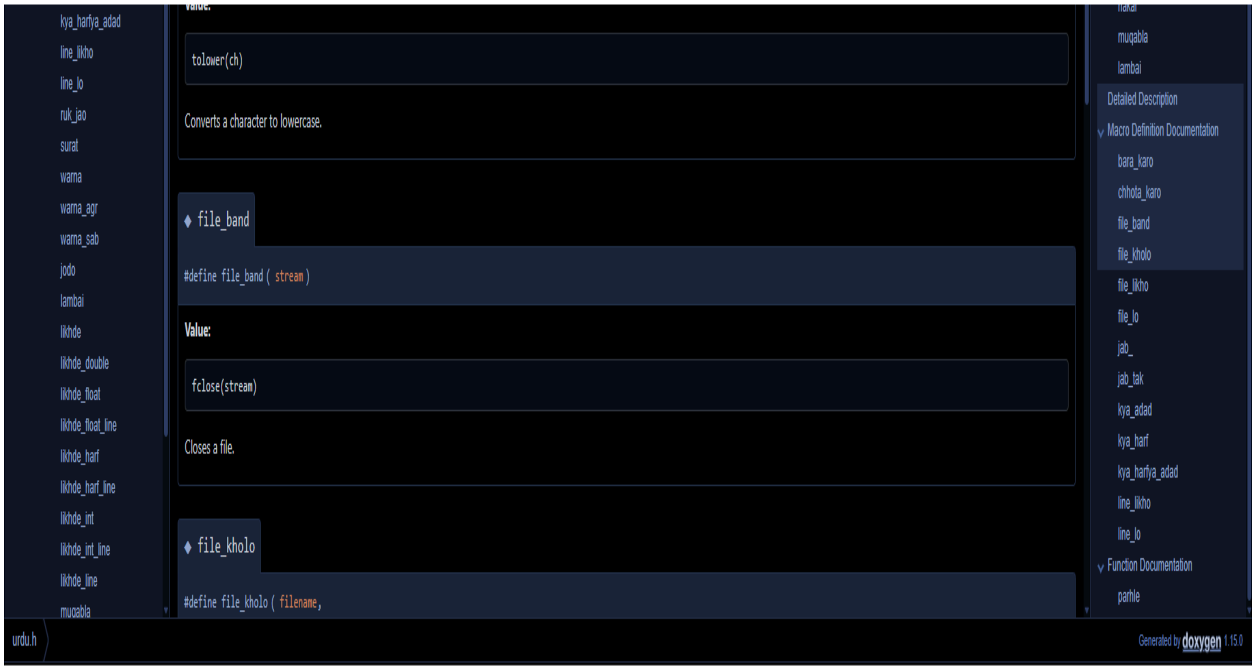
Converts a character to uppercase.

◆ chhota_karo

```
#define chhota_karo ( ch )
```

Value:

chhota_karo
kya_adad
kya_harf
kya_harfya_adad
Functions
parhle_int
parhle
parhle_float
parhle_double
likhde
likhde_int
likhde_float
likhde_double
likhde_line
likhde_int_line
likhde_float_line
likhde_harf
likhde_harf_line
jodo
nakal



kya_harfya_adad
line_likho
line_lo
ruk_jao
surat
warna
warna_agr
warna_sab
jodo
lambai
likhde
likhde_double
likhde_float
likhde_float_line
likhde_harf
likhde_harf_line
likhde_int
likhde_int_line
likhde_line
muqabla

Value:

```
tolower(ch)
```

Converts a character to lowercase.

◆ file_band

```
#define file_band ( stream )
```

Value:

```
fclose(stream)
```

Closes a file.

◆ file_kholo

```
#define file_kholo ( filename,
```

nakal
muqabla
lambai
Detailed Description
Macro Definition Documentation
bara_karo
chhota_karo
file_band
file_kholo
file_likho
file_lo
jab_
jab_tak
kya_adad
kya_harf
kya_harfya_adad
line_likho
line_lo
Function Documentation
parhle

urdu.h

Generated by doxygen 1.15.0

8. Assumptions

- Users have a basic understanding of the C programming language.
- The program is executed in a standard C compiler (e.g., Dev-C++, Code::Blocks, GCC).
- UTF-8 encoding is supported to correctly display Urdu characters.
- The UrduC header file is properly placed and included in the IDE's "include" directory.
- User inputs will be valid and of the expected data type.
- File operations (read/write/append) are performed in accessible directories.
- The system environment provides necessary permissions for file handling.
- Standard libraries (`stdio.h`, `stdlib.h`, `string.h`, `ctype.h`) are available and functional.
- Console supports Urdu text display without encoding errors.

9. Future Work

- Expand the urdu.h library with additional Urdu based macros for mathematical and logical operations.
- Introduce advanced string and file handling functions for greater flexibility.
- Develop a graphical interface to demonstrate UrduC usage interactively.
- Add error-handling and debugging features with Urdu based messages.
- Implement cross-platform support for Windows, Linux, and macOS environments.
- Create a package installer for easier integration into popular IDEs like Dev-C++ and Code::Blocks.
- Incorporate localization options for other regional languages alongside Urdu.
- Optimize existing functions for better performance and reduced memory usage.
- Publish urdu.h as an open-source project for community-driven contributions and enhancements.

10. Conclusion

The development of the urdu.h library represents an innovative step toward making programming more accessible to native Urdu speakers. By localizing C language syntax and providing user friendly macros and functions, urdu.h bridges the gap between complex coding concepts and linguistic familiarity. This initiative not only enhances understanding for beginners but also promotes inclusivity in computer science education. With further improvements, such as expanded functionality, optimized performance, and community contributions, urdu.h has the potential to evolve into a complete, culturally adaptive programming toolkit for learners and developers alike.