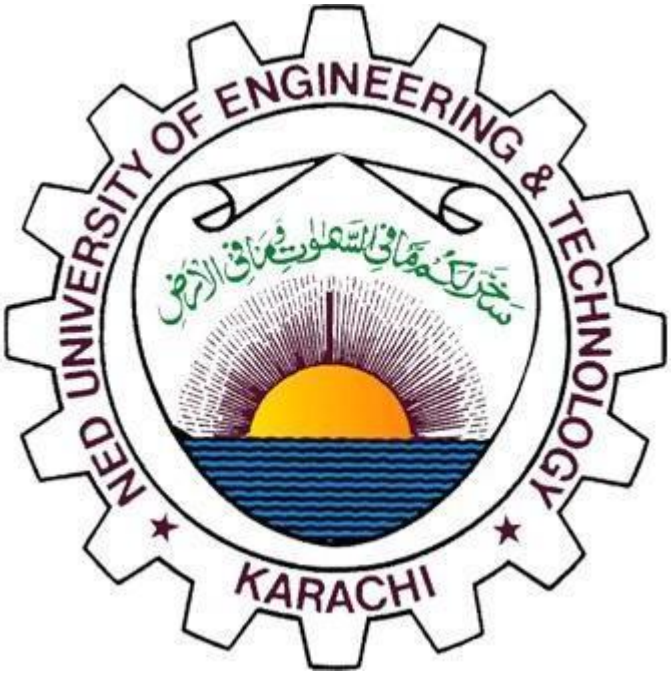
CCP-Proposal

CodeMasters



Group Members:

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

Discipline: BCIT
Teachers:

Sir Muhammad Abdullah

Sir Furqan Hussain

Sir Muhammad Abdullah

Sir Furqan Hussain

1. Project Title:

Custom C Library

2. Project Description:

The **UrduC Project** is an innovative attempt to simplify programming for native Urdu speakers by creating a C-language extension that replaces English syntax and functions with Urdu-based equivalents. This project introduces a custom header file, *urdu.h*, which redefines standard C constructs—such as loops, conditionals, and I/O operations—into intuitive Urdu keywords. The goal is to make programming more accessible to beginners and promote computational literacy among students who may find English-based programming syntax challenging. By maintaining full compatibility with traditional C compilers, the UrduC library bridges the gap between linguistic familiarity and technical understanding, fostering inclusivity in computer science education.

3. Project Methodology:

3.1 Requirement Analysis:

The first phase of the project involved identifying the key challenges faced by Urdu-speaking learners when approaching the C programming language. A survey of beginner programmers and educators revealed that English syntax often created a language barrier, leading to confusion and disengagement. Based on these insights, the project defined its core objective: to design a system that preserves C's functionality while replacing its syntax and standard library calls with Urdu equivalents. The requirement analysis also outlined compatibility with Dev-C++ and other common IDEs to ensure ease of adoption.

3.2 Design and Architecture:

The design phase focused on creating the *urdu.h* header file, which acts as the backbone of the UrduC language. The library was structured to mirror the original C syntax but with Urdu keywords and macros for core functionalities like loops (`jab_tak`, `karo`), conditionals (`agr`, `warna`), and file handling (`file_kholo`, `file_band`). The architecture was kept modular, separating user input/output, control flow, and file operations into distinct logical sections. This modular approach ensures scalability, making it easier to add future features such as error handling and extended data type support.

3.3 Implementation Phase:

In the implementation phase, all macros and functions were coded using standard C to ensure backward compatibility. The *urdu.h* file was developed incrementally, with rigorous testing for each feature—starting from simple input/output functions to advanced control structures and file manipulation. A demonstration file (*demo.c*) was created to showcase the use of all implemented macros and functions in practical examples, allowing users to see the complete workflow in action. The implementation also emphasized code readability, structured commenting, and Doxygen-based documentation to make the library developer-friendly.

3.4 Testing and Validation:

The testing phase included multiple stages to ensure robustness and accuracy. Each UrduC function was validated against its corresponding standard C counterpart to confirm identical behavior. Edge cases, such as invalid user inputs, empty files, and loop terminations, were tested thoroughly. The integration tests ensured that the UrduC library compiled correctly across different IDEs like Dev-C++ and Code::Blocks without modifications. The final validation involved generating Doxygen documentation and executing user-interactive test cases to verify usability and consistency. The results confirmed that the UrduC library successfully translated logical thinking into Urdu syntax without compromising program efficiency or execution speed.

3.5 Timeline:

Week	Task
Day 1	Problem definition, requirement analysis, and project planning.
Day 2	Design and creation of *urdu.h* structure, defining core macros and function prototypes.
Day 3	Implementation of basic input/output and control structure macros (e.g., `agr`, `warna`, `jab_tak`).
Day 4	Integration of advanced string and file handling functionalities.
Day 5	Development of the demonstration file (*urdu_demo.c*) showcasing all library features.
Day 6	Testing and debugging across multiple IDEs (Dev-C++, Code::Blocks, etc.).
Day 7	Generation of Doxygen documentation, preparation of project report, and final review.

3.6 Expected Outcomes:

The expected outcome of this project is the successful development of a user-friendly and efficient C language library named *urdu.h* that enables programming using Urdu-based macros and functions. This library will simplify coding for beginners by bridging the gap between natural language and programming syntax, promoting better understanding of fundamental programming concepts. The project will produce a fully functional demo file, comprehensive documentation using Doxygen, and a report outlining implementation details and usage. Additionally, the library will serve as a foundation for future enhancements such as integration with graphical interfaces, expanded file-handling features, and compatibility with multiple development environments.

3.7 Goals:

- To design and implement a custom C header library (*urdu.h*) that allows programming using Urdu-based macros and function equivalents.
- To simplify the learning process for new programmers by enabling code writing in a more familiar, natural-language-like syntax.
- To demonstrate the practical use of macros, file handling, and string operations through a comprehensive demo program.
- To document the entire system professionally using Doxygen for better understanding and maintainability.
- To ensure the library is modular, easy to integrate, and extensible for future upgrades or additional language-based functionalities.
- To encourage innovation in localized programming tools and improve accessibility to coding education.

4. Justification – Why it is a Complex Computing Problem:

The development of the *urdu.h* library represents a complex computing problem because it involves creating a custom abstraction layer on top of the C programming language while maintaining compatibility, efficiency, and readability. It requires an in-depth understanding of C preprocessor directives, macro behavior, memory management, and file I/O operations. Ensuring that Urdu-based keywords translate accurately into valid C syntax without breaking compiler rules adds further complexity. Additionally, designing the library to be modular, portable across compilers, and intuitive for end-users introduces challenges in both linguistic mapping and code structure. The integration of documentation automation through Doxygen and real-time demonstrations of macro functionalities further increases the project's technical depth, making it an advanced and multidisciplinary programming task.

6. Industrialization/Commercial Product Potential:

The *urdu.h* library has strong potential for industrialization and commercialization, particularly in regions where Urdu is widely spoken and used in educational or governmental institutions. It can be developed into a full-fledged localized programming toolkit that makes coding more accessible to beginners who are not yet comfortable with English-based syntax. This initiative can be extended to support other regional languages, turning it into a global "localized programming framework." Additionally, integrating the library into IDEs like Dev-C++, Code::Blocks, or Visual Studio Code as a plugin could open opportunities for commercialization in academic sectors, coding bootcamps, and national digital literacy programs. With further development, it can evolve into a paid educational software or open-source toolkit sponsored by tech institutions or governments to promote inclusive programming education.