Modern Periodic Table Explorer

Project Report

Submitted By: Nikhil Pant

Submitted To: Dr. Rahul Prasad

Course: B. Tech (Computer Science)

UPES Dehradun

Abstract

This report presents the design and implementation of a console-based application titled *Modern Periodic Table Explorer*, developed in the C programming language. The system allows users to explore detailed information about all 118 elements of the modern periodic table through a user-friendly, menu-driven interface. Using structured data storage via C structs and static arrays, the application supports element lookup by atomic number, displays core chemical properties, and presents formatted tabular output. The project demonstrates fundamental concepts of procedural programming, modular design, data structures, and input validation. Testing confirms that the system behaves correctly for valid, boundary, and invalid inputs. Potential extensions include support for search by name or symbol, graphical interfaces, and file-based data storage.

Problem Definition

The periodic table is a foundational tool in chemistry, but manual lookup of element data in textbooks or static charts can be time-consuming and inefficient, especially for students learning the subject. There is a need for a lightweight, offline, and easy-to-use digital tool that allows quick retrieval of important properties such as atomic number, symbol, atomic mass, group, block, category, and physical state.

The objective of this project is to design and implement a C program that:

- Stores information about all 118 chemical elements in a structured and organized form.
- Allows users to retrieve element data by atomic number through a simple console-based menu.
- Presents information in a clear, formatted, and readable way.
- Handles invalid inputs gracefully and prevents crashes.

The scope of the project is limited to:

- Single-user, console-based interaction.
- Read-only access to element data (no editing of element properties at runtime).
- Search by atomic number only.

System Design

Overall Architecture

The system follows a modular design with a clear separation between:

- **Data Layer**: Static array of Element structures storing the periodic table.
- **Logic Layer**: Functions for searching and retrieving element data.
- **Presentation Layer**: Functions for displaying menus, lists, and element details in formatted console output.

Program Flowchart

Below is a conceptual flowchart representing the main control flow of the application.

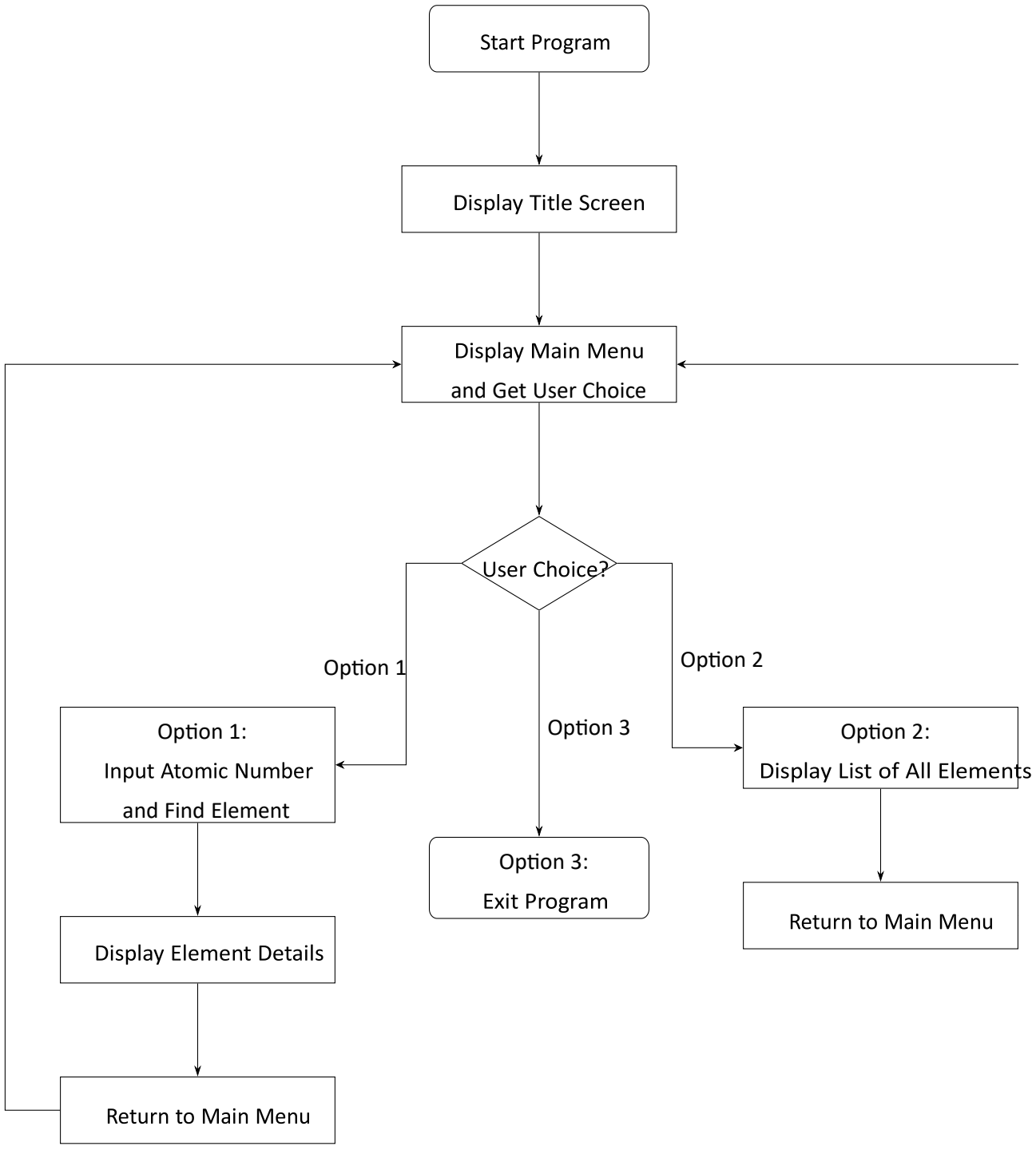

Figure 1: Main Program Flowchart for the Modern Periodic Table

Key Algorithms

Algorithm: Search Element by Atomic Number

1. Read the atomic number input from the user.
2. Validate that the atomic number is within the range 1 to 118.
3. If invalid, display an error message and return to the main menu.
4. If valid, linearly scan the array of elements:
    - For each element, compare its stored atomic number with the input.
    - If a match is found, return the corresponding element record.
5. If no element is found (should not occur for valid input), display an error message.

Algorithm: Display List of All Elements

1. Print a table header containing columns for atomic number, symbol, and name.
2. Iterate from index 0 to 117 in the element array.
3. For each element, print its atomic number, symbol, and name in a formatted row.
4. After the loop, print a closing table line and return to the main menu.

Implementation Details

Programming Language and Tools

- Programming Language: C
- Compiler: GCC (GNU Compiler Collection)
- Environment: Any terminal/command-prompt based system

Core Data Structure

The project uses a custom structure named Element to store the properties of each chemical element:

```
typedef struct {
    int atomic number;
    char name [25];
    char symbol [5];
    float atomic mass;
    char electronic configuration [60];
    char group [15];
    char block [5];
    char state_at_stp [15];
    char category [35];
} Element;
```

A static constant array of 118 Element records is used to represent the full periodic table.

Key Functions

- print title_screen(): Displays the ASCII art title and project description.
- findElementDetails(int atomicNum): Searches for an element by its atomic number and returns a pointer to the corresponding structure.
- displayElement(const Element *element): Prints detailed information about a given element in a formatted table.
- main (): Implements the main menu loop, handles user input, and calls other functions.

Sample Code Snippet: Element Display

```
void displayElement(const Element* element) {
    if (element == NULL) {
        printf("\n[ERROR] Element data is unavailable.\n");
        return;
    }
    printf("\n+-------------------------------------------------------------------------+\n");
    printf("| %-71s |\n","ELEMENT DETAILS");
    printf("+----------------------------+----------------------------------------+\n");
    printf("| %-28s | %-38d |\n","Atomic Number (Z)",element->atomic_number);
    printf("| %-28s | %-38s |\n","Name",element->name);
    printf("| %-28s | %-38s |\n","Symbol",element->symbol);
    printf("| %-28s | %-38.3f |\n","Atomic Mass (g/mol)",element->atomic_mass);
    printf("| %-28s | %-38s |\n","Group",element->group);
    printf("| %-28s | %-38s |\n","Block",element->block);
    printf("| %-28s | %-38s |\n","Category",element->category);
    printf("| %-28s | %-38s |\n","State (STP)",element->state_at_stp);
    printf("| %-28s | %-38s |\n","Electron Configuration",element->electronic_configuration);
    printf("+----------------------------+----------------------------------------+\n\n");
}
```

Testing & Results

Test Cases

The application was tested using a variety of inputs, including valid atomic numbers, boundary values, and invalid data. A summary of key test cases is shown below.

Test Case	Input	Expected Output	Status
Valid Element Search	Atomic Number = 1	Display details of Hydrogen	Pass
Valid Element Search	Atomic Number = 118	Display details of Oganesson	Pass
Lower Boundary	Atomic Number = 0	Error message: Invalid range	Pass
Upper Boundary	Atomic Number = 119	Error message: Invalid range	Pass
Invalid Input Type	Input = abc	Error message: Invalid input	Pass
Transition Metal	Atomic Number = 26	Display Iron as a Transition Metal	Pass
Noble Gas	Atomic Number = 10	Display Neon as a Noble Gas	Pass
Liquid Element	Atomic Number = 35	Display Bromine as Liquid at STP	Pass
List All Elements	Menu Option = 2	Display a table of all 118 elements	Pass
Exit Program	Menu Option = 3	Program terminates gracefully	Pass

Table 1: Summary of Test Cases and Results

Observed Behaviour

All test cases produced the expected output. The program handled incorrect inputs gracefully and consistently returned to the main menu instead of crashing. The element data displayed matched standard periodic table references.

Conclusion & Future Work

The *Modern Periodic Table Explorer* successfully demonstrates the use of C programming concepts such as structs, arrays, modular functions, and console-based user interfaces. The project meets its primary objective of providing quick and structured access to periodic table data for all 118 elements.

Key Achievements

- Implementation of a complete element database using C structures.
- Support for search by atomic number with robust input validation.
- Clean, readable tabular output that is easy for users to interpret.

Future Enhancements

Potential improvements include:

- Adding search functionality by element name or symbol.
- Introducing color-coded output or a graphical user interface (GUI).
- Loading element data from external configuration files instead of hardcoding.
- Providing additional properties such as electronegativity, atomic radius, and ionization energy.

References

- International Union of Pure and Applied Chemistry (IUPAC), Official Periodic Table Data.
- Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall.
- Online documentation for GCC and C standard libraries.
- Educational resources and periodic table references used for verifying element data.