IIT BHUBANESWAR

Project By:
SHREYANSH CHOUDHARY(23MM01023)
ANDALIB RASHID HASAN(23CE01004)
YASAS MOHANTY(23MM01026)

UNDER SUPERVISION OF
DR. DEVASHREE TRIPATHI

CONTENT

Your paragraph text

INTRODUCTION

We have developed an Utility Application ToolKit and decided to name it as “TaskEase” as it includes a bouquet of different day-to-day applications which are essential to one’s aid. This program is completely based on C Programming and runs solely on terminal. There are various functions in our program about which we will describe later

MOTIVATION

The motivation behind creating a utility application project that encompasses several daily use applications, such as Splitwise, Typing speed checker, calculator, unit converter, base system converter, and BMI calculator, can be multifaceted and driven by various factors:

1. Convenience and Efficiency: Instead of having to download and switch between multiple standalone applications for different tasks, users can access all the necessary tools within a single interface. This streamlines their workflow and saves time by eliminating the need to navigate between different apps.

2. User-Centric Approach: The project reflects a user-centric approach aimed at addressing common daily needs and tasks. By incorporating a diverse range of utilities, the application caters to a broad audience with varying requirements.

3. Resource Optimization: From a development standpoint, instead of allocating resources to develop and maintain separate applications, combining functionalities into a unified project enables better resource allocation and optimization. This can result in cost savings and streamline the development process.

In conclusion, I would like to state that our prime motivation to create this project was to dive deep into the development processes of the various utility apps that we use on a daily basis and also to create a convenient platform where we can utilise them conveniently and efficiently.

OVERVIEW

The All-in-One Utility Application is a comprehensive software project designed to streamline daily tasks and enhance productivity. Developed in C programming language, this versatile application integrates a diverse range of functionalities, including Splitwise integration, Typing speed checker, calculator, unit converter, Calendar, base system converter, and BMI calculator. By consolidating these utilities into a single platform, the application aims to provide users with a convenient and efficient solution to address their everyday needs.

DEVELOPMENT PROCESS

The development process of the All-in-One Utility Application involved iterative stages of planning, design, implementation, and testing. Initially, requirements were gathered, and a comprehensive plan was outlined. Following this, the application was designed, keeping user experience and functionality in mind. The implementation phase involved writing code in C programming language, integrating different utilities. Finally, rigorous testing was conducted to identify and address any bugs or issues, ensuring a stable and reliable application for users.

FUNCTION DESCRIPTION AND LEARNINGS

For Calendar :

- New header library like `<time.h>`:

  - This library provides functions and data types to work with date and time.
  - It includes functions like `time()` for obtaining the current time,
 ‘localtime()’ for converting a `time_t` value to a `struct tm`,and ‘mktime()’
 for converting a`struct tm``time_t` value .
  - It also includes data types like `time_t` and `struct tm` for representing
 time values and calendar time information.

- Time function is called while executing the main function :

  -It obtains the current time using the time() function and converts it to astruct tm using the localtime() function.
 It then extracts the current day, month, year, and day of the week from the struct tm.
 It prompts the user to enter a date (day, month, year) using printf() and scanf().
 It converts the input date to a struct tm using the provided day, month, and year values.

FUNCTION DESCRIPTION AND LEARNINGS

For Calendar :

- New header library like `<time.h>`:

  - This library provides functions and data types to work with date and time.

  - It includes functions like `time()` for obtaining the current time,

  'localtime()' for converting a `time_t` value to a `struct tm`,and 'mktime()'

  for converting a`struct tm``time_t` value .

  - It also includes data types like `time_t` and `struct tm` for representing

  time values and calendar time information.

- Time function is called while executing the main function :

  -It obtains the current time using the time() function and converts it to astruct tm using the localtime() function.

  It then extracts the current day, month, year, and day of the week from the struct tm.

  It prompts the user to enter a date (day, month, year) using printf() and scanf().

  It converts the input date to a struct tm using the provided day, month, and year values.

  It converts the struct tm representation of the input date to a time_t value using mktime().

It extracts the day of the week for the input date from the struct tm.

Finally, it prints both the current date and day, and the input date and its corresponding day.

- New variables and operations that take external input from system :

  I. time_t now:

  - This variable is of type time_t and is used to store the current time obtained from the system using the time() function.

  II. struct tm *local:

  - This is a pointer to a struct tm object.
  - It is used to store the current date and time in a more human-readable format.
  - It is obtained by calling the localtime() function with the current time (now) as an argument.

III. char *days[]:

- This is an array of character pointers.
- It stores the names of the days of the week.
- Each element corresponds to a day of the week, starting from Sunday.

IV. int currentDay, int currentMonth, int currentYear, int currentDayOfWeek:

- These variables store the current day, month, year, and day of the week extracted from the struct tm object

V. int inputDay, int inputMonth, int inputYear:
- These variables store the input day, month, and year entered by the user using scanf().

VI. struct tm inputDate:
- This is a struct tm object used to store the input date.
- It is initialized with the input day, month, and year obtained from the user.

VII. time_t inputTime:
- This variable of type time_t is used to store the time_t representation of the input date obtained using mktime().

VIII. struct tm *inputLocal:
- This is a pointer to a struct tm object.
- It is used to store the struct tm representation of the input date obtained from the time_t value inputTime.

IX. int inputDayOfWeek:
- This variable stores the day of the week for the input date extracted from the struct tm object inputLocal.

V. int inputDay, int inputMonth, int inputYear:
- These variables store the input day, month, and year entered by the user using scanf().

VI. struct tm inputDate:
- This is a struct tm object used to store the input date.
- It is initialized with the input day, month, and year obtained from the user.

VII. time_t inputTime:
- This variable of type time_t is used to store the time_t representation of the input date obtained using mktime().

VIII. struct tm *inputLocal:
- This is a pointer to a struct tm object.
- It is used to store the struct tm representation of the input date obtained from the time_t value inputTime.

IX. int inputDayOfWeek:
- This variable stores the day of the week for the input date extracted from the struct tm object inputLocal.

Unit Conversion:

- void lengthConversion() :

This function performs conversions related to length units.
It displays a menu for selecting the conversion type: meters to kilometers, kilometers to meters, inches to centimeters, feet to meters, or yards to meters.
After receiving the user's choice and the value to be converted, it calculates and displays the converted result.
It handles invalid choices gracefully by displaying an error message.

- void weightConversion() :

The weightConversion() function handles conversions related to weight units.
It presents a menu for selecting the conversion type: grams to kilograms, kilograms to grams, ounces to grams, or pounds to kilograms.
Upon receiving the user's choice and the value to convert, it performs the calculation and displays the converted result.
It also provides error handling for invalid choices.

- void temperatureConversion() :

This function handles conversions between Celsius and Fahrenheit temperatures.
It prompts the user to choose the conversion type: Celsius to Fahrenheit or Fahrenheit to Celsius.
After receiving the value to be converted, it performs the calculation and displays the result.
Similar to other conversion functions, it includes error handling for invalid choices.

BMI Calculator

- float calculateBMI(float weight, float height): This is a function definition for a function named calculateBMI. It takes two parameters: weight (of type float) and height (of type float). This function calculates the Body Mass Index (BMI) using the formula BMI = weight / (height * height) and returns the result as a float.

- #define UNDERWEIGHT_THRESHOLD 18.5: This is a preprocessor directive which defines a symbolic constant named UNDERWEIGHT_THRESHOLD with a value of 18.5. It is used to represent the threshold value below which a person is considered underweight according to BMI.

- *#define NORMAL_WEIGHT_THRESHOLD 25*: This is a preprocessor directive which defines a symbolic constant named NORMAL_WEIGHT_THRESHOLD with a value of 25. It is used to represent the threshold value below which a person is considered to have a normal weight according to BMI.

- *#define OVERWEIGHT_THRESHOLD 30*: This is a preprocessor directive which defines a symbolic constant named OVERWEIGHT_THRESHOLD with a value of 30. It is used to represent the threshold value below which a person is considered overweight according to BMI.

Calculator:

- main():

This function serves as the entry point of the program.
It initializes variables, displays a welcome message, and enters an infinite loop to allow the user to perform multiple calculations until they choose to exit.
Within the loop, it prompts the user to enter an operator and performs the corresponding calculation based on the selected operation.

- sin(), cos(), tan():

These functions calculate the trigonometric sine, cosine, and tangent of an angle (in radians) respectively.
They take a single argument (the angle in radians) and return the corresponding trigonometric value.

- asin(), acos(), atan():

These functions are inverse trigonometric functions, which calculate the arcsine, arccosine, and arctangent of a value, respectively.
They take a single argument and return the corresponding angle (in radians).

- fmod():

This function calculates the remainder of dividing one number by another (the modulus operation).
It takes two arguments and returns the remainder after dividing the first argument by the second one.

Splitwise Application:

1. struct User: Represents a user in the expense management system. It contains a `name` field to store the user's name (up to 50 characters) and a `balance` field to track the amount owed or owed by the user.

2. struct Expense: Represents an expense in the system. It includes a `description` field to store a brief description of the expense (up to 100 characters), an `amount` field to record the expense amount, a `num_users` field to track the number of users associated with the expense, and a `users` array to store pointers to users who participated in the expense.

3. addUser: This function adds a new user to the expense management system. It prompts the user to input a name, allocates memory for a new `User` struct, initializes the user's balance to 0.0, adds the user to the `users` array, and increments the count of users (`num_users`). If the maximum number of users (`MAX_USERS`) is reached, it notifies the user that no more users can be added.

4. createExpense: This function allows the creation of a new expense. It prompts the user to input a description and amount for the expense. Then, it iterates through each existing user, asking whether they participated in the expense. If a user participated (`response` is 'y' or 'Y'), it adds that user to the `users` array within the new `Expense` struct and increments the count of users for that expense. After gathering information about participants, it adds the new expense to the `expenses` array and increments the count of expenses (`num_expenses`). It also handles the case when the maximum number of expenses (`MAX_USERS`) is reached.

5. settleExpenses: This function calculates the balance owed by each user based on their participation in expenses. It iterates through each user and calculates the total amount owed by that user by looping through each expense and checking if the user participated in it. If the user participated, their share of the expense is added to the total owed. Finally, it updates the user's balance accordingly.

6. displayBalances: This function displays the current balances of all users. It iterates through each user and prints their name along with their balance. This function is called to provide users with an overview of how much they owe or are owed within the system.

Typing Test Evaluator:

The `case6` function implements a typing test program. It prompts the user to type a predefined passage within a specified time limit. Upon confirmation, it records the start time using `time(NULL)`. The user is then prompted to start typing, and their input is recorded using `fgets`. After completion or reaching the time limit, the end time is recorded. The function calculates elapsed time using `difftime`, and iterates through the passage and user input to count correct characters. It then displays various metrics: elapsed time, total characters typed, correct characters, accuracy percentage, and typing speed in characters per minute. Finally, it recalls the `main` function to restart the program.

Overall, this function provides a simple yet effective tool for measuring typing proficiency by assessing speed and accuracy within a specified time frame, offering users a clear assessment of their typing skills.

Testing and Debugging:

We were facing a large number of errors in our initial runs. This was expected as we were new to these concepts. Solving these errors provided a significant insight of our code which was necessary for understanding the basic underlying concepts. For debugging, we introduced various print statements to understand the flow of code as well as tackled any new error with enthusiasm. Traversing the struct and comparing the value was a major contributor to the errors, these were solved after using the right format specifier as well as logic.

Contribution:

This project was broadly divided under the three members of our group.
Andalib prepared the codes for the Calculator, Unit Conversion Tool and BMI Calculator.
Shreyansh prepared codes for the Splitwise Application and the Typing Speed Evaluator.
Yasas prepared the codes for the Calender and Base System Converter.
Preparation of the report was the result of the combined efforts of Andalib, Shreyansh and Yasas.

What We Learnt:

In developing this C project, we learned to implement diverse functionalities such as expense tracking (Splitwise), typing speed assessment, basic arithmetic operations, unit conversion, base system conversion, and BMI calculation. Through this project, we gained experience in designing a multi-functional utility application, addressing common daily needs within a single platform. We explored various programming concepts including user input handling, algorithm implementation, function design, and modular programming, while emphasizing practicality and user convenience. This project underscores the versatility and applicability of C programming in creating comprehensive utility applications for everyday tasks.

Improvements in programming problems for utility software at an undergraduate level could include:

1. *Complexity*: Introduce problems that require understanding and implementation of algorithms with varying levels of complexity, such as sorting, searching, graph algorithms, etc.

2. *Real-world scenarios*: Design problems that mimic real-world scenarios encountered in utility software development, such as file management, database interactions, network communication, etc.

3. *Error handling*: Include exercises that focus on error handling and validation to teach students how to anticipate and manage errors effectively in their programs.

4. *Performance optimization*: Challenge students to optimize their code for performance and efficiency, encouraging them to consider factors like time complexity, space complexity, and algorithmic optimizations.

5. *Security considerations*: Incorporate security-related challenges to raise awareness about potential vulnerabilities in software, such as input validation, secure data storage, encryption, etc.

6. *Code readability and maintainability*: Emphasize the importance of writing clean, well-structured code by providing tasks that require good code organization, proper documentation, and adherence to coding standards.

7. *Version control*: Integrate exercises that involve version control systems like Git, teaching students how to collaborate on projects, manage code versions, and resolve conflicts.

8. *Testing and debugging*: Include assignments focused on writing unit tests and debugging techniques to help students learn how to verify the correctness of their code and identify and fix errors efficiently.

9. *User interface design*: Introduce projects that involve designing user interfaces for the utility software, helping students understand the importance of user experience and usability in software development.

Conclusion:

In conclusion, the creation of this utility application project in C has been a fulfilling endeavor, aiming to streamline daily tasks within a single platform. By integrating essential tools such as Splitwise for expense management, a Typing Speed Checker, a Calculator, a Unit Converter, a Base System Converter, and a BMI Calculator, this project addresses the practical needs of users. Through the development process, We have honed our programming skills, learned valuable concepts in software design, and gained insight into the versatility of C programming for building comprehensive utility applications. We hope this project proves useful and convenient for users in their daily lives. Overall, this experience will surely help in our future endeavors in the field of Programming and Data Structures.