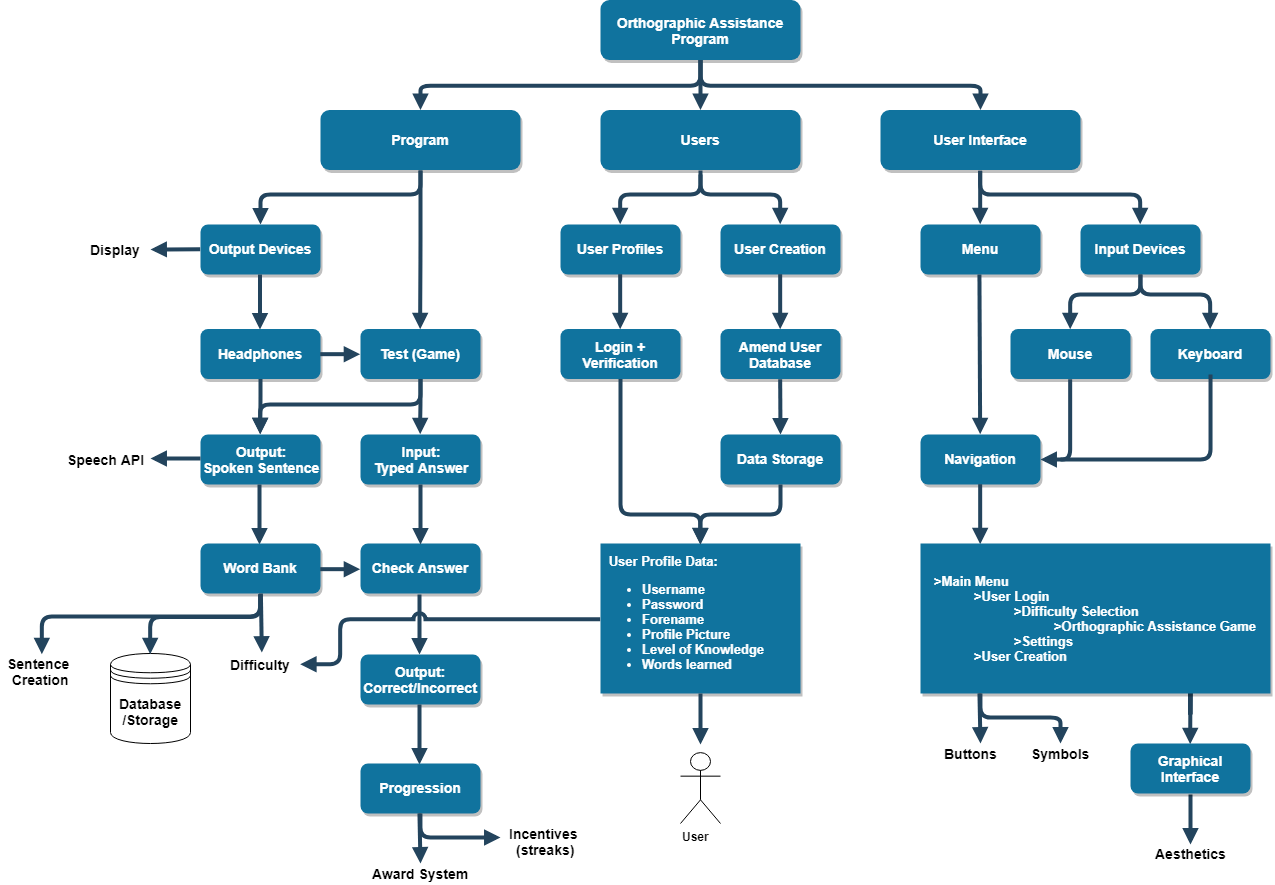
# Decomposing the problem

**3.2.1 Decompose the problem**

**(a) Break down the problem into smaller parts suitable for computational solutions justifying any decisions made.**

In order to compartmentalise the problem, I constructed a HIPO model. The resulting flow chart represents the top down structure of the program, broken down into individual parts of Inputs-Processes-Outputs. I can then use this model to start producing a prototype, as the basic structure is visible and the main features have already been covered. Decomposing the problem also produces smaller, more focused problems that are easier to tackle.



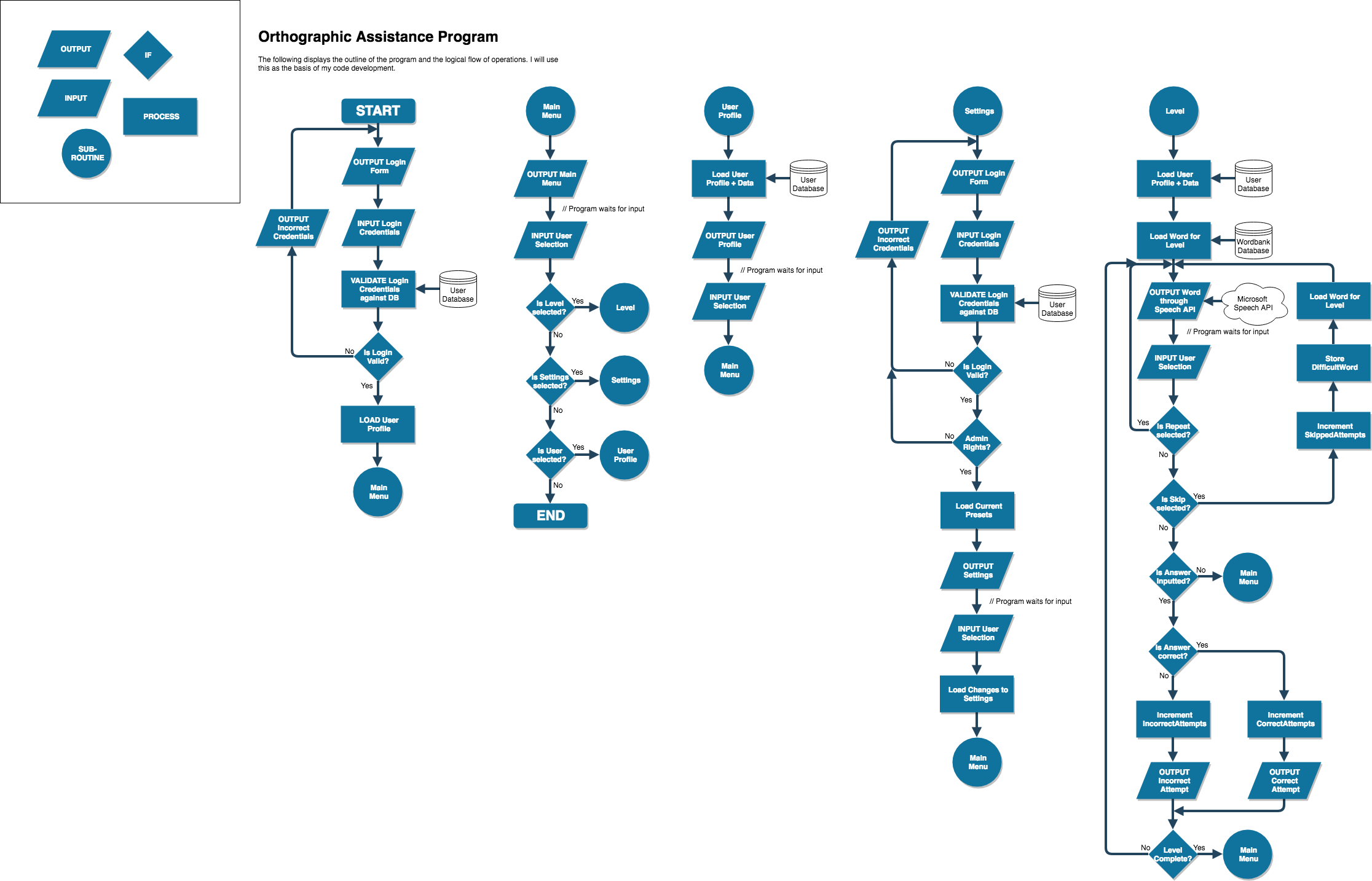
The problem can be computerised, as orthographic assistance can be provided to the user and tested on. Firstly, questions that inform the user on the accuracy of their spelling can be visualized and displayed to them using a graphical interface. A pair of headphones can be used as an output device to provide the user with questions. By using a Speech API, the solution can provide a simulation of another human testing the user on their spelling, which has been proven to be an effective form of orthographic assistance. The questions, or the words that the user is tested on, can also be selected from a large data set, an automated process which prevents the user from knowing the solutions.

This problem may be tackled from a variety of angles, as spellings are not just taught through the testing of knowledge. For example, spellings can be taught through studying their origins and meanings, spotting patterns and irregularities. However, for the purpose of this solution, the problem is abstracted into testing the knowledge of singular words, and allowing them to learn and improve from their mistakes. This method may not be the most effective for every single user, but it still approaches the problem in a way that a computer can realistically interact with a user, through visual outputs. This also makes the solution much smaller in size, as only the tested words need to be included in the program, which conforms to the success criteria. Finally, this method allows the program to be built upon the focused sub-routine of testing singular words, incorporating other features such as user profiles and scores/incentives. The use of a Top Down Design allows the program to be developed in sections, which is easier to comprehend and test.

# The Solution

**3.2.2 Describe the solution**

**(a) Explain and justify the structure of the solution.**

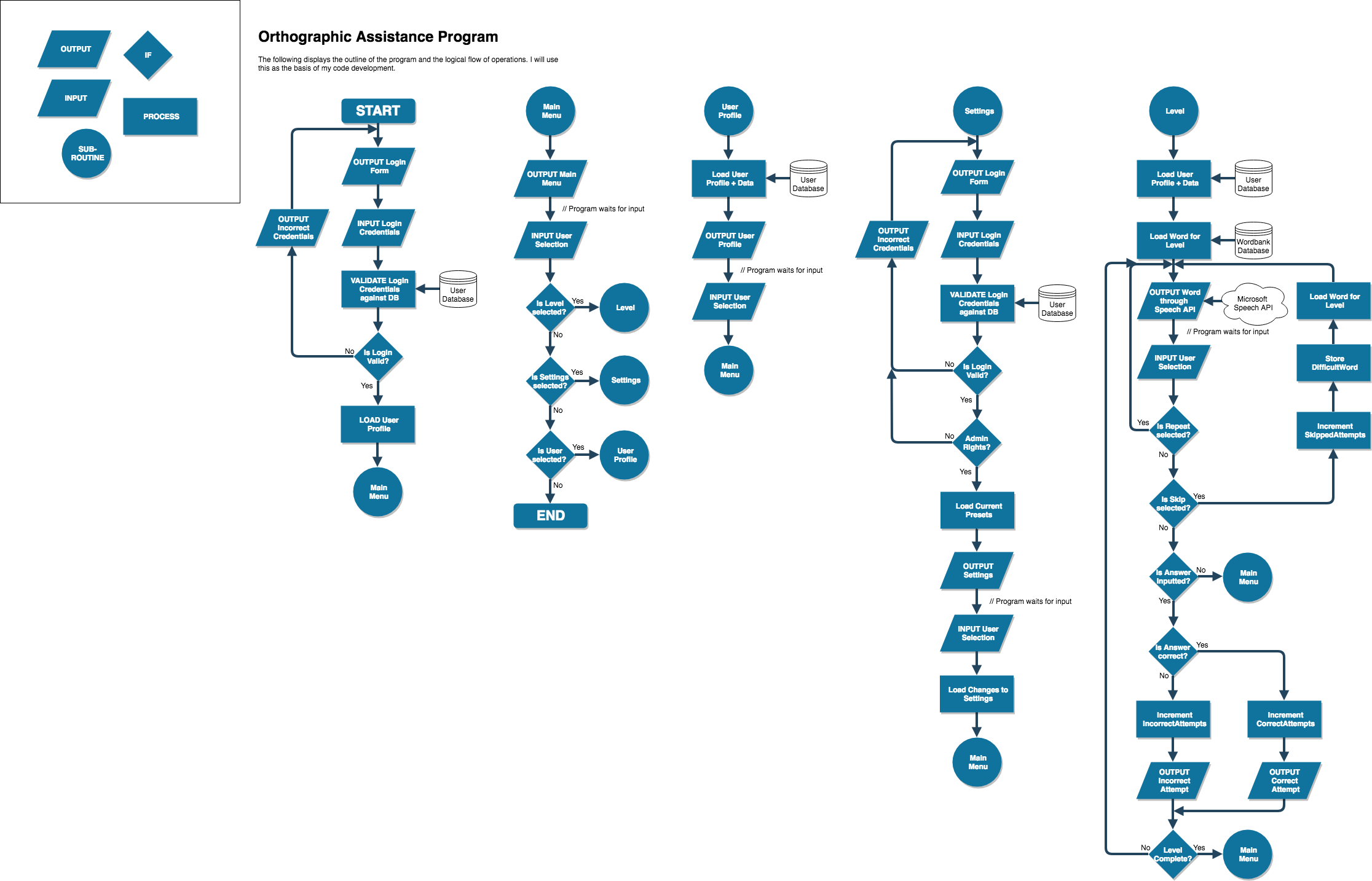


The solution will comprise of a series of subroutines encompassed within a main menu. The user can then make a selection through the graphical interface to access their user profile, settings for the program or a specified ‘level’ in which their spellings will be tested. This will be in accordance to success criteria (3+4), to provide a graphical interface that is easy to follow… The inclusion of subroutines will make it clear to the user where they will want to navigate to in order to carry out a task, without cluttering a single pane with all of the options. As well as this, later maintenance would only need to be carried out on the affected subroutine, as opposed to the entire program, which would follow success criteria (9) to make maintenance feasible.

User profiles will be heavily integrated into the solution. Upon starting the program, the user will have to login using a unique set of credentials in order to gain access. This allows further abstraction to output only the necessary options relevant to the user. Furthermore, this integration will allow the user to only be tested on words that they have not previously inputted correctly, through the use of a user database and a ‘words covered’ entity. This follows success criteria (2+6), to provide a tailored learning experience. The outputs of the solution are dependent on the individual user and their current level of knowledge. This provides an effective form of orthographic assistance.

Therefore, the success criteria laid out in the analysis is achievable through the structure of this solution.

**(b) Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem.**

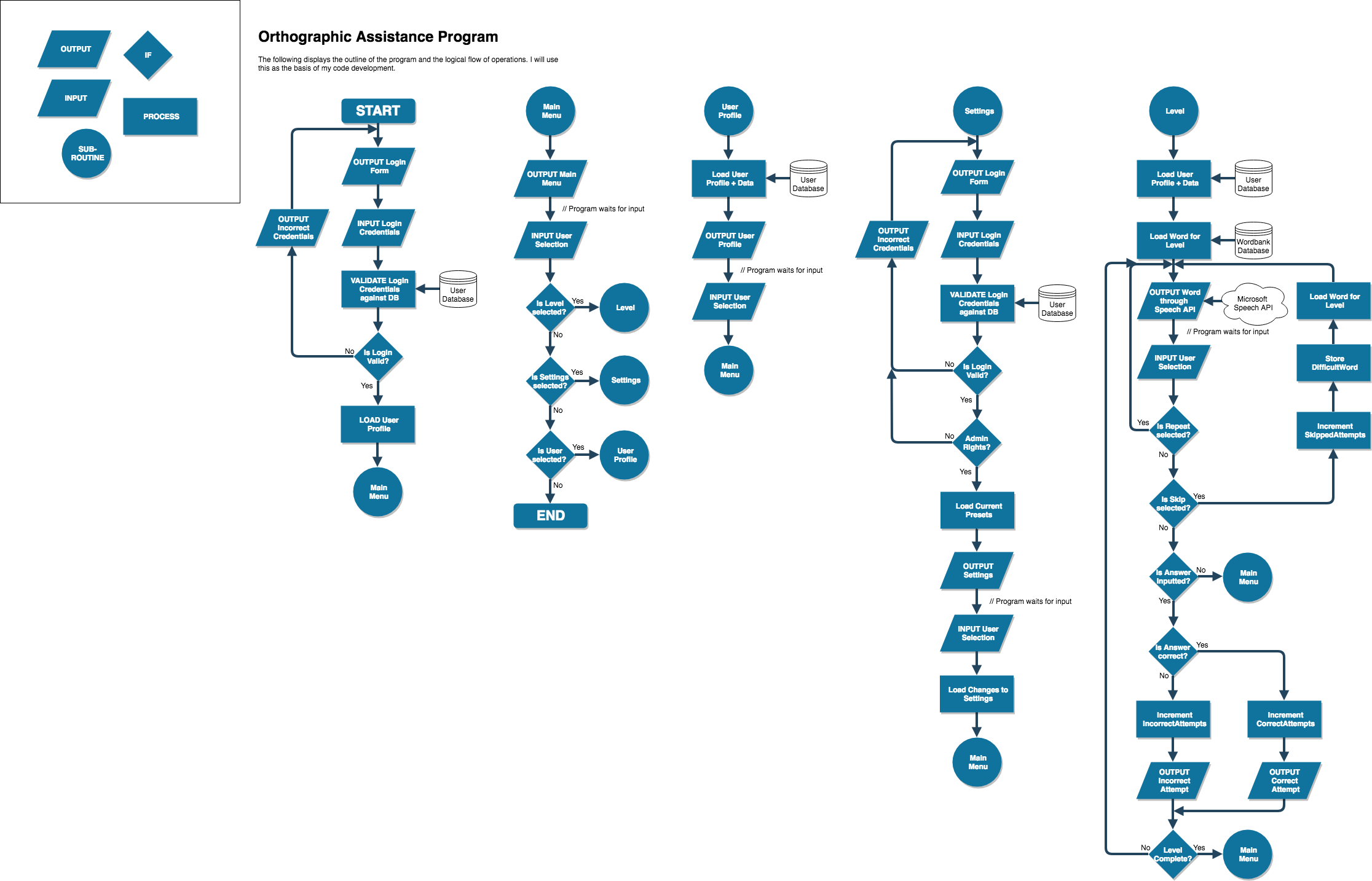


Login

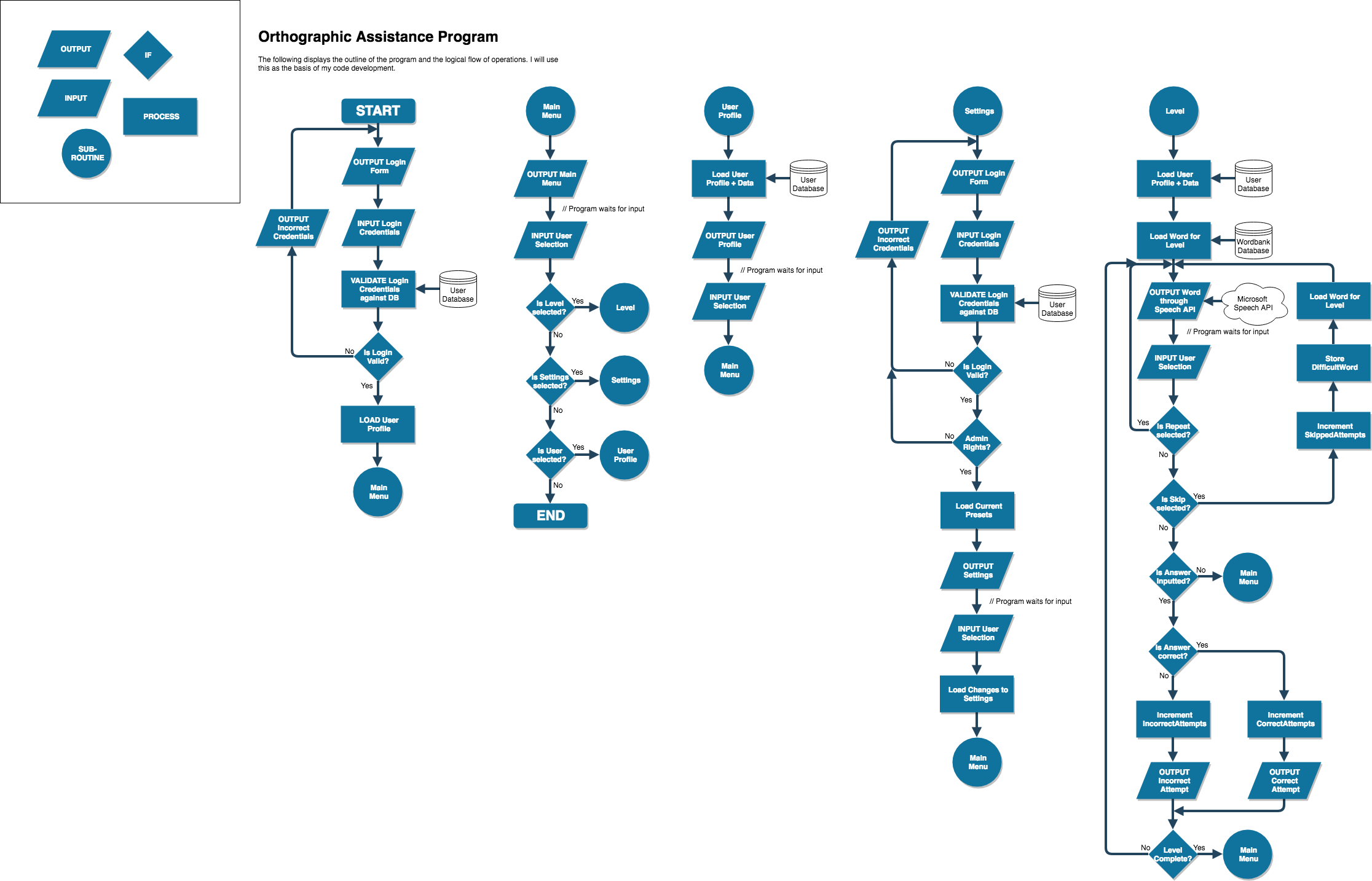
This algorithm is the initial access point of the program. It requires the user to enter their own credentials for the program, validating them against a protected database of accepted users. As well as credentials, the user database contains entities and data which informs the other algorithms of the user’s current level of knowledge. This algorithm also protects against unrestricted access.

Main Menu

This algorithm brings together all of the individual subroutines together so that the user may navigate through them graphically. It is clear what the user needs to interact with in order to access a certain part of the program.

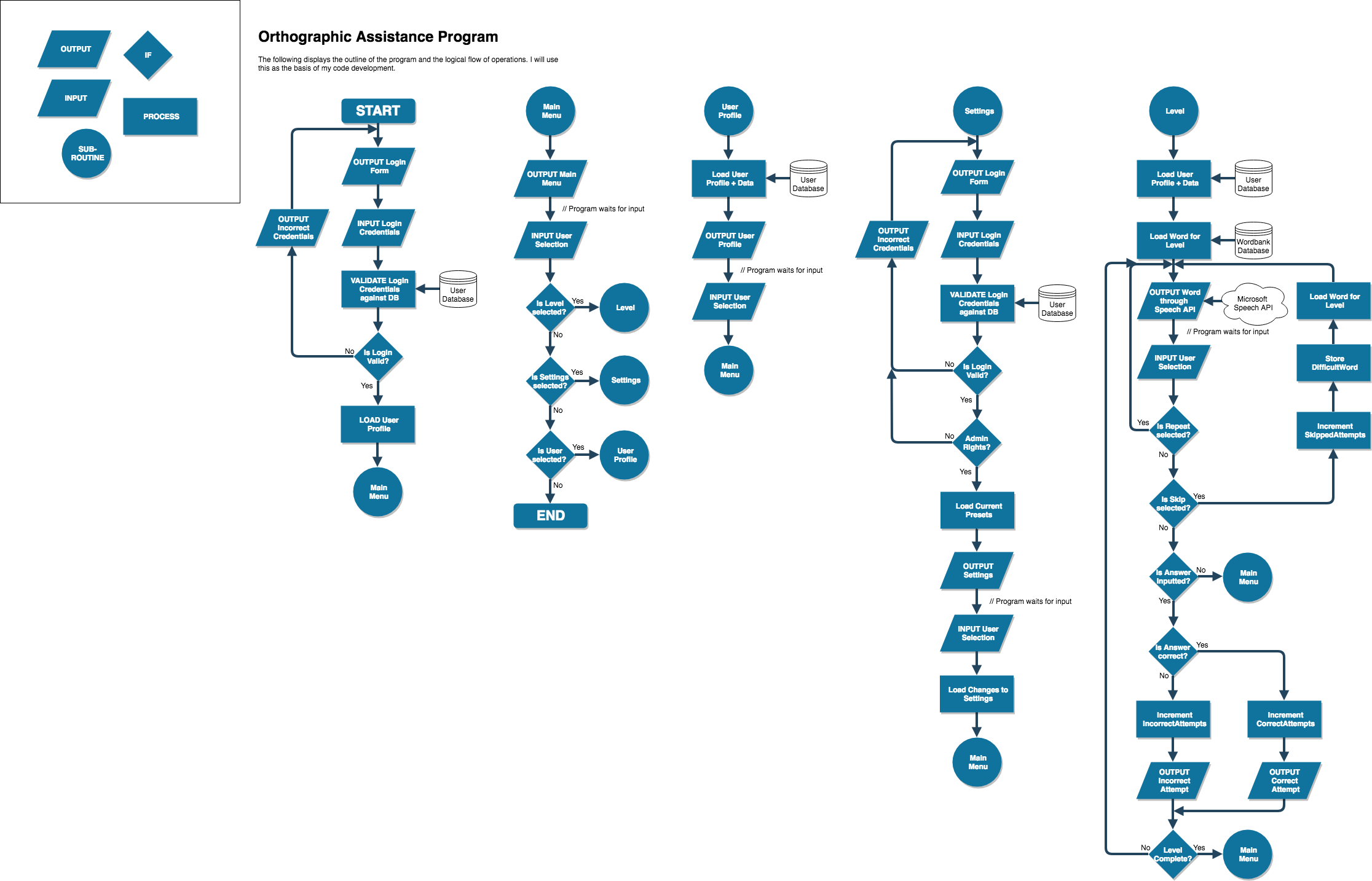


User Profiles

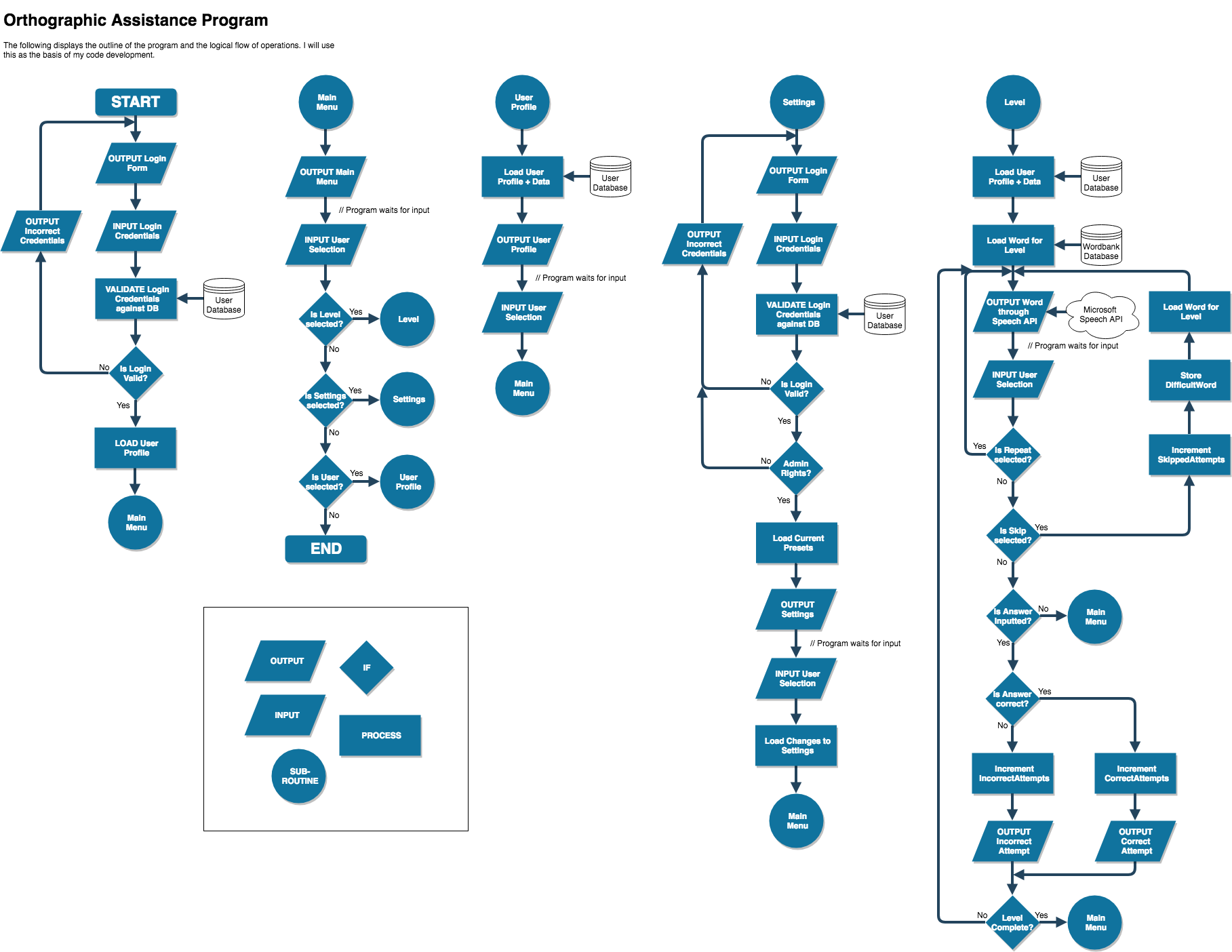
This algorithm serves to provide a graphical representation of the user’s data. When navigated to, this subroutine will display relevant information around the user’s current level of knowledge, extrapolated from the user database. Such information includes a username, profile picture, comparative ‘score’ to other users of the program and a current daily streak. This information provides a competitive incentive to the solution.

Settings

This algorithm allows a user with admin rights to adjust the settings of the program. Intended to be used by a supervisor, this graphical pane displays aspects of the program that can be tailored to a specific user. Adjustments include volume level for headphone output, speech rate etc... Since these settings heavily affect the user experience, they are held behind administrative privileges.

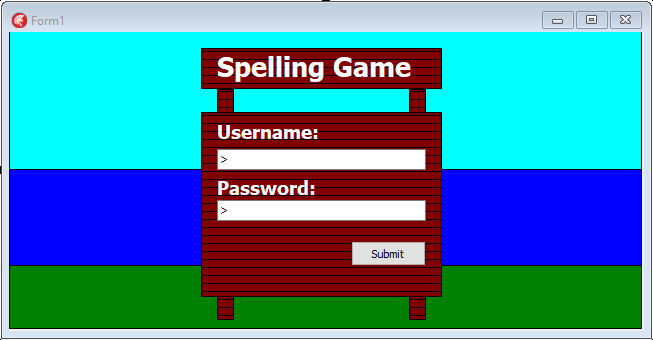
Level

This algorithm is the main aspect of the solution, since it receives data from the other algorithms to provide tailored orthographic assistance. Firstly, the user selects a level by its difficulty, with such levels being displayed to the user based on their current level of knowledge. Difficulty refers to the words tested in the level, based upon factors including word length and spelling irregularity. Words will be fetched from a word-bank database, which contains a large data set. Once the words are loaded, this algorithm will output the word to the user through a pair of headphones, by means of a Speech API. The user will be asked, simply to type the word they hear. Their input will then be tested against the correct word to determine the accuracy of their spelling. This process will be repeated for each question in the level, with options for the user to repeat the output or skip difficult words. In whatever outcome, their attempts are recorded and stored against their user profile. Their performance in these levels will inform the content displayed to them in the future.

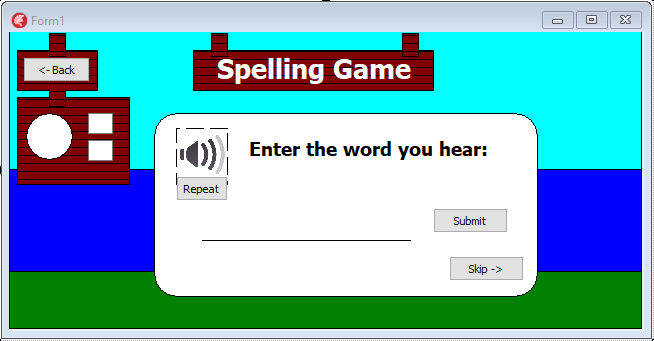


The solution is heavily dependent on a graphical interface, as human input is required to provide an experience relevant to the user.

**(c) Describe usability features to be included in the solution.**







These experimental screen designs show off the aspects of the UI that make the system more efficient to use. For example, buttons allow for shortcuts to move through the program. Instead of opening and closing forms manually, users can more naturally progress through the program. Buttons are clearly captioned with the action attached to them and are easy to identify/understand. Similarly, text boxes are a clear form of data entry and interaction with the program, containing user input to a pre-defined box. Finally, combo-boxes make the system more efficient by displaying all available options to the user in a small space. This removes the need for manual entry, where the user would already need to know their selection, which may not even be available to them. Removing steps of processing makes the system more responsive and more efficient.

Other than the User Interface, the system’s usability is improved by the outputs of the program. This program uses an audio output through a pair of headphones. Other than providing another element of interaction and engagement for the user, the audio output allows the user to be tested on their spellings without the possibility of them seeing a text output. The alternative to this would be to display a series of correct and incorrect spellings and prompt the user to choose the correct one. This method is not as inquisitive as the one in this solution, therefore the outputs improve usability.

**(d) Identify key variables / data structures / classes justifying choices and any necessary validation.**

Main Menu

|  |  |
| --- | --- |
| **Variable** | **Purpose** |
| choice | An integer variable that refers to the selection that the user makes (level, settings etc..). This allows for navigation of the program. |

User Profile

|  |  |
| --- | --- |
| **Variable** | **Purpose** |
| user | A record of strings that is loaded and set from the user database, with information on the user and their progress in the program. This allows data from an external DB to be defined locally, and be accessed much quicker than pulling data directly. |
| loginUsername | A string variable defined as the user input for their username in the login form. This is checked against the correlating username entity in the user DB. |
| loginPassword | A masked string variable defined as the user input for their password in the login form. This is checked against the correlating password and username entity in the user DB. |
| dailyStreak | An integer variable that refers to the amount of days of continuous use of the program. It is incremented for every consecutive day. This is used as an incentive to use the program regularly, and is linked to the user database. |

Settings

|  |  |
| --- | --- |
| **Variable** | **Purpose** |
| adminRights | An integer variable that refers to the access rights the user has been defined. Relevant information is then displayed based on this variable’s value. |
| currentPresets | A record of strings that is loaded and set from within the program. It contains data on the current pre-sets within the settings and their corresponding values. |

Level

|  |  |
| --- | --- |
| **Variable** | **Purpose** |
| word | A string variable that stores the current word being tested on the user. This is processed from the level difficulty and queried in the word-bank database. This variable is not shown to the user. |
| answer | A string variable that stores the inputted guess from the user. This is compared with the word variable to determine the user’s accuracy and success. |
| incorrectAttempts | An integer variable that is incremented for every incorrect attempt made by the user in the current level. It therefore keeps track of the user’s progress and is used to process the user’s overall score. |
| correctAttempts | An integer variable that is incremented for every correct attempt made by the user in the current level. It therefore keeps track of the user’s progress and is used to process the user’s overall score. |
| skippedAttempts | An integer variable that is incremented for every skip performed by the user in the current level. It therefore keeps track of the user’s progress and is used to process the user’s overall score. |

*User Database*

User Table - encrypted

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity** | **Attributes (Fields)** | **Data type** | **PK** | **FK** | **Validation** |
| User | UserID | Number | Yes |  |  |
| Forename | Text |  |  |  |
| Username | Text |  |  | Presence check, Length<12 |
| Password | Text |  |  | Presence check, Length<12, Uppercase/Lowercase |
| DailyStreak | Number |  |  |  |
| Score | Number |  |  |  |
| LastLogin | Date/Time |  |  |  |
| ProfilePicture | Text(Link) |  |  |  |

*Word-bank Database*

Word Table (Flat File)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity** | **Attributes (Fields)** | **Data type** | **PK** | **FK** | **Validation** |
| Level1 | Word | LongText |  |  |  |
| Level2 | Word | LongText |  |  |  |
| … |  |  |  |  |  |

The word-bank flat file DB will contain a large data set of English words sorted into beginner, intermediate and expert depending on word length and spelling irregularity. The program can then fetch words for the relevant level.

Both tables are said to be in 2nd normal form. By extension, they are also in 1st normal form. This means that the data in these tables is organised into distinct rows and columns, where one column (primary key) uniquely identifies each row. Each column contains atomic values, with no repeating groups of columns. Furthermore, in the case of the user table, every column is dependent on the primary key of the table, being the user. This satisfies 2NF. The table would be in 3rd normal form if it contained only columns that are non-transitively dependent on the primary key, using intermediate columns and table relationships. In this case, the solution only needs to store information about the user. With only one entity, the table can be 2NF at most. Furthermore, a link cannot be made to the flat file word table as it must be flat file to interact with the program seamlessly. This normalisation of data is important because it reduces data redundancy, retaining data integrity and compliance to ACID. Redundant data is duplicate or useless data that isn’t required by the program, so normalising also allows the database to be smaller and run quicker.

# Approach to testing

**3.2.3 Describe the approach to testing**

**(a) Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data.**

I will adopt the Iterative Model for the development of this solution. This follows a simple structure of planning, analysis and design, implementation, testing and evaluation. These steps will be followed in order, looping over them for every iteration of the program. The solution will then repeatedly be developed in this order until it is deemed functional and in accordance to the success criteria. Developing in this way generates working software quickly and early during the software life cycle. More importantly, it makes the solution easier to test and debug during a smaller iteration. Each iteration is an easily managed milestone. Another method of development known as the waterfall method follows the same steps as the iterative method, but requires each step to be fully completed before moving on. This works well for smaller projects where requirements are very well understood, but does not allow for a change of scope in mid-development. This solution will be subject to change, so this method is not appropriate

Black box testing assesses the program’s efficiency, without reference to the internal structure of the component or system. Since the solution can be compartmentalised into subroutines, an input can be passed into each and the returned output can be used to assess functionality.

White box testing tests based on an analysis of the internal structure of the component or system. This will be the initial form of testing, with an analysis on the syntax and flow of the program. Whilst most of this can be perceived through a flowchart diagram, later analysis will provide useful insight.

Normal, abnormal and boundary data will be used to test the program’s efficiency and functionality as they will represent real-life inputs. If the program cannot handle boundary data then it is non-functional. These data items will be generated based on the requirements of each algorithm and sub-system. Depending on the output, an analysis can be made.