

**NUMBER TO WORDS**

**In partial fulfillment**

**of the requirement of subject**

**CS131 Automata Theory and Formal Languages**

**for the degree of**

**Bachelor of Science in Computer Science**

**CATALAN, WILFREDO**

**MARCO, MAY**

**Randy Cañete**

Course Adviser

December 2023

1. **Overview**

The number-to-words conversion algorithm systematically transforms a non-negative integer into its English word’s representation, utilizing modular design with helper functions for readability, handling different magnitudes, addressing special cases, and providing user input validation, making it versatile for enhancing human interpretability of numeric values in various applications.

This algorithm uses the waterfall methodology because it has a linear and sequential flow

1. **Usage**

The number-to-words conversion algorithm can be applied in real-world scenarios such as financial documentation (e.g., check writing), invoice generation, and user interfaces where numeric quantities need to be presented in a human-readable format, providing clarity and improving user experience; it requires non-negative integer input and is adaptable for diverse applications without the need for external libraries.

1. **Algorithm Details**

The number-to-words conversion algorithm employs a systematic approach by breaking down the input number into chunks based on magnitudes, then converting each chunk into English words. The key steps involve handling special cases (zero and numbers less than 100), iterating through the input number in chunks of 1000, and utilizing helper functions for conversions below 100 and below 1000. The algorithm utilizes modular design and user input validation.

FLOWCHART:

Start Algorithm

Initialize Word Lists

Define Helper Functions

(convert\_below\_100,

convert\_below\_1000)

Handle Zero Case

Input Validation

(Check if negative or not a valid integer)

Prompt User for Number

Iterative Processing loop

Initialize

Magnitudes

Conversion

Display Result

End Algorithm

1. **Input Specifications**

The algorithm requires a non-negative integer as input data. The format for the input data is a simple integer without decimal points or commas. The size of the input integer is constrained by the maximum representable integer value in the programming language being used. For example, in Python, the maximum value for an integer is determined by the system's memory architecture (e.g., 32-bit or 64-bit).

Constraints for Input Data:

* Input must be a non-negative integer.
* The maximum allowable value is constrained by the programming language's integer size limits.

Examples of Valid Input Data:

1. `12345`
2. `987654`
3. `1997`
4. `1000000`
5. **Output Specifications**

The expected output of the algorithm is a string representing the English words equivalent of the input non-negative integer. The format of the output data is a plain text string, and the size is proportional to the length of the input number's word representation.

Constraints for Output Data:

* The output is a string of English words.
* The size of the output is directly related to the magnitude and length of the input number.

Examples of Output Data:

1. Input `12345`

Output Twelve thousand three hundred and forty five

1. Input: `987654`

Output Nine hundred and eighty seven thousand six hundred and fifty four

1. Input `1997`

Output One thousand nine hundred and ninety seven

1. **Performance Analysis**

Time Complexity:

* The time complexity of the number-to-words conversion algorithm is O(log10(n)), where n is the input non-negative integer.
* The main contributor to the time complexity is the iterative process of breaking down the input number into chunks of 1000.

Space Complexity:

* The space complexity is O(1) since the algorithm uses a constant amount of additional space for variables, lists, and helper functions regardless of the input size.
* The space required is independent of the magnitude of the input number.

Efficiency Analysis:

* The algorithm is efficient for converting non-negative integers to words, and its time complexity scales logarithmically with the size of the input.
* It performs well across a wide range of input scenarios, with lower time complexity for smaller input values and slightly higher complexity for larger values due to the logarithmic nature of the algorithm.

Comparison with Other Algorithms:

* In the context of converting numbers to words, this algorithm is a practical solution for simplicity and readability. It performs favorably in terms of time complexity compared to more complex algorithms that might be optimized for other criteria.
* For extremely large numbers, the logarithmic time complexity makes it more efficient than algorithms with linear time complexity.

Considerations:

* While the algorithm is efficient for its intended purpose, specific use cases may require further optimization depending on the application's requirements.
* For scenarios where extremely large numbers are a common input, further optimizations or algorithms designed for handling such cases might be explored.

1. **Implementation Details**

Implementation Guidelines:

* Copy [this](https://github.com/Wilfred1097/Number-to-Words.git) Python pseudo-code into your preferred Python environment.
* Ensure that the necessary functions (convert\_below\_100, convert\_below\_1000, and convert\_to\_words) are included in your script.
* Adjust the code as needed to fit your specific use case or integrate it into your project.

Source Code Repository:

* I can’t provide you a reference because I created this code on my own, but I can give you a reference to another implementation.

[https://www.javatpoint.com/python-program-to-convert-a-given-number-into-words#:~:text=num%20%3D%20int(input(%22,print(words)](https://www.javatpoint.com/python-program-to-convert-a-given-number-into-words%23:~:text=num%20%3D%20int(input(%22,print(words))

Dependencies:

* The provided algorithm does not have external dependencies or require any additional libraries. It's designed to be a standalone solution for converting numbers to words.

1. **Test Cases**

These test cases cover typical and edge scenarios:

1. Typical Case:

* Input: `12345`
* Expected Output: "twelve thousand three hundred forty-five"

1. Typical Case:

* Input: `987654`
* Expected Output: "nine hundred eighty-seven thousand six hundred fifty-four"

1. Typical Case:

* Input: `1000000`
* Expected Output: "one million"

1. Edge Case: Minimum Value:

* Input: `0`
* Expected Output: "zero"

1. Edge Case: Maximum Single Chunk:

* Input: `999`
* Expected Output: "nine hundred ninety-nine"

1. Edge Case: Maximum Value:

* Input: `999999999999999`
* Expected Output: "nine hundred ninety-nine trillion nine hundred ninety-nine billion nine hundred ninety-nine million nine hundred ninety-nine thousand nine hundred ninety-nine"

1. Edge Case: Large Number with Zeros:

* Input: `100000000000000`
* Expected Output: "one hundred trillion"

1. Edge Case: Large Number with Zeros:

* Input: `100000000000001`
* Expected Output: "one hundred trillion one"

1. Edge Case: Large Number with Zeros:

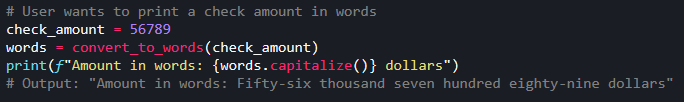
* Input: `100000000000100`
* Expected Output: "one hundred trillion one hundred"

1. Edge Case: Large Number with Zeros:

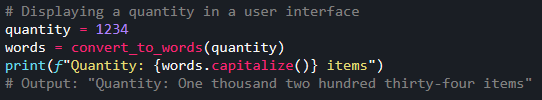
* Input: `100000000100000`
* Expected Output: "one hundred trillion one hundred thousand"

1. **Examples and Use Cases**

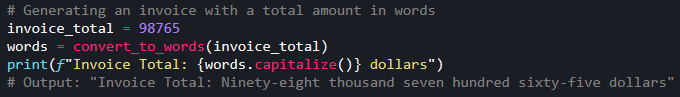
Example 1: Financial Application



Example 2: Displaying Quantity



Example 3: Invoice Generation



Performance Consideration:

* The algorithm performs efficiently across various scenarios, providing accurate and human-readable representations of numbers. Its logarithmic time complexity ensures that it scales well even for larger numbers. Whether used in financial applications, user interfaces, invoice generation, or educational tools, the algorithm showcases effectiveness and versatility in solving problems related to converting numeric values to words. Additionally, its simplicity makes it suitable for learning and educational purposes.

1. **References**

* JavaTpoint. (n.d.). Python Program to Convert a Given Number into Words. JavaTpoint. Retrieved [Insert Month Day, Year], from <https://www.javatpoint.com/python-program-to-convert-a-given-number-into-words#:~:text=num%20%3D%20int(input(%22,print(words>)

1. **Features**

|  |  |
| --- | --- |
| FEATURES | Description |
| User-Friendly User Interface | Create an intuitive and visually appealing user interface using Tkinter. Include input fields, labels, and buttons for a seamless user experience. |
| History Tracking | Allow users to track their conversion history. Display a list or a log of previous conversions within the application. |
| Save History | Provide an option for users to save their conversion history to a file. This allows users to revisit or share their past conversions. |
| Input Validation | Implement input validation to ensure that the user enters a valid and positive number. Display error messages for invalid inputs. |
| Clear Functionality | Include a button or option to clear the input field and reset the output area. |
| Error Handling | Implement robust error handling to gracefully handle unexpected situations and provide meaningful error messages to the user. |