String & Streams

Programming Abstractions for Geoinformatics with C++

Depicted Candela

December 5, 2024

Chapter 4: Streams (GIS Applications Focused on Stream Concepts, Including Skills from Chapter 3: Strings)

1 GIS Applications Focused on Stream Concepts with Strings

1.1 Basic String Manipulation and Output Formatting

Task: Write a program that reads the full name of a city (including spaces) using `getline` and outputs it in uppercase using the `toupper` function from the `<cctype>` library.
Application: Display the city name in a neatly centered format on a 50-character wide line, simulating a header for a map display.

1.2 Reading and Writing GIS Data Files

Task: Create a program that reads a list of place names from a file and writes only the places that start with a vowel to another file. Use functions from Chapter 3 (`startsWith`) and Chapter 4 (file streams).
Application: This could simulate filtering place names for a specific GIS application, like displaying only certain locations on a map.

1.3 Parsing CSV Data for GIS

Task: Implement a function that reads a line of comma-separated values (CSV format) containing coordinates and place names and splits them into individual fields. Use the `getline` function with a delimiter to perform the split.
Application: Write a program that reads multiple lines of CSV data from a file (representing coordinates and place names) and outputs the parsed data to the console in a formatted table, simulating loading and displaying data in a GIS system.

1.4 Advanced String Operations in Stream Contexts

Task: Develop a program that reads a paragraph of text from a file describing geographic features, counts the frequency of each word (ignoring case and punctuation), and writes the results to an output file in descending order of frequency.
Application: This can be used to analyze and summarize the content of GIS metadata or descriptions associated with map layers.

1.5 Stream Manipulators for Number Formatting

Task: Write a program that reads a list of floating-point numbers from a file representing latitude and longitude values, formats them to six decimal places using stream manipulators, and outputs them aligned in columns with specific width.
Application: This simulates formatting geographical coordinates for a printed report or console output in a GIS application.

1.6 Interactive GIS Console Application Using Streams and Strings

Task: Write a command-line tool that continuously accepts user input, processes commands like "count locations", "reverse coordinates", the tools that you desire, and displays results using formatted output.
Application: Utilize streams for interaction and file I/O, and string manipulation functions for processing commands, providing a simple interactive GIS tool for data analysis.

2 Advanced Stream Manipulator Usage in GIS Contexts

2.1 Hexadecimal and Octal Formatting for Coordinate Systems

Task: Write a program that reads a coordinate value (as an integer, for simplicity) and displays it in decimal, hexadecimal, and octal formats using `dec`, `hex`, and `oct` manipulators.
Application: This could be used in debugging low-level GIS systems or map projections that require such formats.

2.2 Scientific Notation and Precision Control in GIS Data

Task: Create a program that reads a floating-point number representing elevation data and displays it using both fixed and scientific notation with a precision of four decimal places.
Application: This simulates displaying geospatial data with appropriate precision and format for scientific analysis in GIS.

2.3 Whitespace Control in GIS Reports

Task: Implement a program that reads a line of text from the user, then reads the same input character by character twice: first using `skipws` (skipping whitespace) and then

using `noskipws` (not skipping whitespace). Output the characters read in both cases to demonstrate the difference.

3 Stream States and Error Handling Mechanisms in GIS

3.1 Stream State Check and Error Recovery in GIS File Processing

Task: Write a program that attempts to read integers representing coordinates from a file until the end of the file is reached. Use stream state-checking methods like `eof()`, `fail()`, `bad()`, and `good()` to handle various errors, and implement recovery mechanisms for non-critical errors.
Application: Demonstrate clearing the error state using `clear()` and explain its significance in the context of GIS data processing.

3.2 Robust Input Handling for GIS Data

Task: Develop a program that prompts the user to enter valid coordinates (floating-point numbers). If the user enters invalid data (e.g., letters or symbols), the program should detect the error using `fail()`, display an error message, clear the stream, and prompt for input again.
Application: This ensures that only valid coordinate data is entered, crucial for maintaining data integrity in GIS applications.

4 Stream Buffer Manipulation in GIS

4.1 Redirecting Output Streams for GIS Logs

Task: Create a program that redirects the standard output to a GIS log file using `rdbuf()` and `ofstream`. Then, demonstrate restoring the standard output back to the console.
Application: Extend this exercise to redirect the standard error (`cerr`) to a log file while keeping regular output on the console, useful for debugging GIS applications.

5 Advanced Formatted Input Operations for GIS

5.1 Extracting Formatted Geospatial Data

Task: Write a program that reads date values representing data collection dates from a formatted string (e.g., "2024-09-17") using `>>` with `setfill` and `setw` to ensure proper extraction and handling of each date component (year, month, day).
Application: Include validation steps to confirm the format is correct, simulating the handling of timestamped GIS data.

6 Working with stringstream in GIS

6.1 Parsing and Formatting Geospatial Data with `stringstream`

Task: Implement a program that reads a line of text containing comma-separated values (e.g., "latitude,longitude,altitude") and uses `stringstream` to parse each value into a vector of strings.
Application: Extend the program to convert numerical string inputs into integers and floating-point numbers, handling potential conversion errors gracefully, which is crucial when dealing with coordinate data in GIS.

7 Binary File I/O for GIS

7.1 Reading and Writing Binary GIS Data

Task: Write a program that opens a binary file and reads data structures (e.g., a struct with an integer for place ID and a double for coordinates) using the `read()` method. Then, modify the data and write it back to another binary file using the `write()` method.
Application: Include checks to handle end-of-file and any I/O errors during the read and write operations, simulating handling of binary GIS data files.

8 Custom Manipulators for GIS

8.1 Creating Custom Manipulators for GIS Coordinates

Task: Design a custom manipulator called `coord` that formats coordinates with a fixed precision and adds a degree symbol after the values (e.g., 12.345678 becomes "12.345678°").
Application: Test the custom manipulator by integrating it into a larger program that formats coordinate data for display in a GIS system.

9 Locale and Internationalization in GIS

9.1 Using Locales for International GIS Data Formatting

Task: Create a program that formats coordinates and dates according to different locales (e.g., US, Germany, Japan). Use the `locale` class to set the locale for input and output streams, demonstrating how the formatting changes.
Application: Specifically, show how different locales handle decimal points, thousands separators, and date formats in GIS applications.

9.2 International Date and Number Parsing for GIS Data

Task: Write a program that reads a date and a coordinate from the user in a locale-specific format (e.g., dates in DD/MM/YYYY for Europe). Use the `locale` class to

correctly parse these inputs and display them in a standardized format.
Application: This ensures compatibility and proper formatting of GIS data in international contexts.