Analysis of C++ for Simple Computation

David Mata

CSCI 4342-001

12-5-24

# Introduction

A quick summary of C++’s popularity would be because of its notable performance and efficiency in numerical computaions and file handling. It also allows for fine control over memory management and execution, such as the use of pointers. The reason why I picked C++ is because of the mentioned popularity. To me it has always been an itimidating language to try to learn, so with the purpose of this report I finally decided it was time to try. Surprisingly, it wasn’t as bad as I thought and this assignment was the best stepping stone into learning C++.

# Description

C++ is a high-level, general-purpose language that follows object-oriented programming principles. C++ is also portable and can be used to devleop applications that can be adapted to various platforms. As the language supports OOP, this means it can use concepts such as inheritance and polymorphism. C++ is a compiled language, where the source code is translated into machine code by the compiler before being executed. Since compiled languages tend to be faster than interpreted languages, I saw this as another reason to choose the language as the evaluation consisted of working with a large data set(i.e. 100,00 numbers). As mentioned above, the use of pointers allows for low-level programming in order to specifically delegate memory and control memory management overall.

# Program Information

Here is some basic information about the C++ file that was used for the evaluation. The file was named *ResearchPaper.cpp:*

* Amount of lines – 131 lines
* File size – 527kb
* Compiled in order to run
* Computation time – 93ms

Since the evaluation also consisted of File I/O, the files *input.txt* and *output.txt* were used in tangent with the C++ file.

# Summary of Results

The program was ran in VS Code, so in order to execute it, it was as simple as pressing the Run button. However, it can also be run in the command line by using the following commands:

g++ -o ResearchPaper ResearchPaper.cpp

./ResearchPaper

This will then read in from the input file(containing the 100,000 numbers) and output the results into the created output file. Those results are as follow:

Results:

Minimum: 1

Maximum: 1000

Mean: 500.66

Median: 499

Mode: 431

Standard Deviation: 289.278

The terminal also prints out the computation time which is as follow:

Computation completed in 93 ms.

# Conclusion

Working on this project highlighted the strengths and challenges of using C++. Its efficiency in processing 100,000 numbers within 93ms demonstrated the performance benefits of a compiled language. The control over memory and precise file I/O operations were invaluable, although they required careful attention to avoid errors like segmentation faults. Which happens when you try to access a location in memory that hasn’t been allocated yet, which is something I haven’t had to worry about in other languages I’ve coded in. Compared to Python or Java, C++ was more tedious in managing syntax and memory but delivered faster execution and lower overhead. Although C++ is a popular language and it is the one I wanted to learn most, I still am not a fan of the syntax. The inclusion of file handling emphasized the language's powerful but lengthy approach to streams, providing flexibility at the cost of simplicity. Overall, while the learning curve was steeper, the experience showed that C++ is a reliable choice for performance-critical tasks, making it a suitable fit for this assignment.

# Appendix

# Sample Execution

g++ -o ResearchPaper ResearchPaper.cpp

./ResearchPaper

# Output

Computation completed in 93 ms.

*Output.txt* file output:

Results:

Minimum: 1

Maximum: 1000

Mean: 500.66

Median: 499

Mode: 431

Standard Deviation: 289.278

# Source Code

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

#include <cmath>

#include <chrono>

#include <unordered\_map>

using namespace std;

using namespace chrono;

// Functions required

void readNumbers(const string &filename, vector<int> &numbers);

void writeResults(const string &filename, int min, int max, double mean, double median, int mode, double stddev);

double calculateMean(const vector<int> &numbers);

double calculateMedian(vector<int> &numbers);

int calculateMode(const vector<int> &numbers);

double calculateStdDev(const vector<int> &numbers, double mean);

int main() {

    string inputFile = "input.txt";

    string outputFile = "output.txt";

    vector<int> numbers;

    // Read numbers from file

    readNumbers(inputFile, numbers);

    // Start timing

    auto start = high\_resolution\_clock::now();

    // Calculate stats

    sort(numbers.begin(), numbers.end());

    int min = numbers.front();

    int max = numbers.back();

    double mean = calculateMean(numbers);

    double median = calculateMedian(numbers);

    int mode = calculateMode(numbers);

    double stddev = calculateStdDev(numbers, mean);

    // End timing

    auto end = high\_resolution\_clock::now();

    auto duration = duration\_cast<milliseconds>(end - start).count();

    // Write results to file

    writeResults(outputFile, min, max, mean, median, mode, stddev);

    // Output timing information

    cout << "Computation completed in " << duration << " ms." << endl;

    return 0;

}

// Function to read numbers from a file

void readNumbers(const string &filename, vector<int> &numbers) {

    ifstream file(filename);

    if (!file.is\_open()) {

        cerr << "Error: Unable to open file " << filename << endl;

        exit(EXIT\_FAILURE);

    }

    int num;

    while (file >> num) {

        numbers.push\_back(num);

    }

    file.close();

}

// Function to write results to a file

void writeResults(const string &filename, int min, int max, double mean, double median, int mode, double stddev) {

    ofstream file(filename);

    if (!file.is\_open()) {

        cerr << "Error: Unable to open file " << filename << endl;

        exit(EXIT\_FAILURE);

    }

    file << "Results:" << endl;

    file << "Minimum: " << min << endl;

    file << "Maximum: " << max << endl;

    file << "Mean: " << mean << endl;

    file << "Median: " << median << endl;

    file << "Mode: " << mode << endl;

    file << "Standard Deviation: " << stddev << endl;

    file.close();

}

// Function to calculate the mean

double calculateMean(const vector<int> &numbers) {

    double sum = 0;

    for (int num : numbers) {

        sum += num;

    }

    return sum / numbers.size();

}

// Function to calculate the median

double calculateMedian(vector<int> &numbers) {

    size\_t size = numbers.size();

    if (size % 2 == 0) {

        return (numbers[size / 2 - 1] + numbers[size / 2]) / 2.0;

    } else {

        return numbers[size / 2];

    }

}

// Function to calculate the mode

int calculateMode(const vector<int> &numbers) {

    unordered\_map<int, int> freq;

    for (int num : numbers) {

        freq[num]++;

    }

    int mode = numbers[0];

    int maxCount = 0;

    for (const auto &pair : freq) {

        if (pair.second > maxCount) {

            maxCount = pair.second;

            mode = pair.first;

        }

    }

    return mode;

}

// Function to calculate the standard deviation

double calculateStdDev(const vector<int> &numbers, double mean) {

    double variance = 0;

    for (int num : numbers) {

        variance += (num - mean) \* (num - mean);

    }

    variance /= numbers.size();

    return sqrt(variance);

}