C/C++ code for G code edit and parser

Creating a G-code editor and parser in C++ involves reading, interpreting, and possibly modifying G-code instructions, which are used to control CNC machines. Below is a simplified example that includes basic functions to read G-code from a file, parse it, and perform some basic editing operations.

**G-Code Editing and Parsing in C++**

1. **Class for Handling G-code**
2. **Functions to Read, Parse, and Edit G-code**
3. **Example Code Execution**

**Code Example**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

// Structure representing a G-code command

struct GCode {

char command;

int number;

std::vector<std::pair<char, float>> parameters;

GCode(char cmd, int num) : command(cmd), number(num) {}

};

// Class to handle G-code operations

class GCodeEditor {

private:

std::vector<GCode> gCodes;

public:

// Read G-code from a file

void loadFromFile(const std::string& filename) {

std::ifstream file(filename);

if (!file.is\_open()) {

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

return;

}

std::string line;

while (std::getline(file, line)) {

parseLine(line);

}

file.close();

}

// Save G-code to a file

void saveToFile(const std::string& filename) {

std::ofstream file(filename);

if (!file.is\_open()) {

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

return;

}

for (const auto& code : gCodes) {

file << code.command << code.number;

for (const auto& param : code.parameters) {

file << " " << param.first << param.second;

}

file << "\n";

}

file.close();

}

// Display G-code

void display() const {

for (const auto& code : gCodes) {

std::cout << code.command << code.number;

for (const auto& param : code.parameters) {

std::cout << " " << param.first << param.second;

}

std::cout << "\n";

}

}

// Parse a line of G-code

void parseLine(const std::string& line) {

std::istringstream iss(line);

char command;

int number;

iss >> command >> number;

GCode gcode(command, number);

char paramType;

float paramValue;

while (iss >> paramType >> paramValue) {

gcode.parameters.emplace\_back(paramType, paramValue);

}

gCodes.push\_back(gcode);

}

// Modify a specific G-code command

void modifyGCode(int index, const GCode& newGCode) {

if (index < 0 || index >= gCodes.size()) {

std::cerr << "Index out of range" << std::endl;

return;

}

gCodes[index] = newGCode;

}

// Add a new G-code command

void addGCode(const GCode& newGCode) {

gCodes.push\_back(newGCode);

}

};

int main() {

GCodeEditor editor;

// Load G-code from a file

editor.loadFromFile("gcode.txt");

// Display loaded G-code

std::cout << "Loaded G-code:\n";

editor.display();

// Modify a G-code command

GCode newGCode('G', 1);

newGCode.parameters.emplace\_back('X', 10.0);

newGCode.parameters.emplace\_back('Y', 20.0);

editor.modifyGCode(0, newGCode);

// Add a new G-code command

newGCode.command = 'M';

newGCode.number = 2;

newGCode.parameters.clear();

newGCode.parameters.emplace\_back('S', 1500);

editor.addGCode(newGCode);

// Display modified G-code

std::cout << "Modified G-code:\n";

editor.display();

// Save modified G-code to a file

editor.saveToFile("modified\_gcode.txt");

return 0;

}

C++

Copy code

**Explanation**

1. **GCode Structure**: Represents a G-code command, storing the command type (like 'G' or 'M'), the number (like '0', '1', '2', etc.), and a list of parameters (like 'X', 'Y' coordinates, etc.).
2. **GCodeEditor Class**: Manages the G-code commands. Functions include:

Copy

Good response

Bad response

continue

Certainly! Let's continue with the explanation of the GCodeEditor class and its methods.

**GCodeEditor Class (Continued)**

* **loadFromFile**: Reads G-code commands from a file, line by line, and parses each line using the parseLine method.
* **saveToFile**: Writes the current list of G-code commands to a file.
* **display**: Outputs the G-code commands to the console for visualization.
* **parseLine**: Parses a single line of G-code into its constituent parts and stores it in the gCodes vector.
* **modifyGCode**: Modifies an existing G-code command at the specified index with a new G-code command.
* **addGCode**: Adds a new G-code command to the list of commands.

**Example Execution**

Here's an example outline of how the main function works:

int main() {

GCodeEditor editor;

// Load G-code from a file

editor.loadFromFile("gcode.txt");

// Display loaded G-code

std::cout << "Loaded G-code:\n";

editor.display();

// Modify a G-code command (for example, the first command)

GCode newGCode('G', 1);

newGCode.parameters.emplace\_back('X', 10.0);

newGCode.parameters.emplace\_back('Y', 20.0);

editor.modifyGCode(0, newGCode);

// Add a new G-code command

newGCode.command = 'M';

newGCode.number = 2;

newGCode.parameters.clear();

newGCode.parameters.emplace\_back('S', 1500);

editor.addGCode(newGCode);

// Display modified G-code

std::cout << "Modified G-code:\n";

editor.display();

// Save modified G-code to a file

editor.saveToFile("modified\_gcode.txt");

return 0;

}

C++

Copy code

**Sample G-code File (gcode.txt)**

To test the program, you can create a simple G-code text file named gcode.txt with the following content:

G0 X0 Y0

G1 X10 Y10 Z5

M3 S1000

M5

G0 X0 Y0

* **Loading G-code**: The loadFromFile method reads the file gcode.txt, parses each line using parseLine, and stores it in the internal gCodes vector.
* **Displaying G-code**: The display method outputs the loaded commands to the console.
* **Modifying G-code**: The example modifies the first G-code command (G0) to G1 X10.0 Y20.0.
* **Adding New G-code**: Adds a new command M2 S1500 to the end of the list.
* **Saving Modified G-code**: Saves the modified G-code to a new file modified\_gcode.txt.

**Note**

* **Error Handling**: The code example includes basic error handling for file operations. Depending on the usage, more sophisticated error handling and validation might be necessary.
* **Parameter Handling**: This example assumes all parameters are separated by whitespace. Depending on the specific G-code syntax you are dealing with, you might need to adjust the parseLine method.

This example provides a foundation you can expand upon for more specific and complex G-code handling requirements. If you need additional features or further customization, let me know!