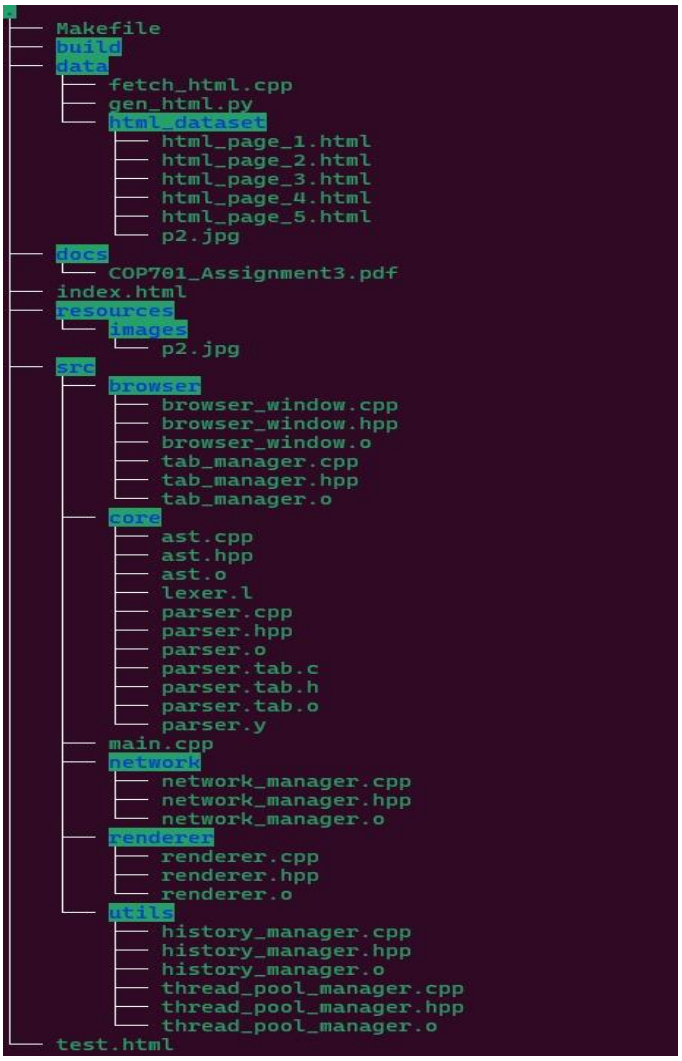
Mini Web Browser-Horcrux Report

1. Project Overview

In this project we implemented a custom HTML parser and Renderer that can parse HTML documents into an Abstract Syntax Tree (AST) and render them in a Qt-based graphical user interface.

2. Project Structure



```
├── Makefile
├── build
├── data
│   ├── fetch_html.cpp
│   ├── gen_html.py
│   └── html_dataset
│       ├── html_page_1.html
│       ├── html_page_2.html
│       ├── html_page_3.html
│       ├── html_page_4.html
│       ├── html_page_5.html
│       └── p2.jpg
├── docs
│   └── COP701_Assignment3.pdf
├── index.html
├── resources
│   └── images
│       └── p2.jpg
├── src
│   ├── browser
│   │   ├── browser_window.cpp
│   │   ├── browser_window.hpp
│   │   ├── browser_window.o
│   │   ├── tab_manager.cpp
│   │   ├── tab_manager.hpp
│   │   └── tab_manager.o
│   ├── core
│   │   ├── ast.cpp
│   │   ├── ast.hpp
│   │   ├── ast.o
│   │   ├── lexer.l
│   │   ├── parser.cpp
│   │   ├── parser.hpp
│   │   ├── parser.o
│   │   ├── parser.tab.c
│   │   ├── parser.tab.h
│   │   ├── parser.tab.o
│   │   └── parser.y
│   ├── main.cpp
│   ├── network
│   │   ├── network_manager.cpp
│   │   ├── network_manager.hpp
│   │   └── network_manager.o
│   ├── renderer
│   │   ├── renderer.cpp
│   │   ├── renderer.hpp
│   │   └── renderer.o
│   └── utils
│       ├── history_manager.cpp
│       ├── history_manager.hpp
│       ├── history_manager.o
│       ├── thread_pool_manager.cpp
│       ├── thread_pool_manager.hpp
│       └── thread_pool_manager.o
└── test.html
```

3. Architecture

- ❖ **Initial Setup:** **HTML Page Fetching** : The HTML fetching module uses *libcurl* to send *HTTP GET* requests and retrieve HTML content from a local server. The fetched content is stored in a string for further processing.
- ❖ *fetchHTML():* Initializes a CURL session and retrieves HTML content

2.1 Core Components

1) **Lexer (*lexer.l*)**
   - Tokenizes HTML input using Flex
   - *Tokens: HTML_TAG, TITLE_TAG, P_TAG, H1_TAG, IMG_TAG, TEXT_CONTENT, etc.*

- Recognizes HTML tags, attributes, and text content
- Provides tokens to the parser

2) **Parser (*parser.y*)**
   - Implements a Bison/Yacc grammar for HTML parsing
   - Creates an Abstract Syntax Tree (AST) representation
   - Handles nested HTML elements and attributes
   - parseHTML(): Initiates parsing and constructs the DOM tree.
   - addChild(): Adds a child node to the current DOM element.

3) **AST (*ast.hpp/cpp*)**
   - Defines the tree structure for HTML representation
   - Manages node types and relationships
   - Provides methods for tree traversal and manipulation
   - Node Classes:
     - **ElementNode**: Represents HTML elements with tag names and child nodes.
     - **TextNode**: Represents text content within HTML elements.
   -
4) **Renderer (*renderer.hpp/cpp*)**
   - Qt-based rendering engine
   - The rendering engine is implemented using Qt Widgets. It traverses the DOM tree and displays HTML content such as paragraphs, headers, and images on a GUI window.
   - Converts AST to visual representation
   - Handles different HTML elements (headings, paragraphs, lists, etc.) text formatting (bold, italic, underline), Headings (h1-h6), Lists (ordered and unordered), Links with click handling, Images with alt text, Paragraphs with proper spacing, Code blocks, Blockquotes with indentation.

2.2 Supporting Components

1) **Browser Window (*browser_window.hpp/cpp*)**
   - Main application window
   - URL bar and navigation controls (fwd/bwd)
   - Tab management
2) **Thread Pool Manager (*thread_pool_manager.cpp*)**
   - Manages concurrent parsing operations
   - Prevents UI blocking during parsing
   - Callback-based result handling
3) **Network Manager (*network_manager.cpp*)**
   - Handles URL fetching
   - Supports both local and remote content
   - Synchronous network operations like- file loading operations, resource fetching, content processing, error handling, memory allocation for content etc.

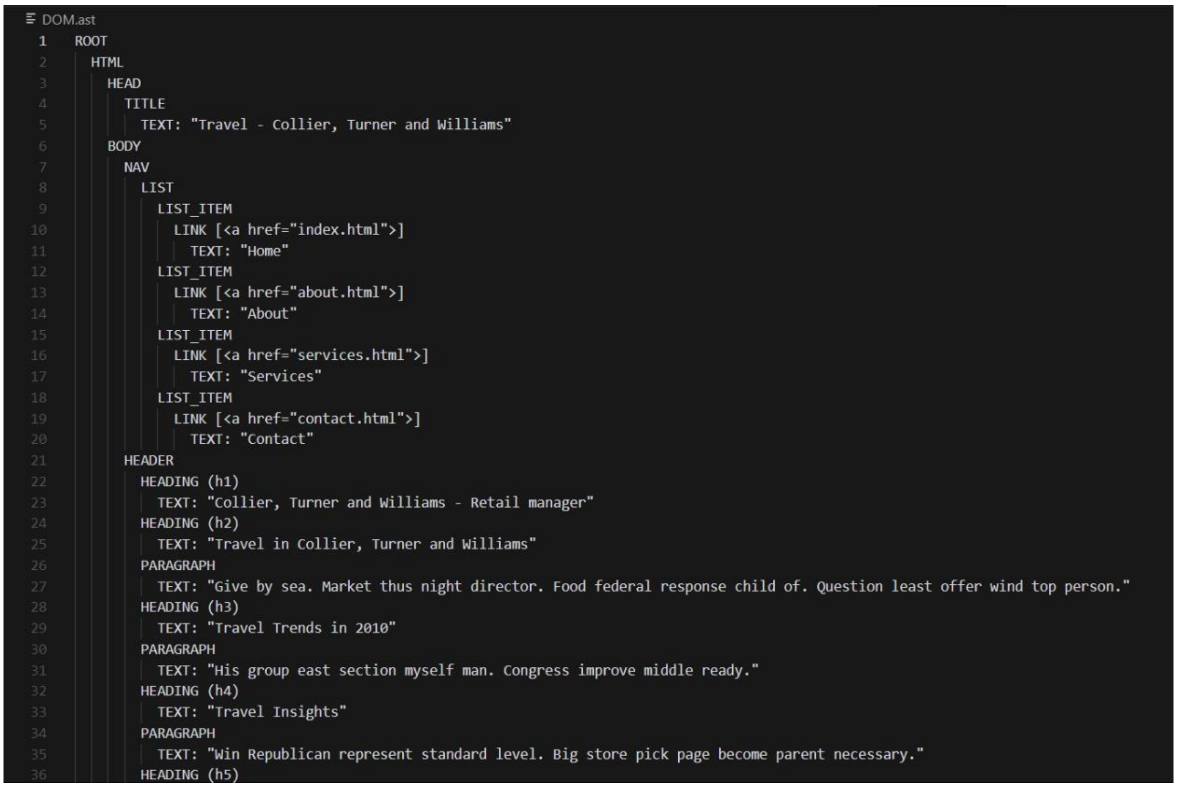
4. Implementation Details

❖ **DOM Inplementaion(AST):**

-Uses Bison/Yacc for grammar definition

- Creates AST nodes during parsing

- Handles nested structures recursively



```
DOM.ast
ROOT
  HTML
    HEAD
      TITLE
        TEXT: "Travel - Collier, Turner and Williams"
    BODY
      NAV
        LIST
          LIST_ITEM
            LINK [<a href="index.html">]
              TEXT: "Home"
          LIST_ITEM
            LINK [<a href="about.html">]
              TEXT: "About"
          LIST_ITEM
            LINK [<a href="services.html">]
              TEXT: "Services"
          LIST_ITEM
            LINK [<a href="contact.html">]
              TEXT: "Contact"
      HEADER
        HEADING (h1)
          TEXT: "Collier, Turner and Williams - Retail manager"
        HEADING (h2)
          TEXT: "Travel in Collier, Turner and Williams"
        PARAGRAPH
          TEXT: "Give by sea. Market thus night director. Food federal response child of. Question least offer wind top person."
        HEADING (h3)
          TEXT: "Travel Trends in 2010"
        PARAGRAPH
          TEXT: "His group east section myself man. Congress improve middle ready."
        HEADING (h4)
          TEXT: "Travel Insights"
        PARAGRAPH
          TEXT: "Win Republican represent standard level. Big store pick page become parent necessary."
        HEADING (h5)
```

❖ **Renderer Implementation**

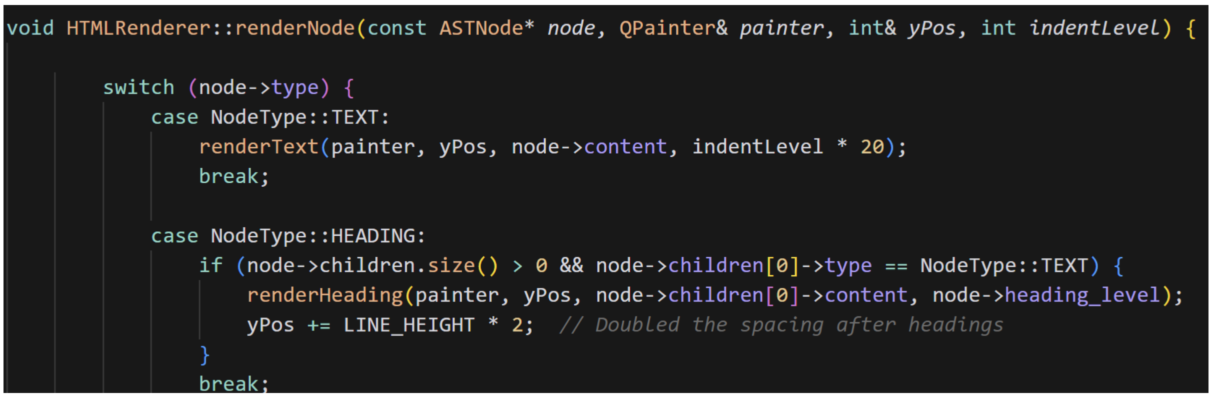
- Qt-based painting system

- The HTML Renderer class uses *QPainter* to draw text and other elements directly onto the widget, indicating a manual rendering process.

- Hierarchical rendering
- Layout management
- Event handling
- **Key Functions**:
- `renderNode()`: Recursively renders each node from the DOM tree onto the GUI.
- `show()`: Displays the GUI window with the rendered HTML content.

Rendering Components:

- QPainter: Core rendering
- QColor: Color management
- QRect: Layout management



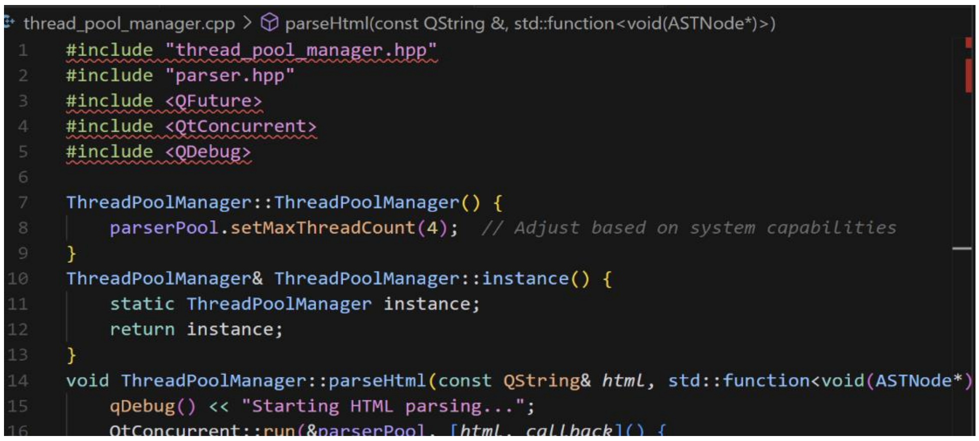
```cpp
void HTMLRenderer::renderNode(const ASTNode* node, QPainter& painter, int& yPos, int indentLevel) {

        switch (node->type) {
            case NodeType::TEXT:
                renderText(painter, yPos, node->content, indentLevel * 20);
                break;

            case NodeType::HEADING:
                if (node->children.size() > 0 && node->children[0]->type == NodeType::TEXT) {
                    renderHeading(painter, yPos, node->children[0]->content, node->heading_level);
                    yPos += LINE_HEIGHT * 2;  // Doubled the spacing after headings
                }
                break;
```

- QScrollArea: Content scrolling
- QLabel: Only used for simple text or image display within the UI (for rendering)

- **<u>Multi-Process and Multi-Threaded Support</u>**

1) <u>Thread Pool Management</u>

- Singleton pattern for global thread pool access
- Configurable thread count (default: 4 threads)
- Asynchronous HTML parsing
- Non-blocking UI operations
- Thread-safe callback handling

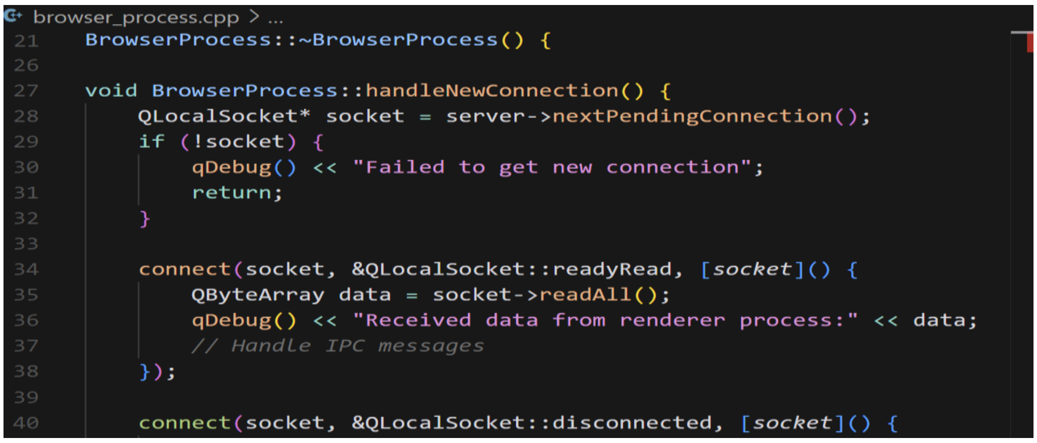


thread_pool_manager.cpp > parseHtml(const QString &, std::function<void(ASTNode*)>)

```cpp
#include "thread_pool_manager.hpp"
#include "parser.hpp"
#include <QFuture>
#include <QtConcurrent>
#include <QDebug>

ThreadPoolManager::ThreadPoolManager() {
    parserPool.setMaxThreadCount(4);  // Adjust based on system capabilities
}
ThreadPoolManager& ThreadPoolManager::instance() {
    static ThreadPoolManager instance;
    return instance;
}
void ThreadPoolManager::parseHtml(const QString& html, std::function<void(ASTNode*)
    qDebug() << "Starting HTML parsing...";
    QtConcurrent::run(&parserPool, [html, callback]() {
```

2) <u>Multi-Process Architecture</u>

- IPC (Inter-Process Communication) server
- Socket-based communication
- Process isolation for security
- Error handling for process management



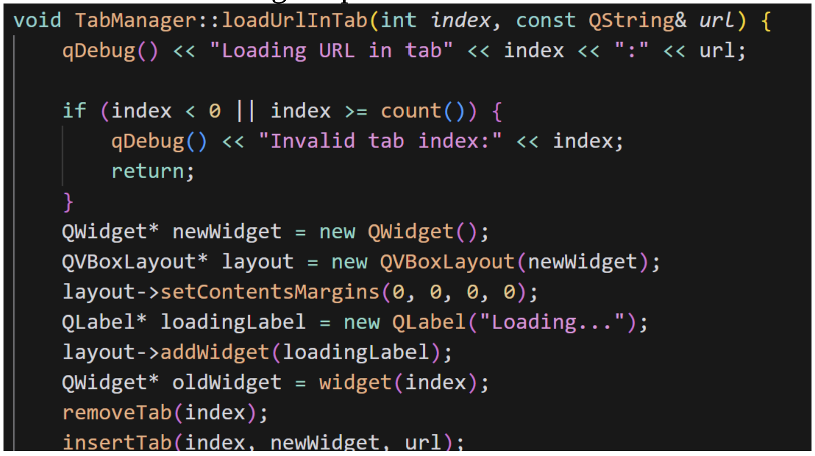
browser_process.cpp > ...

```cpp
BrowserProcess::~BrowserProcess() {

void BrowserProcess::handleNewConnection() {
    QLocalSocket* socket = server->nextPendingConnection();
    if (!socket) {
        qDebug() << "Failed to get new connection";
        return;
    }

    connect(socket, &QLocalSocket::readyRead, [socket]() {
        QByteArray data = socket->readAll();
        qDebug() << "Received data from renderer process:" << data;
        // Handle IPC messages
    });

    connect(socket, &QLocalSocket::disconnected, [socket]() {
```

3) <u>Tab Management</u>

- Independent process per tab
- Process lifecycle management
- Clean process termination

- Error handling for process failures



```
void TabManager::loadUrlInTab(int index, const QString& url) {
    qDebug() << "Loading URL in tab" << index << ":" << url;

    if (index < 0 || index >= count()) {
        qDebug() << "Invalid tab index:" << index;
        return;
    }
    QWidget* newWidget = new QWidget();
    QVBoxLayout* layout = new QVBoxLayout(newWidget);
    layout->setContentsMargins(0, 0, 0, 0);
    QLabel* loadingLabel = new QLabel("Loading...");
    layout->addWidget(loadingLabel);
    QWidget* oldWidget = widget(index);
    removeTab(index);
    insertTab(index, newWidget, url);
```

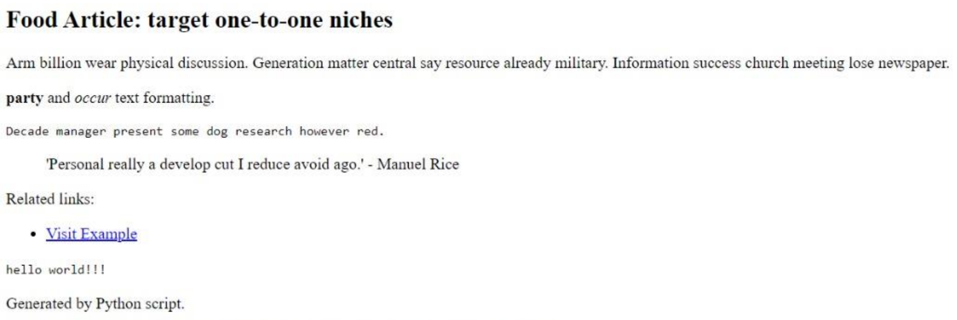
5. <u>Bonus Implemented</u>

i. Scrolling feature in renderer GUI
ii. Maintained a browser history
iii. Forward & backward move through browser history
iv. An option to save the browser session in the setting bar in GUI
v. Link redirects to the page (local) on click

6. <u>Testing and Validation</u>

- Sample HTML files provided to validate the result
- DOM tree output validation
- Visual rendering verification
- Error handling testing

<u>Actual HTML input</u>:



Food Article: target one-to-one niches

Arm billion wear physical discussion. Generation matter central say resource already military. Information success church meeting lose newspaper.

party and *occur* text formatting.

`Decade manager present some dog research however red.`

> 'Personal really a develop cut I reduce avoid ago.' - Manuel Rice

Related links:

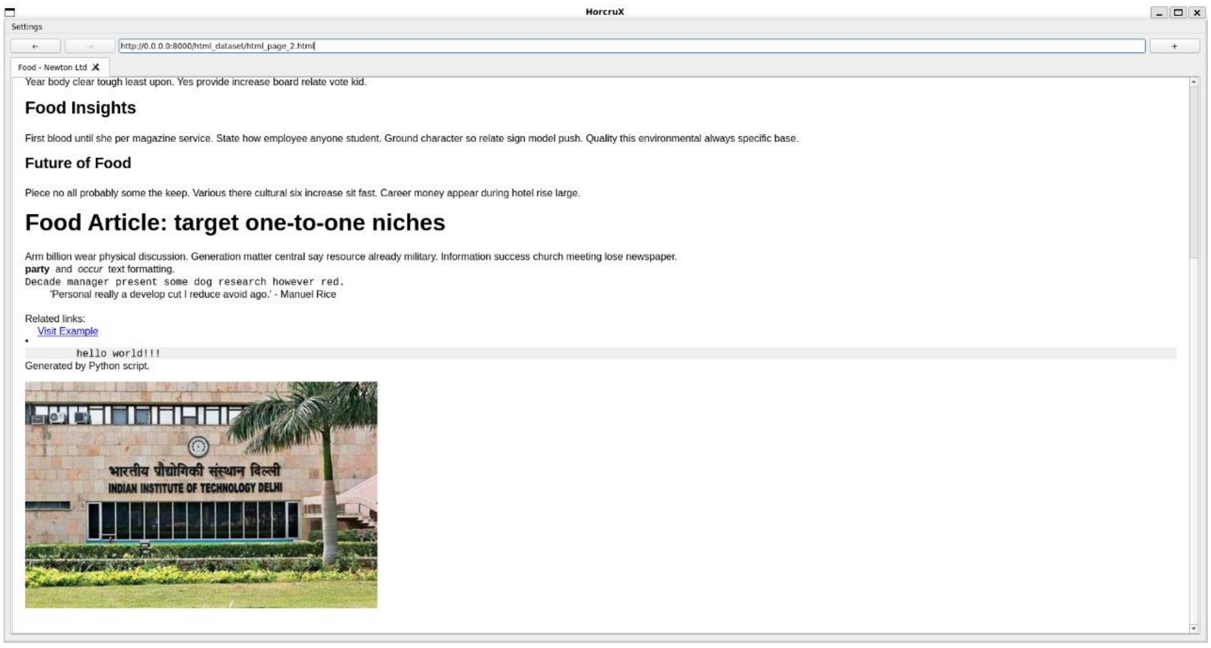
- [Visit Example]

`hello world!!!`

Generated by Python script.

भारतीय प्रौद्योगिकी संस्थान दिल्ली
INDIAN INSTITUTE OF TECHNOLOGY DELHI

Renderer GUI output:



http://0.0.0.0:8000/html_dataset/html_page_2.html

Year body clear tough least upon. Yes provide increase board relate vote kid.

Food Insights

First blood until she per magazine service. State how employee anyone student. Ground character so relate sign model push. Quality this environmental always specific base.

Future of Food

Piece no all probably some the keep. Various there cultural six increase sit fast. Career money appear during hotel rise large.

Food Article: target one-to-one niches

Arm billion wear physical discussion. Generation matter central say resource already military. Information success church meeting lose newspaper.
party and *occur* text formatting.
`Decade manager present some dog research however red.`
'Personal really a develop cut I reduce avoid ago.' - Manuel Rice

Related links:
Visit Example
•

```
        hello world!!!
```

Generated by Python script.

भारतीय प्रौद्योगिकी संस्थान दिल्ली
INDIAN INSTITUTE OF TECHNOLOGY DELHI

7. Conclusion

- The project successfully implements a basic HTML parser and renderer with essential features for web content display.
- The GUI successfully displays the title, heading, paragraph, and image as specified in the input HTML.
- The modular architecture allows for future extensions and improvements while maintaining code maintainability.

8. References Used

- HTML5 Specification
- libcurl Documentation: https://curl.se/libcurl/
- Flex/Bison Documentation, Flex manual: https://westes.github.io/flex/manual/
- Modern C++ Features- https://devdocs.io/cpp/
- Qt Documentation
- Qt Concurrent: https://doc.qt.io/qt-6/qtconcurrent-index.html
- Qt Network: https://doc.qt.io/qt-6/qtnetwork-index.html
- Browser architecture design patterns
- Chromium Design Documents: https://www.chromium.org/developers/design-documents/
- "C++ Concurrency in Action" by Anthony Williams for Multi-threading and Concurrency
- For performance & optimization we referred- Valgrind Documentation: https://valgrind.org/docs/