Instituto Tecnológico de Costa Rica

Área académica de Ingeniería en Computadores

Proyecto 1: C!

Documentación

Profesor: Antonio González Torres

Estudiantes:

- Jose Antonio Espinoza Chaves 2019083698
- Eduardo Zumbado Granados 2019003973

I Semestre 2021

Índice:

Contents

Enlaces a los repositorios:

Enlace al repositorio del IDE: https://github.com/JoseA4718/Proyecto-1-Datos-II-C

Enlace al repositorio del Servidor: https://github.com/Eduardo2108/C-_Server

Descripción del problema

C! Es el primer proyecto del curso CE Algoritmos y Estructuras de Datos II, de la carrera de Ingeniería en Computadores del Instituto Tecnológico de Costa Rica. C! es un ambiente de programación para el pseudo lenguaje C! junto con un manejador de memoria para el mismo. Este consta de tres mayores secciones, un IDE donde el usuario pueda escribir código, ejecutarlo, revisar el estado de la memoria alojada en el servidor, así como las respuestas que el servidor envía por cada acción en el IDE. La otra sección es un servidor, este se encarga del manejo de memoria y de cualquier cambio en las variables que el usuario pida desde el cliente o IDE. Por último, se encuentra el memory collector, este se encarga de revisar qué dirección de memoria en el servidor es dejada de utilizar, de manera que la reserva para la siguiente variable en crearse.

En este primer proyecto de Algoritmos y Estructuras de Datos 1, uno de los principales objetivos va a ser la correcta implementación de las diferentes estructuras como colas, pilas y listas; y patrones de diseño utilizados como el singleton y facade.

Diagramas de Clases

Servidor

Para visualizar el lucid chart del diagrama del servidor favor diigirse a:
https://lucid.app/lucidchart/invitations/accept/inv_a3909a2e-d93e-4e75-a25a-b0fef413b634?viewport_loc=-1899%2C-2110%2C3048%2C1442%2C0_0

Para verlo con mas detalle puede ir al siguiente enlace con formato .jpg:
https://cdn.discordapp.com/attachments/772625100896862209/837199804546678804/C-EDITOR_-_Server.jpeg

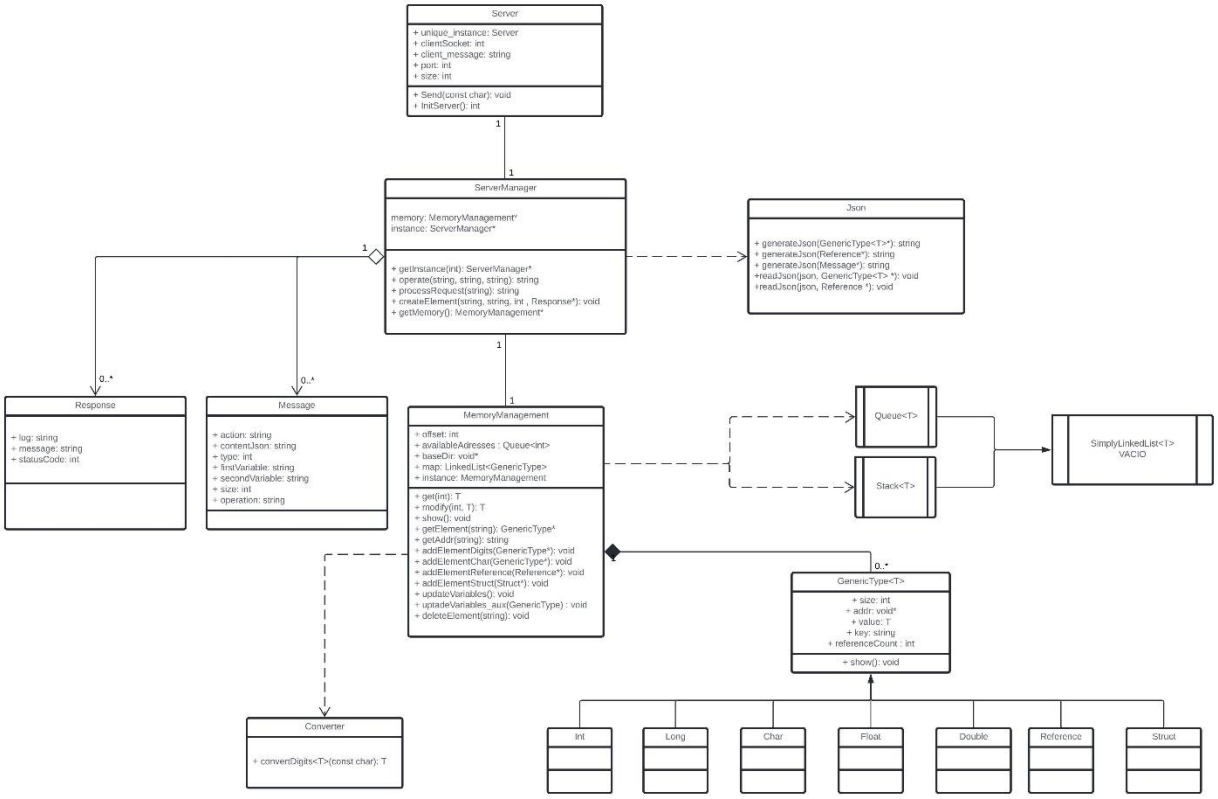
Cliente

Para visualizar el lucid chart del diagrama del cliente favor dirigirse a:
https://lucid.app/lucidchart/d3f615bf-5b74-4a71-a380-aa3c5ec852dc/edit?shared=true&page=0_0#

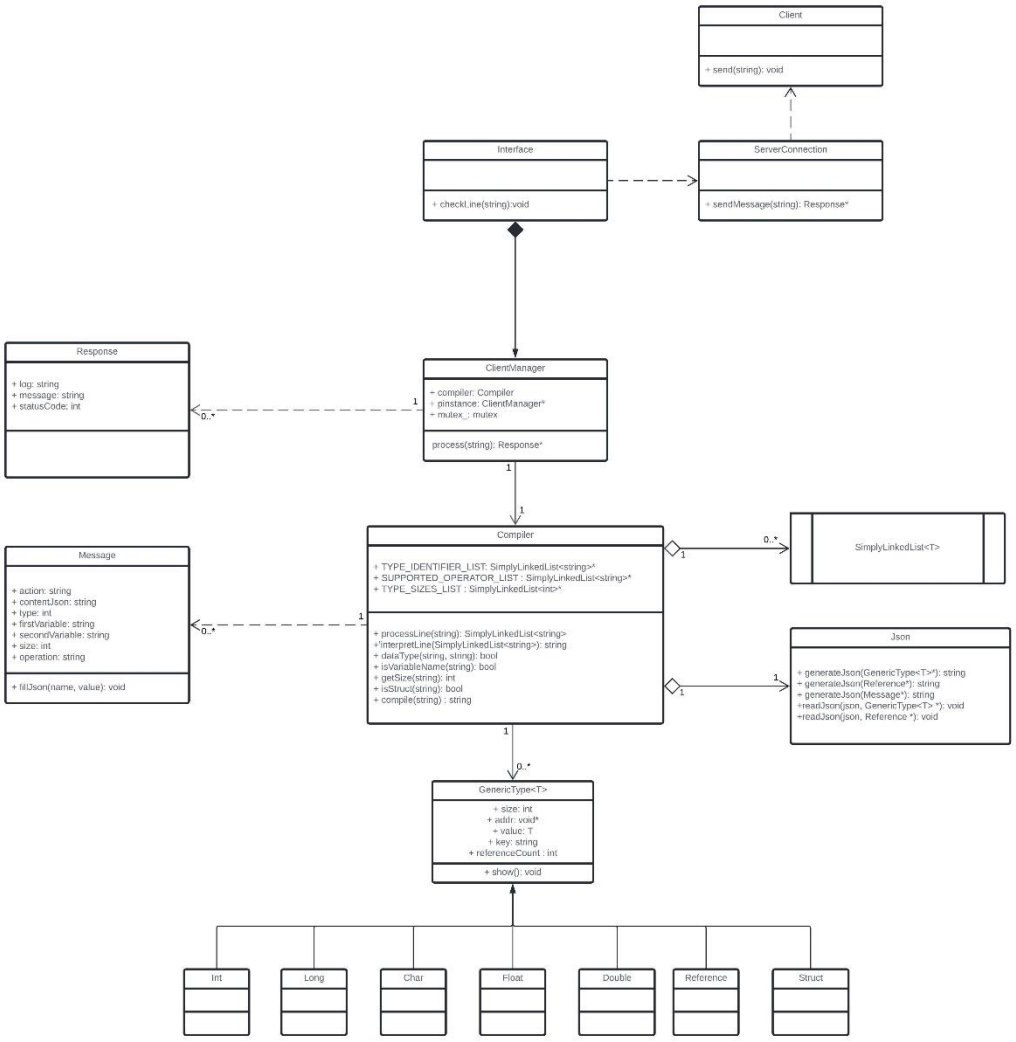
Para verlo con mas detalle puede ir al siguiente enlace con formato .jpg:
https://cdn.discordapp.com/attachments/772625100896862209/837202849456062514/C_Editor.jpeg

Diagrama de clases en formato JPEG o PNG

Servidor:



Cliente:



Descripción de las estructuras de datos desarrolladas

Lista simple:

Una lista simple es una estructura de datos lineal que puede ser recorrida en una sola dirección, además esta lista posee la característica de tener un inicio y un final bien definidos, tomando como inicio el primer nodo que en ella se registre y como final aquel nodo cuyo atributo siguiente sea un estado nulo (NULL). Se utilizó una lista simple para el desarrollo de las fases A y B del tablero, además de otros usos

menores que se le dio en sitios puntuales donde una lista simple podía provocar un mejor rendimiento y mayor comodidad para el manejo de datos.

Pila:

Esta es una estructura de datos que se basa en la implementación de una lista simple y la agregación de diferentes funciones únicamente propios de las pilas, como los son los métodos push, pop y peek. Las pilas trabajan siguiendo una metodología de tipo "LI-FO" (Last In - First Out) que permite un acceso rápido y ordenado al último elemento que se insertó en la lista. Esta estructura se utilizó en la implementación del almacenamiento de los eventos, ya que se requería una estructura que permitiera obtener los elementos de una lista de un modo ágil y rápido.

Cola:

Esta estructura funciona como una fila, se insertan elementos en un extremo de esta, y para la extracción se toma el elemento que fue insertado primero, este ahora estará en el otro extremo de la fila (F.I.F.O). Utilizada para recorrer los árboles por niveles.

Descripción detallada de los algoritmos desarrollados

El algoritmo utilizado para procesar las líneas de código ingresadas por el cliente se ejecuta un búcle while, en el cual se realiza una evaluación de los espacios en blanco y la aparición de puntos y coma, todo este proceso se lleva a cabo para descomponer la linea en una lista de palabras palabras que otro método podrá analizar y determinar el orden en que se deberán encontrar las palabras.

```
static SimplyLinkedList<string> processLine(string line) {
    bool flag = false;
    auto *result = new SimplyLinkedList<string>();
    int counter = 0;
    char character;
    string word;

    while (counter < line.length()) {
        character = line[counter];
        if (isblank(character)) {
            if (!word.empty()) {
                result->append(word);
                word.clear();
            }
        } else if (character == ';') {
            if (!word.empty()) {
                result->append(word);
                word.clear();
            }
            word.push_back(character);
            result->append(word);
            word.clear();
            flag = true;
            break;
        } else if (character == '=') {
            if (!word.empty()) {
                result->append(word);
                word.clear();
            }
            word.push_back(character);
            result->append(word);
            word.clear();
        } else if (character == '<') {
            //verificar que la palabra que ya está es "Reference"
            //recorrer el string hasta encontrar el >
            //sino, tirar error
            //asignar el valor en la lista y seguir normalmente
            // buscar el nombre, normal.
            // luego del nombre tiene que seguir un ____.getAddr()

        } else {
            word.push_back(character);
        }
```

Problemas encontrados en forma de bugs

Durante la implementación del proyecto no se han podido implementar varios aspectos como lo son el *reference(), el struct() y el scope, para este caso se han creado las clases y se intentó integrarlas así como sus métodos en el workflow del trabajo, sin embargo, su funcionamiento no es el que se esperaba y en ocasiones no servían del todo.

Patrones de diseño utilizados:

Singleton

En ingeniería de software, singleton o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. En este proyecto utilizamos el patrón de diseño singleton, con el cual se hacen únicas instancias de cada objecto o clase, de esta manera podemos acceder desde cualquier otra clase a los valores de la instancia de la clase a la cual se le aplicó este patrón de diseño. Esto ayudó masivamente a aplicar cambios en los estados de la clase Client, Server, Client Manager, etc.

Facade

Fachada (Facade) es un tipo de patrón de diseño estructural. Viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos. Este se aplicó en la clase de ServerManager(), el cual maneja todos los procesos internos del funcionamiento del servidor detrás de procesos fáciles de entender dentro de su clase.