Instituto Tecnológico de Costa Rica,

ITCR, Cartago, Costa Rica

Ingeniería en Computadores

Algoritmos y Estructuras de datos 2

Proyecto Programado 1

Profesor:

Isaac Ramirez

Estudiantes:

Gunther Karolyi Gutierrez

2017238873

Eduardo Quiroga Alfaro

2017079534

Grupo 1

18/04/18

Índice

Introducción

El siguiente proyecto fue realizado por los estudiantes: Eduardo Quiroga Alfaro y Gunther Karolyi Gutiérrez, bajo la supervisión del profesor Isaac Ramírez, en el Instituto Tecnológico de Costa Rica. La entrega de los resultados es el día 18 de abril del año 2018.

El proyecto consiste en una serie de partes que van a constituir una aplicación para la administración y gestión de memoria junto con un idle que compile el pseudo código C!. La idea es generar un ambiente de programación en el cual se pueda visualizar de manera real el uso de la memoria RAM.

Los componentes de la aplicación son:

- **C! Idle**: Interfaz de usuario que contiene un editor de texto capaz de reconocer patrones en la sintaxis del código C!, debe compilarlo y mostrar el administrador de memoria.

- **Mserver** : El servidor es el responsable de administrar la memoria RAM, así mismo, debe solicitar la memoria y estar recibiendo las peticiones del C! Idle.

Descripción del Problema

Se desea diseñar e implementar un ambiente de programación para el pseudo lenguaje C!, junto con un manejador de memoria. El ambiente debe presentar los siguientes features:

- **Editor de C!**, el cual debe permitir el ingreso del código. Así mismo, el idle interpreta y ejecuta el código al presionar el botón **Run.**
- **Stdout**, sección bajo el editor donde se imprime las llamadas a la función **print** o método equivalente.
- **Application Log**, se debe mostrar el log del idle , es decir, mostrar los errores internos, llamadas al server y todo lo que ocurre internamente.
- **RAM Live View**, se muestra en tiempo real la memoria RAM conforme se ejecutan las líneas de código de C!. Se debe mostrar la dirección de memoria, valor, etiqueta y conteo de referencias.
- **Mserver**, solicita la memoria RAM, permite administrar la memoria y debe recibir solicitudes del Idle.

Planificación y administración del proyecto

Features e historias de usuario

El proyecto se divide en dos grandes secciones las cuales se tienen una serie de features específicos que se mostraran a continuación:

- C! idle:
    - El IDE interpreta y ejecuta el código C! cuando se presiona el botón Run. La ejecución se hace de manera detenida al estilo de depuración.
    - En caso de que el IDE detecte un error de sintaxis o semántico, la ejecución del programa se detiene y se muestra el error en stdout.
    - Stdout: Cumple la función de standard output. Si el código C! contiene llamadas a ***print ()*** el resultado se imprime en esta sección
    - Application Log: Se debe mostrar cualquier error interno, cualquier llamada al servidor de memoria, todo lo que ocurre internamente, se mostrará en esta sección.
    - RAM Live View: En esta sección se ve en tiempo real, el estado del RAM conforme cada línea de C! se ejecuta. Este vista consulta

constantemente al servidor de RAM para obtener el estado de la memoria y lo pinta en pantalla.

- Mserver:
    - El servidor realiza un único malloc de la memoria total. El servidor tiene un mapa interno del bloque entero de memoria. Con cada petición recibida de C!, el servidor maneja offsets para determinar la posición de cada variable de C! dentro del bloque de memoria real.
    - El servidor escucha peticiones de C! IDE enviados en formato JSON. Cuando C! IDE envía una petición debe indicar el tipo de datos, el nombre de la variable y el tamaño que se deberá reservar.
    - El servidor maneja la memoria automáticamente.
    - El servidor lleva el conteo de referencias y cada cierto tiempo ejecuta el garbage collector que elimina los espacios de memoria que no son referenciados.

Las historias de usuario son las siguientes:

C! Idle:

- Como cliente, quiero que el ambiente tenga un interfaz amigable.
- Como cliente, quiero tener fácil acceso a la información de la memoria RAM.
- Como cliente, quiero poder visualizar los errores de forma clara.
- Como cliente, quiero un editor de texto que muestre la linea que estoy trabajando.
- Como cliente, quiero que el editor reconozca palabras claves para visualizar facilmente.

Msever:

- Como desarrollador, quiero una conección rápida con el Idle.
- Como desarrollador, quiero facilidad al cambiar el tamaño de RAM que se solicita.
- Como desarrollador, quiero facilidad de uso de los offsets.
- Como desarrollador, quiero autonomía del server y se detenga cuando se solicite.

Minimal System Span

Para la aplicación, los que requerimientos mínimos del dispositivo que se va a utilizar son los siguientes:

- Ubuntu 16.04 (recomendado)
- 1GB de RAM (recomendado)

Asignación de user stories

Estudiante	Eduardo Quiroga	Gunther Karolyi
User Stories	● Como desarrollador, quiero una conección rápida con el Idle. ● Como desarrollador, quiero facilidad al cambiar el tamaño de RAM que se solicita. ● Como desarrollador, quiero facilidad de uso de los offsets. ● Como desarrollador, quiero autonomía del server y se detenga cuando se solicite. ● Como cliente, quiero tener fácil acceso a la información de la memoria RAM.	● Como cliente, quiero que el ambiente tenga un interfaz amigable. ● Como cliente, quiero poder visualizar los errores de forma clara. ● Como cliente, quiero un editor de texto que muestre la linea que estoy trabajando. ● Como cliente, quiero que el editor reconozca palabras claves para visualizar facilmente.

Descomposición de user stories en tareas

Estudiante	Eduardo Quiroga	Gunther Karolyi
Tareas	● Implementar el servidor ● Implementar el parseo del código ● Convertir el código en objetos JSON ● Implementar el cliente del servidor para enviar los objetos JSON	● Crear una base para la interfaz gráfica ● Implementar los detalles en el área de escritura de código ● Implementar la visualización de errores

Bitácora

Descripción de la actividad	**Tiempo invertido**	**Fecha**
Investigación	6 horas	7/4/2018
Desarrollo del server	5 horas	7/4/2018 - 8/4/2018
Diseño del Idle	7 horas	11/4/2018 - 17/4/2018 - 20/4/2018
Documentación	2.5 horas	15/4/2018 - 17/4/2018
Manejo de datos Json	1 hora	8/4/2018
Desarrollo del compilador	5 horas	8/4/2018 - 10/4/2018
Tiempo Total	26.5 horas	7/4/2018 - 17/4/2018

Diseño

Diagrama de componentes

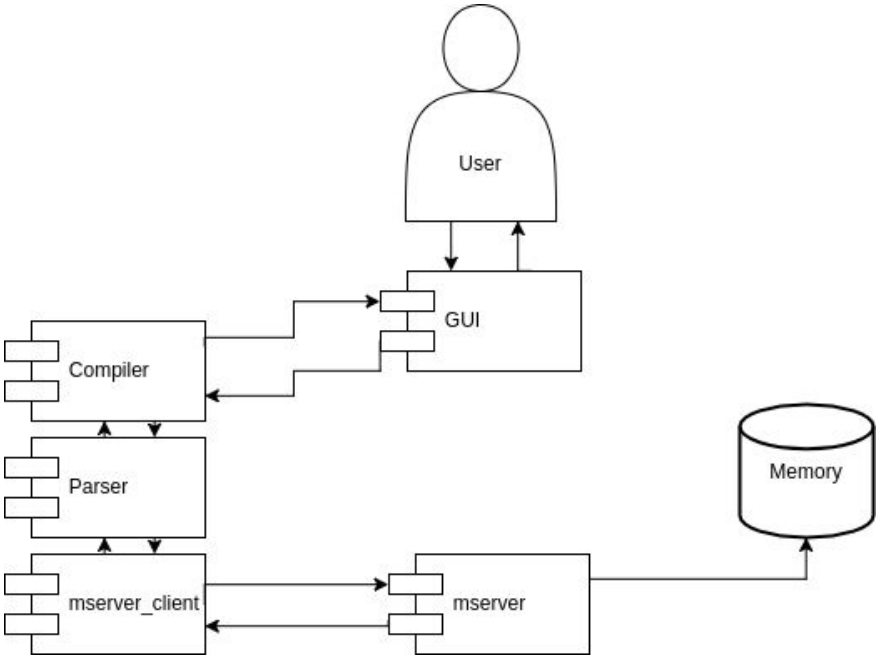


Diagrama de despliegue

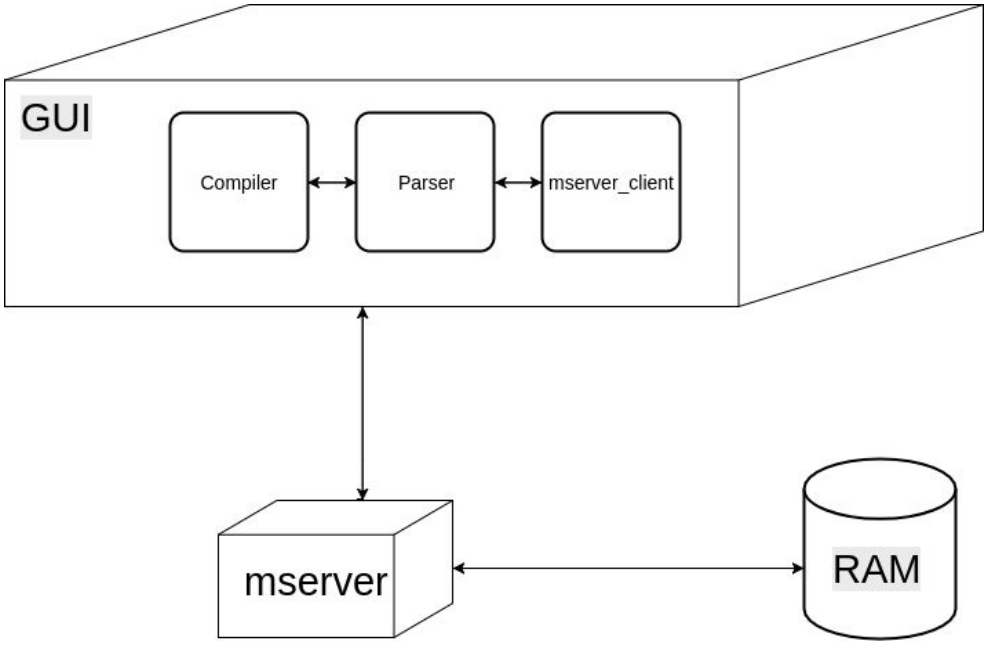


Diagrama de clases

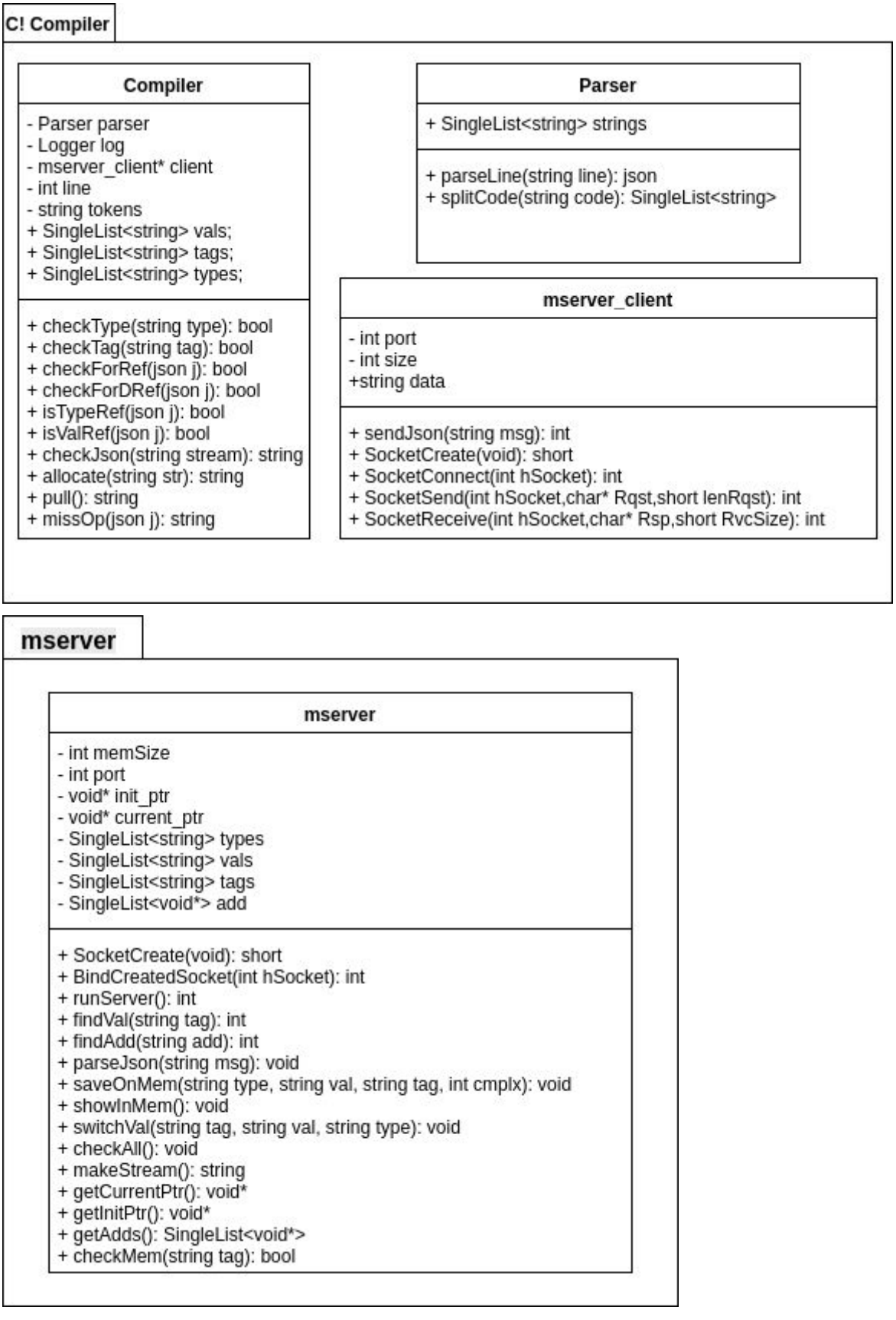
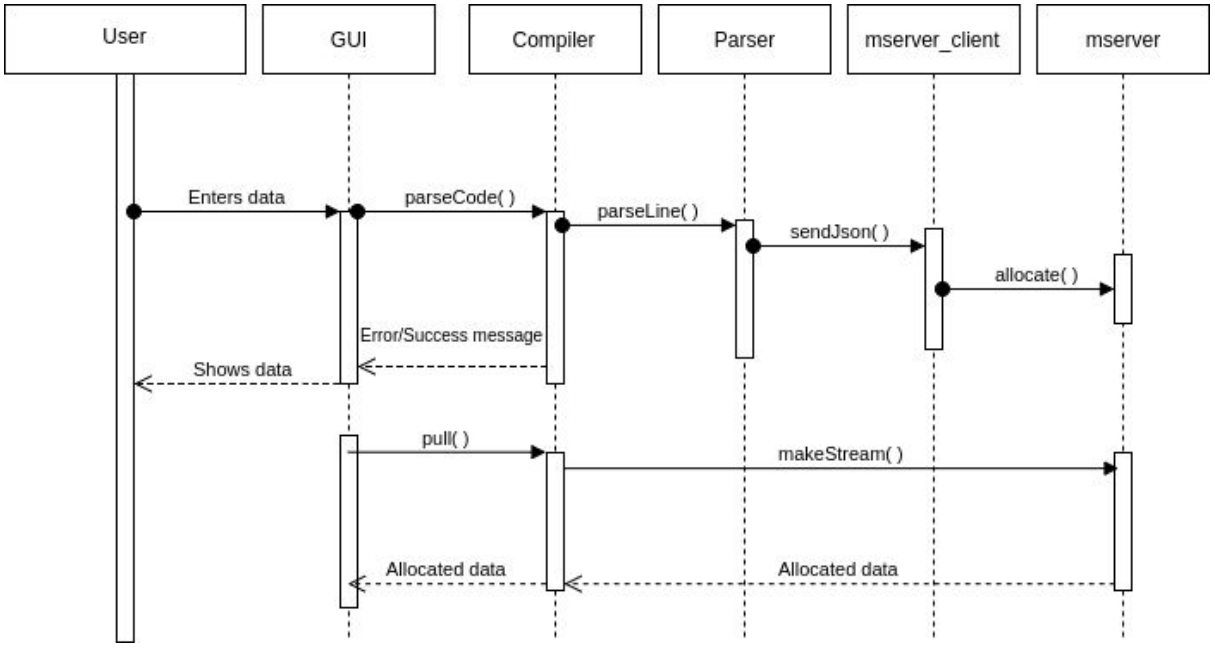


Diagrama de secuencia



*Se hizo énfasis en las historias de usuario que involucran el almacenamiento en memoria y la visualización de mensajes de error y datos claves de la memoria utilizada.

Implementación

Descripción de bibliotecas utilizadas

Para el manejo de JSON se utilizó la biblioteca “Nlohmann JSON for Modern C++” cuyo link puede ser encontrado en las bibliografías. Se escogió ésta ya que se facilita el método de declaración y creación de objetos JSON, y el parseo y acceso de los datos almacenados en los mismos.

Las estructuras de datos fueron implementadas desde una biblioteca de estructuras básicas programada por el estudiante Eduardo Quiroga, en la primera tarea programada. Se modificó la lista simple sobrecargando un operador para simplificar el acceso a los datos de las listas.

Descripción de algoritmos desarrollados

A continuación se va a dar una breve explicación de los algoritmos más relevantes del proyecto. Algoritmos simples como búsqueda de datos serán omitidos ya que su funcionamiento es básico. También serán omitidos getters y setters.

- Mserver:
    - runServer( ): Inicia el funcionamiento del servidor abriendo un puerto para la entrada de datos. Hay dos palabras reservadas que le indican al servidor que debe ejecutar una acción determinada. Si el stream entrante es “break”, el

servidor detiene su funcionamiento. Si el stream entrante es “pull” el servidor crea un stream de datos que contiene todos los datos guardados en memoria. Si el stream no es ninguno de estos dos anteriores, se asume que es un objeto JSON y procede a parsearse y entrar en memoria.
    - allocate( ): Se encarga de almacenar los datos en memoria. Encuentra el tipo de dato que está por almacenarse y procede a almacenarlo como corresponde.

- Mserver_client:
    - sendJson( ): Esta función es la encargada de enviar los datos por medio del socket a mserver. Al momento de que los datos entran como un string, se asume que son correctos y han sido depurados y validados. El primer paso del proceso es conectar con el servidor, seguido del envío de datos y por último se recibe una respuesta del servidor para desconectarse.

- Parser:
    - splitCode( ): Esta función se encarga de separar el código en líneas individuales colocando en una lista linea por linea.
    - parseLine( ): Se encarga de separar una línea en tokens individuales.

- Compiler:
    - allocate( ): Se encarga de leer el código e implementar todas las clases anteriores para separar el código en líneas, estas separarlas en tokens y convertirlas en un objeto JSON. Este objeto JSON es enviado mediante el mserver_client para ser ubicado en memoria por el servidor mserver.
    - pull( ): Recibe un stream de datos en forma de string para utilizarlos en la GUI. Este stream contiene todos los datos que son enviados por el servidor, que están ubicados en memoria.

Problemas encontrados

- Uno de los problemas que se presentó durante el desarrollo de la interfaz gráfica fue la realización de un botón para ejecutar el código. Esto debido a que el programa QT únicamente permite ubicar objetos que hereden la clase QWidget en los layouts. De esta forma se buscaron componente que tuvieran la capacidad de agrupar varios push button sin embargo no se logra ubicar ninguno. Posteriormente se intentó utilizando mediante el manejo de eventos, pero al utilizar el método connect(), no se logró hacer una conexión exitosa.
- El parseo de Structs no se logró completar debido a que al estar en varias líneas en vez de una sola, crea complicaciones a la hora de la división en tokens, y en el almacenamiento de memoria

Conclusión

El principal objetivo de este proyecto fue el entendimiento y la aplicación de los alojamientos en memoria, el uso y manejo de punteros, y el desarrollo de una aplicación involucrando programación orientada a objetos en C++.

Se puede destacar que el manejo de memoria fue logrado exitosamente. El servidor funciona por completo ya que no es el encargado de valorar las entradas. La parte de la aplicación, interfaz y demás fue la que se complicó a la hora de la integración con el backbone para que todo funcione en conjunto.

Una recomendación que se puede destacar con base en esto es que cuando se vaya a implementar una solución de dos o más partes, se tomen en consideración las demás partes cuando se esté desarrollando una. De esta manera se puede reducir el tiempo de integración ya que las partes van a estar diseñadas con el objetivo de integrarse juntas.

Bibliografía

The QT Company. (n.d.). Qt Documentation. Extraído el día Abril 7, 2018, de http://doc.qt.io

C Board. (n.d.). Extraído el día Abril 7, 2018, de https://cboard.cprogramming.com (C++ Forum)

Nlohmann JSON for Modern C++. (n.d.). Extraído de https://github.com/nlohmann/json el día Abril 7, 2018.

Socket programming in c using TCP/IP - AticleWorld. (2017, Mayo 14). Extraído de https://aticleworld.com/socket-programming-in-c-using-tcpip/

Anexos

Interfaz grafica: https://github.com/karolyi15/C-idle-.git
Backbone del compilador: https://github.com/ed-quiroga-2103/D2P1_CCompiler
Servidor de memoria: https://github.com/ed-quiroga-2103/mserver_ccomp
Solución integrada: https://github.com/ed-quiroga-2103/Datos2Proyecto1