Instituto Tecnológico de Costa Rica

Área académica de ingeniería en computadores

Proyecto 1

IDE C!

Profesor

Antonio Gonzales Torres

Estudiante

Luis David Richmond Soto

Carné

2020077226

Semestre 1, 2021

Tabla de contenido:

Descripción:

El proyecto consiste en el diseño e implementación de un IDE que permite interpretar y ejecutar el pseudo lenguaje C! con manejo de memoria, mediante las diferentes secciones que este posee se puede editar texto, imprimir los datos especificados por medio de la llamada a cout, además permite tener un control detallado de los errores internos producidos durante la ejecución así también se puede observar el estado de la RAM en cuanto a la dirección de memoria, valor, etiqueta y conteo de referencias que posee, conforme cada línea de C! se ejecuta.

Especificaciones de usuario:

Consiste en un editor de texto el cual permite ingresar código de C! el cual tiene una sintaxis derivada del lenguaje de programación C.

Se desarrollo usando Qt para realizar la interfaz además sockets propios de C++ además de usar json para la comunicación entre los sockets el cual se usó la biblioteca Json for modern desarrollada por Nlohmaan.

Para descargar e instalar Qt y Qt creator, se descarga el archivo, lo descomprimimos y ejecutamos para instalarlo. Link de descarga https://www.qt.io/download-open-source?hsCtaTracking=9f6a2170-a938-42df-a8e2-a9f0b1d6cdce%7C6cb0de4f-9bb5-4778-ab02-bfb62735f3e5.

Para el json basta con descargar el el archivo json.hpp de https://github.com/nlohmann/json/releases y adjuntarlo a los archivos de nuestro proyecto.

Con respecto a los tipos de datos y su sintaxis que admite C! son:

a) int: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final. Ejemplo:

int a = 5 + 6;

b) char: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final, además de un solo único elemento y debe ir entre comillas. Ejemplo:

char b = '1';

c) double: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final, además de concordar con un double el cual debe de tener su parte decimal. Ejemplo:

double c = 2.1;

d) float: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final, demás de concordar con un double el cual debe de tener su parte decimal pero sin la necesitas de la f al final. Ejemplo:

float d = 31.4;

e) long: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final. Ejemplo:

long e = 5311314;

f) reference: Debe tener un espacio entre cada elemento de la línea de código y punto y coma al final, además de indicar el tipo de dato (int,char,float,double,long) y antes de la variable a la cual se referencia debe de ir un &. Ejemplo:

reference <int> f = & a;

g) struct: Debe tener un espacio entre cada elemento de la lista, luego de "struct" debe de ir un corchete abierto. En las siguientes líneas de código debe de ir las variable que pertenezcan al struct con su respectiva sintaxis. Para finalizar el struct se cerrará el corchete y seguido de un espacios pondremos los elementos que crearemos con respecto al struct. Si queremos hacer uso de algun dato pondremos "nombre del elemento" + "variable". Ejemplo:

```
struct {

int e1 = 5;

int e2 = 6;

} obj1 obj2
```

h) cout: Debe tener un espacio entre cada elemento de la lista, además de lo que gustemos imprimir en la consola. Si queremos imprimir alguna variable solo agregariamos el nombre de dicha variable, si lo que queremos es imprimir algun texto o número, va entre comillas simple. Ejemplo: cout << '5';

i) Bloques de código: están definidos dentro de "{ }", donde las variables dentro de dicho bloque, solo exestirán dentro de ese bloque de código. Ejemplo:

```
{

int a = 5;

int b = 7;

}
```

Con ejemplo de código sería el siguiente:

```
int a = 5;

int b = 6 + a;

double c = 62.13;

float d = 31.31;

char e = '1';

reference <int> f = & b;

struct {

int dato1 = 10;

int dato2 = 20;

} obj1 obj2
```

cout << a;

Reserva de memoria:

En la parte inferior izquierda encontremos un área donde podemos reservar la memoria que gustemos en MB, es decir si queremos reservar 10MB, indicamos la cantidad de 10 y pulsamos el botón reservar.

Ram View:

Se actualizará después de la asignación de cada elemento, para el caso del struct será después de la línea de código donde termina el struct. Se actuliza también cuando cerramos un bloque de código y por último al hacer uso del garbage collector.

Log:

Cada acción que se realizará aparecerá dentro de dicha área, además de los errores.

Botones:

Run y Next: Para iniciar la ejecucción del código primero daremos click al botón run y para avanzar entre las línas usaremos el botón run. Reset: Reinicia la ejecucción y además borra el RamView Stop: Reinicia la ejeccucicón pero no borra el RamView

Algoritmo Desarrollados:

Split

Por medio de esta función se realiza la separación del código en líneas individuales para así lograr generar un vector de string con el cual se realizara el parse de los datos ingresado.

```
void StringParse::SplitString(const string& texto, char del, vector<string>& v){

    string::size_type i = 0;
    string::size_type j = texto.find(del);

    while (j != string::npos){
        v.push_back(texto.substr(i,j-i));
        i = ++j;
        j = texto.find(del,j);

        if (j == string::npos){
            v.push_back(texto.substr(i,texto.length()));
        }
    }

    if (v.size() == 0){
        v.push_back(texto.substr(0,1));
    }
}
```

Generar Struct

Método encargado de generar struct para poder reservarlo en memoria.

```
string line;
vector<string> aux;
json obj;

for (int i = 1; i < datas.size(); i++) {

    for (int j = 0; j < linesStruct.size(); j++) {

        string lineAux = EliminarEspacios(linesStruct[j]);

        SplitString(lineAux,'#',aux);

        obj["type"] = aux[0];
        obj["name"] = datas[i]+aux[1];
        obj["value"] = aux[3];

        cliente->StartClient(obj.dump(4));

        sleep(0.5);

        obj.clear();

        aux.clear();

    }
```

```
    aux.clear();
}

obj["type"] = "values";
obj["name"] = "notValue";
obj["value"] = "notValue";

return obj.dump(4);
```

Operaciones

Algortimo que realiza la resolución de operaciones basado en la ubicación de carecteres especiales (+,-,*,/) en cada linea de código para poder resolver estas operaciones entre dos valores ingresados asi como cuando se indica una variable anteriormente ingresada por el usuario.

```
  if(line.at(i) == "+"){

      valor1 = ObtenerNumero(line.at(i+1));

      if (valor1 != "error"){
          valor += stoi(valor1);
      } else{
          return "error";
      }
```

Garbage collector

Métodos para generar el garbage collector, que elimine los datos y reacomode la memoria.

```
void MemoryBlock::RefactorCollector() {
    iCollector = size-1;
    size-=garbage.size();
    for (int i = 0; i < garbage.size(); i++) {
        if(RefactorCollectorAux()){
```

```
            if(NotVerificar()){
                mem[garbage.at(i)] = mem[iCollector];
                iCollector--;
            }
        }
    }
    garbage.clear();
    notGarbage.clear();
}
/// Retorna el valor la memoria a cambiar
bool MemoryBlock::RefactorCollectorAux() {
    for (int i = garbage.size()-1; i != -1; i--) {

        if(Verificar()) {
            return true;
        } else{
            iCollector--;
        }
    }
    return false;
}
/// Verifica si esta marcado
bool MemoryBlock::Verificar() {
    int validar = 0;
    for (int j = garbage.size()-1; j != -1 ; j--) {
        if(garbage.at(j) == iCollector){
            validar++;
        }
```

```
    }
    if(validar==0){
        return true;
    }else{
        return false;
    }
}
/// Verifica si esta en los no marcados
bool MemoryBlock::NotVerificar() {
    int validar = 0;
    for (int i = 0;i < notGarbage.size();i++){
        if (notGarbage.at(i) == iCollector){
            validar++;
        }
    }

    if(validar != 0){
        return true;
    } else{
        return false;
    }
}
```

Estructuras Desarrolladas:

Facade

El patrón Facade tiene la característica de ocultar la complejidad de interactuar con un conjunto de subsistemas proporcionando una interface de alto nivel, la cual se encarga de realizar la comunicación con todos los subsistemas necesarios, lo cual permitió realizar la conexión entre el servidor y el cliente para realizar las funciones asignadas para el manejo de los datos recibidos y enviados entre ambos.

Vector

El tipo de variable vector es una secuencia de objetos del mismo tipo almacenados consecutivamente en memoria que permite almacenar varios valores bajo una sola variable y realizar operaciones complejas con esta, esta estructura se utlizó para el manejo de los datos de dirección de memoria, valor, etiqueta y conteo de las referencias que se obtienen de las variables definidas por el usuario en el editor de C!.

Node

Tenemos un objeto node el cual almacena los datos de las variables que el usuario desee almacenar, conteniendo el dato, el nombre y la cantidad de referencias que se estén haciendo durante la ejecucción.

Condicionales

Una estructura condicional permite decidir por cuál alternativa seguirá el flujo del programa dependiendo del resultado de la evaluación de una condición, este tipo de estructura se utilizó para verificar el tipo de dato ingresado para lograr clasificar los datos de acuerdo a este.

Bugs Encontrados:

Falta de ; en el struct:

A la hora de generar un struct, el programa no verifica si al final de struct hay un ";", puede recorrer sin necesitad de escribirlo, no detecta el error.

Problemas con sockets:

Con el tema de los sockets, puede que el socket de cliente no cierre o inicie correctamente, entonces a la hora de mandar un dato o una instrucción el programa puede estar comprendido, además de no estar recibiendo la información por parte del servidor. También si se envían los datos rápidamente puede haber un problema donde se combine la información o puede que se caiga la conexión.

Operaciones:

Al realizar las operaciones entre floats y doubles puede haber pérdida de datos, esto por un error de pasar el string a double o float. Esto se debe un problema de la biblioteca de Qt, al cual fue empleada para realizar la interfaz.

Ram View:

Al resetear la ram view, se borran los datos pero no las filas, además desaparecen los labels, de "dirección, valor, etiqueta y referencias".

Diagrama de clases: