UNIVERSIDAD SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA

Organización Lenguajes y
Compiladores 1 Escuela de
Ciencias y Sistemas
Segundo Semestre 2018
Ing. Mario Bautista, Manuel Castillo,
César Batz Tutores Académicos
Luis Paz, José Soc, Jorge Castañeda

PROYECTO 2: CHATBOT COMPILER

Contenido

1. Objetivos 4
   1.1. Generales 4
   1.2. Específicos 4
2. Descripción 4
3. Componentes de la solución. 4
   3.1. Entorno Integrado de Desarrollo (IDE) 4
   3.2. Integrador 5
   3.3. Proveedor (Chat) 5
4. Arquitectura de la solución 5
5. Descripción del lenguaje cbc 5
   5.1. Notación utilizada 5
   5.2. Sensibilidad a mayúsculas 6
   5.3. Sobrecarga de métodos 7
   5.4. Recursividad 7
   5.5. Identificadores 7
   5.6. Variables 7
   5.7. Tipos de datos 7
6. Sintaxis 8
   6.1. Encabezado 8
   6.2. Bloque de sentencias 8
   6.3. Signos de agrupación 9
   6.4. Comentarios 9
   6.5. Operaciones aritméticas 9
   6.6. Operaciones relacionales 10
   6.7. Operaciones lógicas 11
   6.8. Precedencia y asociatividad de operadores 11
   6.9. Declaración de variables 11
   6.10. Asignación de variables 12
   6.11. Arreglos de una dimensión – declaración y asignación 12
   6.12. Acceso a posiciones de los arreglos 13
   6.13. Funciones/métodos 14
   6.14. Función Main 15

6.15. Llamadas a funciones/métodos 16
6.16. Sentencia Return 17
6.17. Sentencia If 17
6.18. Sentencia Switch 18
6.19. Sentencia For 20
6.20. Sentencias While 20
6.21. Sentencia Do While 21
6.22. Sentencia Break 21
7. Funciones nativas 22
7.1. Sentencia Print 22
7.2. CompareTo 22
7.3. GetUser() 23
8. Almacenamiento de usuarios 23
9. Mensajes y respuestas en la solución 23
10. Aplicación móvil/Web (Chat) 25
10.1. Interfaz gráfica sugerida 26
11. Funcionamiento de la aplicación 27
12. Requerimientos mínimos 29
13. Reportes 29
14. Documentación 29
15. Restricciones 30
16. Fecha de entrega 30

1. Objetivos

1.1. Generales

- Que el estudiante aplique los conocimientos aprendidos en el curso de compiladores 1 para poder solucionar problemas de diversos lenguajes de programación.

1.2. Específicos

- Que el estudiante genere e interprete un árbol de análisis sintáctico
- Que el estudiante comprenda la ejecución de las diferentes sentencias de control de un lenguaje de programación
- Que el estudiante maneje una tabla de símbolos para realizar acciones de un lenguaje determinado.
- Que el estudiante aprenda a manejar variables en el ámbito global y ámbitos y sub ámbitos locales.
- Que el estudiante consolide una base de conocimientos previa al curso de Lenguajes y Compiladores 1.
- Que el estudiante sea capaz de implementar acciones en un compilador aplicándola a temas útiles en la vida real.

2. Descripción

La atención al cliente es muy importante debido a que establece una conexión entre una empresa y sus clientes o futuros clientes, pero desgraciadamente no es suficiente con el personal que una empresa posee para realizar dicha labor por lo que se le solicita a usted para que le dé solución al problema por medio de un ChatBot. Un ChatBot es una aplicación informática basada en inteligencia artificial que permite simular la conversación con una persona, dándole respuestas automatizadas a sus dudas o preguntas más comunes. Debido a que es la primera versión de la solución solamente podrá responder a preguntas cerradas. La manera de darle solución al problema es que usted utilice sus conocimientos de compiladores 1 creando un lenguaje de programación y su respectivo compilador para crear las instrucciones del integrador y así establecer una comunicación con los clientes de manera automatizada.

3. Componentes de la solución.

3.1. Entorno Integrado de Desarrollo (IDE)

El IDE es una aplicación que proporciona todas las herramientas necesarias para facilitarle al programador del desarrollo del software y contendrá los siguientes elementos:

- **Nuevo:** creará una nueva pestaña en blando en el editor.
- **Abrir:** abrirá un cuadro de dialogo para seleccionar un archivo con extensión .cbc y mostrara su contenido en una nueva pestaña.
- **Guardar:** Guardará el contenido de la pestaña actual en un archivo, si la pestaña es una pestaña nueva, se deberá mostrar un cuadro de dialogo para seleccionar la ubicación y nombre del archivo.
- **Guardar como:** esta opción mostrará directamente un cuadro de diálogo para seleccionar la ubicación y el nombre del archivo a guardar.

- **Cerrar pestaña:** esta opción eliminara la pestaña de la interfaz.
- **Ejecutar archivo:** ejecutará el contenido de la pestaña actual.
- **Ver reporte de errores**: abrirá el documento pdf donde se encuentran los errores.
- **Consola salida:** en esta área se visualizarán todas las expresiones o cadenas que se envíen a imprimir desde el programa.

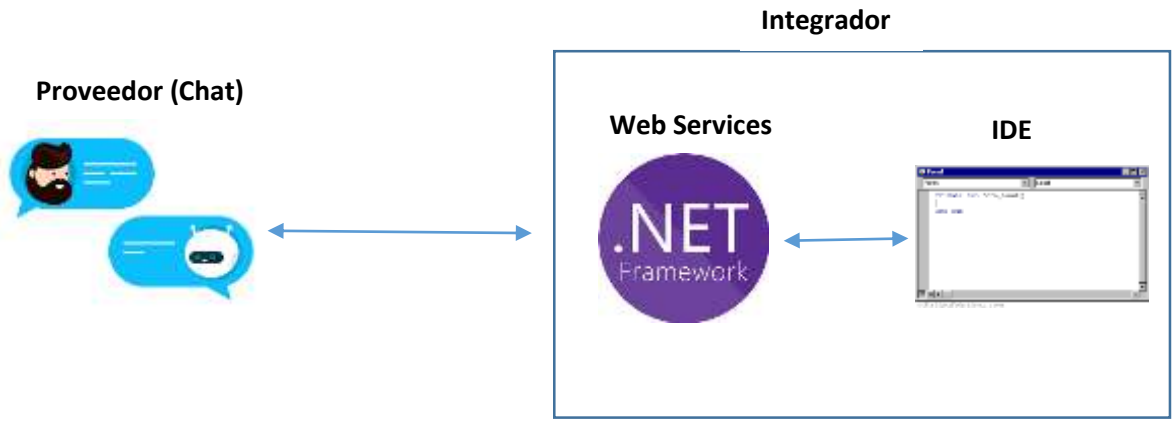
3.2. Integrador

Es la pieza clave de un chatbot, es la parte de inteligencia artificial de un chatbot, por ser la primera versión de la solución este componente solamente contara con funcionalidades básicas que se describen más adelante. Este elemento de la solución utilizara el lenguaje cbc pare procesar las preguntas y retornar una repuesta.

3.3. Proveedor (Chat)

Es la interface mediante el cual el usuario habla con el bot, es decir es el componente que le servirá al usuario para hacer las preguntas.

4. Arquitectura de la solución



5. Descripción del lenguaje cbc

El lenguaje cbc es como cualquier otro lenguaje de programación, tiene su propia estructura, reglas de sintaxis y paradigma de programación, es un derivado del lenguaje TypeScript, por lo que sus reglas de sintaxis son parecidas a dicho lenguaje.

5.1. Notación utilizada

Se utilizarán las siguientes notaciones cuando se haga referencia a sentencias del lenguaje cbc

Sintaxis y ejemplos

Para definir la sintaxis y ejemplos de sentencias de código se utilizará un rectángulo celeste.

Asignación de colores

Dentro del enunciado, para el código del lenguaje XXX, se seguirá el formato de colores establecidos en la tabla 1.

Tabla 1: Colores del lenguaje XXX	
Color	Token
Azul	Palabras reservadas
Naranja	Cadenas, caracteres
Morado	Números
Gris	Comentarios
Negro	Otros

Palabras reservadas

Son palabras que no podrán ser utilizadas como identificadores ya que representan una instrucción específica y necesaria dentro del lenguaje.

Expresiones

Cuando se haga referencia a una 'expresión', se hará referencia a cualquier sentencia que devuelva un valor.

Cerradura positiva

Esta será usada cuando se desee definir que un elemento del lenguaje cbc podrá venir una o muchas veces. (elemento)+

Cerradura de Kleene

Esta será usada cuando se desee definir que un elemento del lenguaje cbc podrá venir cero o muchas veces. (elemento)*

Opcionalidad

Esta será usada cuando se desee definir que un elemento del lenguaje cbc podrá o no venir) cero o una vez). (elemento)?

5.2. Sensibilidad a mayúsculas

cbc será un lenguaje case sensitive, es decir, dicho lenguaje si será sensible a las mayúsculas y minúsculas, esto aplicará tanto para palabras reservadas propias del lenguaje como para identificadores.

5.3. Sobrecarga de métodos

Será posible definir dos o más métodos que compartan el mismo nombre, mientras que las declaraciones de sus parámetros sean diferentes.

Formas de diferenciar las declaraciones de parámetros:

- ✓ Cantidad de parámetros
- ✓ Tipo de parámetros

5.4. Recursividad

Los métodos y funciones de cbc deberán soportar llamadas recursivas.

5.5. Identificadores

Un identificador será utilizado para dar un nombre a variables, métodos o estructuras. Un identificador es una secuencia de caracteres alfabéticos **[A-Z a-z]** incluyendo el guion bajo [_] o dígitos **[0-9]** que comienzan con un carácter alfabético o guion bajo.

Ejemplos de identificadores

```
Identificador_valido_1  //válido

2iden_1 //no válido

Y_otro_mas_10 //válido
```

5.6. Variables

Las variables serán unidades básicas de almacenamiento en cbc. Una variable se definirá por la combinación de un identificador, un tipo y un inicializador opcional. Además, la variable tiene un entorno que define su usabilidad.

5.7. Tipos de datos

- **Int**: este tipo de dato solamente aceptará valores numéricos enteros.
- **Double**: este tipo de dato aceptará números de coma flotante de doble precisión
- **String**: este tipo de dato aceptará cadenas de caracteres, que deberán ser encerradas dentro de comillas dobles.
- **Char**: este tipo de dato aceptará un único carácter, que deberá se encerrado en comillas simples.
- **Bool**: este tipo de dato aceptará valores de verdadero y falso que serán representados con palabras reservadas que serán sus respectivos nombres (**True, False**).
- **Void**: este tipo de datos solamente será para métodos el cual se utiliza cuando un método no retorna nada.

Tabla 2: Ejemplos de tipos de datos	
Tipo de dato	**Ejemplos**
Int	1 2 45
Double	1.2 25.4 3.333
String	“hola mundo”
Char	‘a’ ‘b’ ‘1’
Boolean	True False
Void	Solo para métodos

6. Sintaxis

A continuación, se detalla la sintaxis de todas las sentencias del lenguaje cbc.

6.1. Encabezado

Al inicio de todo archivo cbc se podrán importar diferentes archivos con funciones o métodos escritos con el lenguaje cbc. Cuando un archivo es importado se podrán utilizar todos su métodos, funciones y variables.

Sintaxis

```
(Import nombreYextencionArchivoComoCadena)*;
```

Ejemplo

```
Import “aritmetica.YYY”;

Import “ordenamiento.YYY”;
```

6.2. Bloque de sentencias

Sera un conjunto de sentencias delimitado por llaves “{ }”, estas definirán un entorno local, es decir las variables declaradas en dicho entorno, únicamente podrán ser utilizadas en dicho entorno o bien en entornos hijos.

Ejemplo

```
Void metodo1() {

//Lista de sentencias

}

If(true){

//Lista sentencias

}
```

6.3. Signos de agrupación

Para dar orden y jerarquía a ciertas operaciones aritméticas y lógicas se utilizarán los signos de agrupación. Los únicos signos de agrupación que soporta el lenguaje son los paréntesis.

Ejemplo

```
Var1=1+((2*3)/3)+var2;
If(var1>=5&&(True|| var2==5)){
        //sentencias
}
```

6.4. Comentarios

Los comentarios permiten añadir al código fuente notas o comentarios de texto que son ignorados por el compilador o interprete. Existirán 2 tipos de comentarios:

De una línea

Iniciaran con los símbolos **//** y finalizaran con salto de línea.

Ejemplo

```
//comentario de una linea
// otro comentario de una linea
```

De múltiples líneas

Iniciarán con los símbolos **/*** y finalizarán con los símbolos ***/**.

Ejemplo

```
/*
Este es un comentario de
* Múltiples líneas
*/
```

6.5. Operaciones aritméticas

Las operaciones aritméticas soportadas por el lenguaje cbc son las siguientes las cuales según el tipo de los operadores y los valores que se están operando da como resultado un nuevo valor y tipo como se describe en la siguiente tabla.

	Tipo Dato	Boolean	Double	String	Int	Char
+	**Boolean**	Double	Double	String	Int	Error
	Double	Double	Double	String	Double	Double
	String	String	String	String	String	String
	Int	Int	Double	String	Int	Int
	Char	Error	Double	String	Int	Int
-	-	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Int	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Int	Double	Error	Int	Int
	Char	Error	Double	Error	Int	Int
*	*	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Int	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Int	Double	Error	Int	Int
	Char	Error	Double	Error	Int	Int
/	/	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Errpr	Double	Double
	Char	Error	Double	Error	Double	Double
%	%	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Error	Double	Double
	Char	Error	Double	Error	Double	Double
^	^	**Boolean**	**Double**	**String**	**Int**	**Char**
	Boolean	Error	Double	Error	Double	Error
	Double	Double	Double	Error	Double	Double
	String	Error	Error	Error	Error	Error
	Int	Double	Double	Error	Int	Int
	Char	Error	Double	Error	Double	Double

6.6. Operaciones relacionales

Además de las operaciones aritméticas, también es posible definir operaciones relacionales, cuyo valor siempre será un valor booleano. Estas operaciones solo son aplicables entre expresiones del mismo tipo, de lo contrario se debe reportar error. Entre datos numéricos, las comparaciones se comportan de manera natural, haciendo la comparación que su nombre indica.

Igual	Diferente	Menor	Mayor	Menor o igual	Mayor o igual
==	!=	<	>	<=	>=

Para las cadenas, las comparaciones deben realizarse siguiente el orden lexicográfico de cada uno de los caracteres que componen las cadenas a comparar. Por ejemplo:

Cadena 1	Cadena 2	Resultado

“Agua”	“Aguacate”	Cadena 2 es mayor que cadena 1
La cadena es idéntica en sus primeros 4 caracteres, pero como la cadena 2 aún tiene caracteres el resultado sería que la cadena 2 es “mayor que” la cadena 1.		
“Compiladores 1”	“compiladores1”	Cadena 1 es mayor que cadena 2
En este caso la cadena 1 es menor que la cadena 2 porque el ascii asociado al espacio en blanco (32) es menor que el valor del ascii asociado al digito 1 (49).		
“Japón”	“Japón”	Cadena 2 es igual que cadena cadena 1
Las cadenas son idénticas.		

6.7. Operaciones lógicas

And (y lógico)	Or (o lógica)	Xor	Not (negación lógica)
&&	||	|&	!

6.8. Precedencia y asociatividad de operadores

En la siguiente tabla se definirá la precedencia y asociatividad para cada uno de los operadores definidos en la sección de expresiones, ordenados de menor a mayor precedencia. Los paréntesis son los únicos símbolos que se utilizarán para agrupar operaciones y romper la precedencia normal de las operaciones.

Símbolo	Precedencia	Asociatividad
+ -	1	Izquierda
*** / %**	2	Izquierda
^	3	Derecha
- **(unario)**	4	No aplica
==, !=, <, >, <=, >=	5	No aplica
||	6	Izquierda
!&	7	Izquierda
&&	8	Izquierda
!	9	Izquierda
( )	10	No aplica

6.9. Declaración de variables

Una variable deberá ser declarada antes de poder ser utilizada. La variable podrá o no tener un valor de inicialización. Es posible declarar dos o más variables a la vez, en tal caso los identificadores estarán separados por comas.

Sintaxis

```
LISTA_ID : TIPO  (= EXPRESION)?;
```

Ejemplo

```
nombre : String = "nombre1"; //declaración e inicialización de una variable

val1,val2,val3 : Int;  //declaración de varias variables
```

Valores por defecto	
Tipo de dato	**Valor por defecto**
Int	0
Double	0.0
String	""
Char	'\u0000' carácter nulo
Boolean	False

6.10. Asignación de variables

Una asignación de variable consistirá en asignar un valor a una variable. La asignación se llevada a cabo con el signo **=**

Sintaxis

```
Identificador_X = EXPRESION;
```

Ejemplo:

```
nombre= "Jose"; // asignacion directa

var2=getMedia(1,2,nombre); // asignación en términos de una expresión

var3= var2+(2*3); //asignación en términos de una expresión
```

Notas

- Deberá verificarse que el tipo de la expresión sea el mismo de la variable a la que se le va asignar dicha expresión, de lo contrario, debe reportarse un error semántico.
- Deberá verificarse que la variable a las que se le esta asignando la expresión exista en el contexto o entorno que contiene dicha sentencia de asignación, de lo contrario se debe reportar un error semántico.

6.11. Arreglos de una dimensión – declaración y asignación

Un arreglo será un conjunto de datos de tipos similares que se identifican con un nombre en común.

Los diferentes elementos contenidos en un arreglo se definen por un índice y se acceden a ellos utilizando su índice; el primer índice inicia en 0.

Sintaxis

```
Identificador : TIPO [EXPRESION] (={datos0, … , datoN})? ;

Identificador : TIPO [EXPRESION] ( = identificador)?; // arreglo que es igualado a otro arreglo

//EXPRESION define el tamaño del arreglo

//dato0, … , representan una lista de EXPRESION
```

Ejemplo

```
Contadores : Int [1+getIndice()]={1,2,3,4,5,6};

Indices : Int [3]=valores;

Nombres : String [1+1];

Nombres={"juan", "jose"};
```

Nota:

- Deberá verificarse que la variable que representa al arreglo no exista en el contexto o entorno al momento de declararla, de lo contrario debe reportarse error semántico.
- Deberá verificarse que las dimensiones del arreglo coincidan con las dimensiones que se definan con la expresión de inicialización, de no ser así, debe reportarse erro semántico.
- Deberá verificarse que el tipo de los elementos de inicialización sea el mismo que el declarado, de lo contrario, debe reportarse error semántico.
- Cuando a un arreglo se le asigne otro arreglo, esta asignación es por valor, es decir, si los valores del arreglo í**ndices** cambian, los valores del arreglo **dato**s no deberían cambiar (ver ejemplo).
- Similar a los arreglos de una dimensión de java.
- Si el arreglo solo es declarado se le asignará valores por defecto como las variables normales.

6.12. Acceso a posiciones de los arreglos

Se podrá acceder a posiciones dentro de un arreglo, especificando la posición a la cual se desea acceder.

Existen 2 formas de acceso

Acceder a una posición de un arreglo para utilizar su valor

En este caso se accede al valor de una posición específica para su utilización en alguna otra sentencia.

Sintaxis

```
Identificador [EXPRESION]
```

Ejemplo

```
Indice = indices[1+1];

nombreCompleto = nombres[1] + nombres[0];
```

Acceder a una posición de un arreglo para asignarle un valor

En este caso se accede a una posición específica para modificar el valor actual con el valor implícito de otra expresión.

Sintaxis

```
Identificador [EXPRESION] = EXPRESION;
```

Ejemplo

```
Indices[0]= getIndice(1,2);

Indices[1]=1;
```

Nota

- Deberá verificarse que la variable que representa al arreglo, exista en el contexto que tiene dicha sentencia de asignación
- Deberá verificarse que las expresiones que representan los índices para acceder a una posición especifica de arreglo, estén dentro del rango definido previamente en la declaración del mismo.
- Deberá verificarse que si se asignará un valor a una posición especifica del arreglo, el tipo de la expresión sea el mismo que el declarado.

6.13. Funciones/métodos

Una función/Método es una subrutina de código que se identifica con un nombre y que puede contar con parámetros (y un tipo para su retorno). Para el lenguaje cbc las funciones serán declarados definiendo primero su nombre, luego dos puntos y luego el identificador, seguido una lista de parámetros (que puede no venir) delimitada por paréntesis. Cada parámetro estará compuesto por su nombre, dos puntos, seguido de su tipo y cada uno de ellos estará separado del

otro por una coma. Después de la definición de la función se declarará el cuerpo de la misma, formado por una lista de instrucciones delimitadas por un juego de llaves { }.

Sintaxis

```
ID_METODO : TIPO ( (ID_PARAMETRO : TIPO)*){
        //SENTENCIAS
}
```

Ejemplo

```
GetMedia : Double (val1 : Int, val2 : Int){
        //Ejemplo de función que devuelve un valor
        Val3 : Double=(val1+val2)/2
        Return val3;
}
Mostrar : Void (){
        //ejemplo de método
        Print("Hola Mundo");
}
Mostrar : Void (nombre : String){
        //ejemplo de método
        Print("Hola Mundo " + cadena);
}
```

Nota

- Las funciones / métodos pueden ser sobrecargadas, lo que significa que puede haber dos o más funciones con el mismo nombre.
- Si es una función no retorna ningún valor deberá reportarse un error semántico.

6.14. Función Main

La función Main es la subrutina que arrancara las acciones del interprete/compilador.

Sintaxis

```
Main : TIPO ( (ID_PARAMETRO : TIPO)*){
        //SENTENCIAS
}
```

Ejemplo

```
Main : Void (operacion : String){
        //ejemplo de método
        Print("Hola Mundo " + cadena);
}
Main : String (operacion : String){
        //ejemplo de método
        Return "ProductoX";
}
```

Nota

- Main puede ser función o método, es decir, puede devolver algún valor o no.

6.15. Llamadas a funciones/métodos

Una llamada a función/método se realiza escribiendo el identificador de la función/método, seguido de los valores que tomarán los parámetros de dicha llamada, separados por coma y envueltos en un juego paréntesis finalizando con punto y como. También se debería poder utilizar las llamadas en expresiones.

Sintaxis

```
ID_METODO_FUNCION ( (LISTA_EXPRESIONES)* );
```

Ejemplo

```
Mostrar();

Media : Double = getMedia(1,2);
```

6.16. Sentencia Return

Esta sentencia finalizará la ejecución de la función y devolverá el valor indicado por la expresión que se encuentra adyacente a ella. Si durante la ejecución del código se encuentra la palabra 'Return' sin ninguna expresión asociada se terminará la ejecución del cuerpo del método (Solo aplica para métodos la sentencia return sin expresión asociada). Esta sentencia inicia con la palabra reservada 'Return' luego, puede o no venir una expresión y finaliza con un punto y coma (;).

Sintaxis

```
Return EXPRESION;
```

Ejemplo

```
Return 1+indices[2]+getVariable(); //En una función

Return ; //En un metodo
```

6.17. Sentencia If

Sentencia de selección que bifurcará el flujo de ejecución debido a que proporciona control sobre dos alternativas basadas en el valor lógico de una expresión (Condición).

Sintaxis

```
If ( EXPRESION ) {
        //SENTENCIAS
} (Else{
        //SENTENCIAS
})?
```

Ejemplo

```
If ( var1>5 && True ){
        //SENTENCIAS
}
If ( var1==5){
        //SENTENCIAS
}Else{
        //SENTENCIAS
}
```

Nota

- La sentencia Else puede o no venir.

6.18. Sentencia Switch

Este control de flujo evalúa una expresión y compara el resultado de dicha expresión con cada uno de los valores definidos en los casos fijos del cuerpo de dicha sentencia, cada caso cuenta con un valor fijo y su correspondiente cuerpo de instrucciones que ejecutará toda vez el valor obtenido producto de haber evaluado la expresión sea igual al valor fijo de dicho caso.

Los tipos de datos permitidos en este control de flujo son solamente Strings, Double e Int, que a su vez son mutuamente excluyentes, es decir: Si deseo SELECCIONAR una expresión de tipo String los valores para los CASOS deben ser valores de tipo String, de igual forma sucedería con el tipo Double e Int.

El comportamiento de este control de flujo pregunta cuál es la condición que se cumple y de ahí en más ejecuta todas las sentencias de todos los casos (sin Importar el cuerpo asociado al bloque 'Default') hasta encontrar una sentencia 'Break'. Si no se cumple ninguna de las condiciones se debe ejecutar el bloque de sentencias asociado a la palabra reservada 'Default'.

Sintaxis

```
Switch ( EXPRESION ){
        (Case VALOR :
                //SENTENCIAS
                (Break)?
        )+
        (Default :
                //SENTENCIAS
                (Break)?
        )?
}
```

Sintaxis

```
Switch ( varX ){
        Case 1:
                //SENTENCIAS
                Break;
        Case 2:
                //SENTENCIAS
        Case 3:
                //SENTENCIAS
                Break;
        Default:
                //SENTENCIAS
}
```

6.19. Sentencia For

Este ciclo inicia con la palabra reservada 'For' consta de una declaración inicial de una variable de tipo Double o Int; un punto y coma; una condición para mantenerse en el ciclo; un incremento (++) o decremento (--) que al final de cada iteración realizada afectará directamente a la variable declarada al inicio de este ciclo y un cuerpo de sentencias a ejecutar siempre y cuando la condición sea verdadera. El ciclo de cada iteración será: Evaluar la condición; si es verdadera, ejecutar cuerpo y realizar incremento o decremento, regresar a evaluar la condición; si es falsa, salir del ciclo.

Sintaxis

```
For ( Identificador : TIPO = EXPRESION ; EXPRESION ; INCREMENTO_O_DECREMENTO ){
        //SENTENCIAS
}
```

Ejemplo

```
For ( i : Int = 0 ; i<10 ; i++){
        //SENTENCIAS
}
```

6.20. Sentencias While

Este ciclo consta de una estructura que inicia con la palabra reservada 'While', seguido de una condición (envuelta entre paréntesis) y al final constará con un cuerpo de sentencias (delimitadas por un juego de llaves { }) que se ejecutarán toda vez la condición del ciclo sea verdadera.

Sintaxis

```
While ( EXPRESION ) {
        //SENTENCIAS
}
```

Ejemplo

```
While ( var1<10 ){
        //SENTENCIAS
}
```

6.21. Sentencia Do While

Similar a la sentencia While solo que empieza a verificar la condición después de la primera iteración.

Sintaxis

```
Do{
        //SENTENCIAS
} While ( EXPRESION )
```

Ejemplo

```
Do{
        //SENTENCIAS
} While ( var1<10 )
```

6.22. Sentencia Break

Esta sentencia es aplicable toda vez se encuentre dentro del cuerpo de una sentencia While, For o dentro del cuerpo de un caso específico de una sentencia Switch. Al encontrarla la ejecución deberá detenerse y regresar el foco de control al nivel superior de la sentencia que ha sido detenida.

Sintaxis

```
Break;
```

7. Funciones nativas

7.1. Sentencia Print

La función 'Print' se encargará de imprimir en consola la lista de expresiones que reciba como parámetros.

Sintaxis

```
Print ( EXPRESION );
```

Ejemplo

```
Print ("Hola Mundo");

Print ( "indice: "+ var1 +" valor: "+arreglo[var1] );
```

7.2. CompareTo

Es una función nativa de los tipos de datos String, recibirá un parámetro para verificar que si existe en una cadena.

Sintaxis

```
descripcion.CompareTo(EXPRESION); /*EXPRESION solo puede ser un valor de tipo String, cualquier otro debe reportarse error semantico*/
```

Ejemplo

```
descripcion.CompareTo("cantidad"); /* si descripcion contiene la cadena "cantidad" debe retornar True, de lo contrario False*/

If(descripcion.CompareTo("cantidad")){

        Print("Existe");

}else{

        Print("No Existe");

}
```

7.3. GetUser()

Funcion nativa que devuelve el nombre y el apellido del usuario que actualmente esta logeado.

Ejemplo

```
Print(GetUser());

/* Por ejemplo si el nombre del usuario es Carlos y apellido es Rodriguez, la función Print debera mostrar:

Carlos Rodriguez

*/

nombreCompleto:String=GetUser();
```

Nota

- A una variable nomal o alguna posición de un arreglo de tipo String se le podrá asignar la variable GetUser()

8. Almacenamiento de usuarios

De alguna manera se deberán de guardar los usuarios registrados para que los datos persistan en el tiempo, es decir, se puede caer el sistema y después de levantarse los datos no se deben perder.

9. Mensajes y respuestas en la solución

Los mensaje y respuestas entre los componentes de la solución deberán tener el siguiente formato:

Registro:

```
Registro-Request[

nombre : valor ,

apellido : valor,

dpi : valor,

correo : valor,

contrasenia : valor

]
```

Respuesta al registro:

```
Registro-Response[
estado : valor
]
```

Estado puede ser 0 o 1, 1 si el usuario fue ingresado correctamente y 0 si fue lo contrario.

Login:

```
Login-Request[
correo : valor,
contrasenia : valor
]
```

Respuesta

```
Login-Response[
estado: valor
]
```

Estado será 1 si el usuario existe y el login fue satisfactorio y 0 si fue lo contrario.

Mensaje

```
Mensaje-Request[
Mensaje : valor
]
```

Respuesta

```
Mensaje-Response[
Mensaje : valor
]
```

El formato se usará en ambos lados, es decir en el lado del proveedor y del integrador (servidor).

Nota:

- En el lado del integrador obligatoriamente se debe usar un analizador parar reconocer cada uno de los response.

10. Aplicación móvil/Web (Chat)

Por ser la primera versión del proyecto, solamente necesita 3 módulos:

- Registrar Usuario: para que una persona puede usar el Chatbot deberá registrarse. Los campos necesarios para registrarse son los siguientes: nombre, apellido, dpi, correo y contraseña.
- Login: cuando un usuario desee ingresar al chat, deberá iniciar sesión.
- Chat: es el modulo principal y por el cual la empresa ha decidido contar con sus servicios, su principal función es mantener una conversación con el usuario, para mantener la conversación necesitara de un bot y es aquí donde el lenguaje cbc se necesitará para verificar que es lo que el usuario escribió en el chat y que es lo que se le debe responder.

Nota:

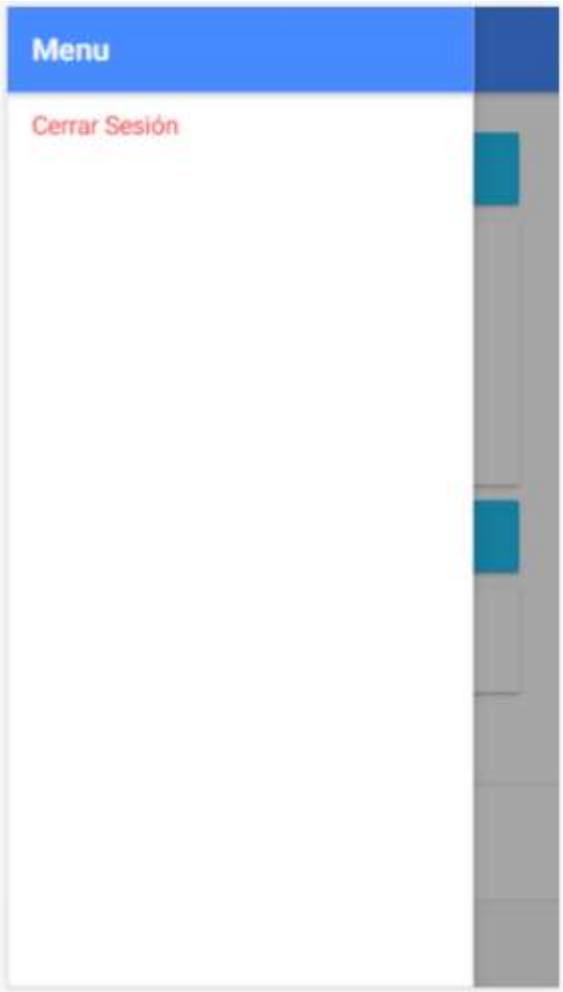
- Para enviar mensajes desde el chat se podrá de 2 formas, la primera es que el usuario enviará dichos mensajes uno por uno mientras que la segunda es una carga masiva de mensajes para que la calificación sea fácil.

- En la carga masiva por cada mensaje el Integrador deberá responder (por cada mensaje se deberá ejecutar el método Main).
- Cada mensaje del archivo de la carga masiva estará divido por un salto de linea
- En la carga masiva se deberá leer un archivo de entrada en el lado del chat

Ejemplo de un archivo de carga masiva de mensajes

```
Hola
Productos
Precios
Horarios
```

10.1. Interfaz gráfica sugerida



11. Funcionamiento de la aplicación

1. En el IDE deberá crearse código válido para automatizar las respuestas
2. Teniendo el código valido para automatizar las respuestas, ya se tiene al integrador listo para responder a cada una de las solicitudes del proveedor (Chat).
3. Cada vez que el usuario envíe un mensaje desde el chat, se deberá activar el integrador (elemento de la solución donde se encuentra el proceso de reconocimiento de lo que envía el usuario por medio del chat) y procesar una respuesta. Sera el método main que se activará cada vez que se envíe un mensaje por lo que este método recibirá un parámetro que tomara el valor del **mensaje** que el usuario escribirá en el chat.
4. Luego de reconocer el mensaje y generar una respuesta por parte del integrador mediante el método main, se deberá devolver dicha respuesta con el formato correcto utilizando la palabra reservada Return del lenguaje **cbc**.

Ejemplo

Mensaje en el chat

```
Hola
```

Integrador

```
Main (mensaje:String){
        If(mensaje==”hola”){
                Return “hola” + GetUser() +”!”;
        }
        Return “Error”;
}
```

Respuesta en el chat

```
Hola
Hola Carlos Rodriguez!
```

Nota:

- Habrá diferentes niveles de dificultad en los archivos de calificación, el ejemplo anterior es lo más básico que puede existir en la solución.
- Para aumentar el nivel de dificultad, se usarán búsquedas en arreglos, ciclos anidados, búsquedas en las cadenas por medio de la función nativa CompareTo.

12. Requerimientos mínimos

- IDE
- Los 3 elementos del Chat
- Comunicación entre los elementos de la solución mediante Web Services
- Import
- Main
- Arreglos
- Sentencia IF
- Sentencia Switch
- Sentencia While
- Sentencia For
- Funcion Nativa Print
- Declaración de variables
- Declaración de Métodos
- Operaciones aritméticas, lógicas y relacionales
- Llamadas a métodos
- Parámetros
- Tipos de datos String, Int, Double y Boolean
- Sentencia Return
- Uso del parámetro del método Main

13. Reportes

Deberá de crearse un documento pdf para reportar los siguientes errores

- Léxicos
- Sintácticos
- Semánticos

14. Documentación

- Manual técnico
- Manual de usuario

15. Restricciones

- El proyecto debe realizarse de forma individual
- El analizador debe implementarse en el lenguaje C# en caso contrario no será calificado y se tendrá una nota automática de 0.
- Para leer los archivos de entrada, se debe utilizar la herramienta Irony.
- Se tomará en cuenta la calidad de la información proporcionada por el compilador cuando se produzcan errores, así como la presentación de la interfaz gráfica y la amabilidad de la aplicación.
- Copias de proyectos o de gramáticas tendrán una nota de 0 puntos y el respectivo reporte al ingeniero, así como también el reporte a la escuela de ciencias y sistemas.
- La calificación del proyecto será personal y durará 1 hora, en un horario que posteriormente será establecido. Se debe tomar en cuenta que durante la calificación no podrán estar terceras personas alrededor o de lo contrario no se calificará el proyecto.
- Todos los errores generados por la aplicación deberán ser mostrados en PDF o de lo contrario no serán calificados
- No se permitirá el uso de librerías para analizar los mensajes en el lado del integrador.
- Sistema operativo libre
- Para el Web services se deberá utilizar .net
- Para el IDE se deberá usar .net, podrá ser aplicación de escritorio o web
- Para el Proveedor (Chat) se deberá utilizar el Framework Angular 2+ si será aplicación Web, Ionic o Android nativo si será aplicación móvil.
- La entrega será por classroom
- Todos los elementos de la solución (código fuente, manuales) en una sola carpeta y comprimida con el siguiente nombre: [OLC1]Proyecto2_#Carnet.rar

16. Fecha de entrega

10 de noviembre de 2018 antes de las 23:59 hrs