Universidad de San Carlos de Guatemala Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas
Organización de Lenguajes y Compiladores 2
Escuela de vacaciones de diciembre 2019
Catedrático: Ing. Carlos Olivio Natareno Yanes
Tutor académico: Ricardo Antonio Menchú Barrios.

Lenguaje RMB

Contenido

Objetivos

Objetivo general

- Aplicar los conocimientos del curso de Organización de Lenguajes y Compiladores 2 en el desarrollo de una aplicación mediante un lenguaje de programación estructurado.

Objetivos específicos

- Utilizar herramientas para la generación de analizadores léxicos y sintácticos.
- Realizar análisis semántico a los diferentes lenguajes a implementar.
- Que el estudiante aplique los conocimientos adquiridos durante la carrera para el desarrollo de un intérprete.
- Que el estudiante sea capaz de implementar los conocimientos adquiridos en el curso tecnologías modernas.

Descripción

RMB se trata de un lenguaje de tipos de datos estáticos, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a bajo nivel.

Descripción de la aplicación

Se requiere la implementación de un entorno de desarrollo de escritorio el cual servirá tanto para el desarrollo de aplicaciones, así como también para la ejecución de la misma. La aplicación contará con los siguientes componentes:

- Lenguaje RMB

Es un lenguaje de programación estructurado, procedural e imperativo. Esto significa que todos los programas deben tener una estructura y orden definido.

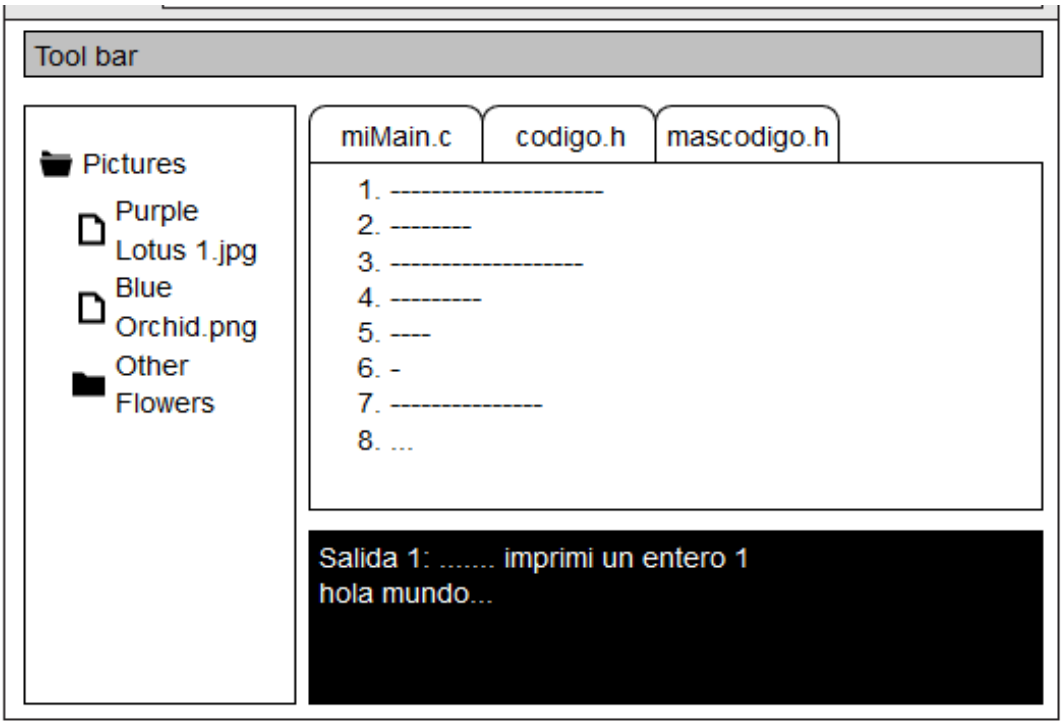
- R-IDE

Es un entorno de desarrollo. Este provee las herramientas para la escritura de programas en Lenguaje RMB. El entorno de desarrollo debe oder colorear la sintaxis, la configuración de colores será la misma que utiliza el editor de texto “sublime” con la sintaxis del lenguaje “C”.

El ide tendrá las siguientes opciones:

- Guardar archivo
- Guardar proyecto
- Abrir archivo
- Abrir proyecto

- Nuevo proyecto
- Nuevo archivo
- Correr proyecto
- Configurar archivo principal



- Consola

Este componente permitirá observar el resultado de ejecución de las aplicaciones desarrolladas en el IDE.

- Reportes

Permite generar reportes sobre el proceso de “compilación” del archivo de entrada. Estos reportes nos permiten tener una idea más detallada sobre el funcionamiento de nuestro compilador ya que nos presenta los errores y/o la tabla de símbolos utilizada. Todos los reportes deben ser presentados en un archivo html.

Ejemplo de reporte de errores

Tipo	Descripcion	Fila	columna	archivo
semantico	la estructura ID no existe	11	11	mi_estructura.h
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----

- Reporte de errores

Mostrará todos los errores encontrados en el proceso de compilación de una manera detallada y en formato de tabla. Se debe mostrar el tipo de error, su ubicación, así como una breve descripción que indique que fue lo que sucedió.

- Reporte de tabla de símbolos

Este reporte mostrará la tabla de símbolos después del proceso de "compilación". Se deberán demostrar todas las variables, funciones y procedimientos que fueron declarados, así como su tipo y toda la información que el estudiante considere necesaria. Se calificará la presentación del reporte

Generalidades del lenguaje

RMB es un lenguaje derivado de C, por lo tanto, está conformado por un subconjunto de sus instrucciones que pretenden facilitar el aprendizaje de programación a las personas, utilizando la programación estructurada y estructuración de datos.

Importes

Este se compone de 3 tipos de archivos diferentes:

- Archivos R

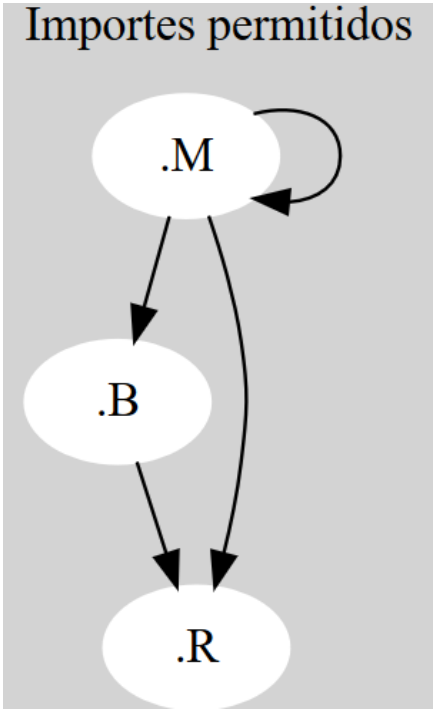
Estos archivos pueden contener código RMB en su interior. Este archivo puede ser llamado para iniciar el proceso de ejecución de un programa, por lo tanto puede contener el método principal (Nucleo). Estos pueden importar código tanto de archivos B como M.

- Archivos M:

Estos archivos pueden contener código RMB en su interior. A diferencia de los archivos R estos no pueden ser llamados para iniciar el proceso de ejecución, esto quiere decir que no puede contener un método principal en su interior. Estos pueden importar código de otros archivos M.

- Archivos B

Estos archivos además de poder contener código RMB en su interior, contiene la configuración grafica de un formulario o ventada. Al igual que los archivos M estos no pueden ser llamados para iniciar el proceso de ejecución, esto quiere decir que no puede contener un método principal en su interior. Estos pueden importar código de otros archivos M.



Tipo de datos

RAM es un lenguaje de programación de tipado estático, esto significa que la comprobación de tipos se realiza durante la compilación y las variables no pueden cambiar su tipo durante la ejecución.

Tipos primitivos

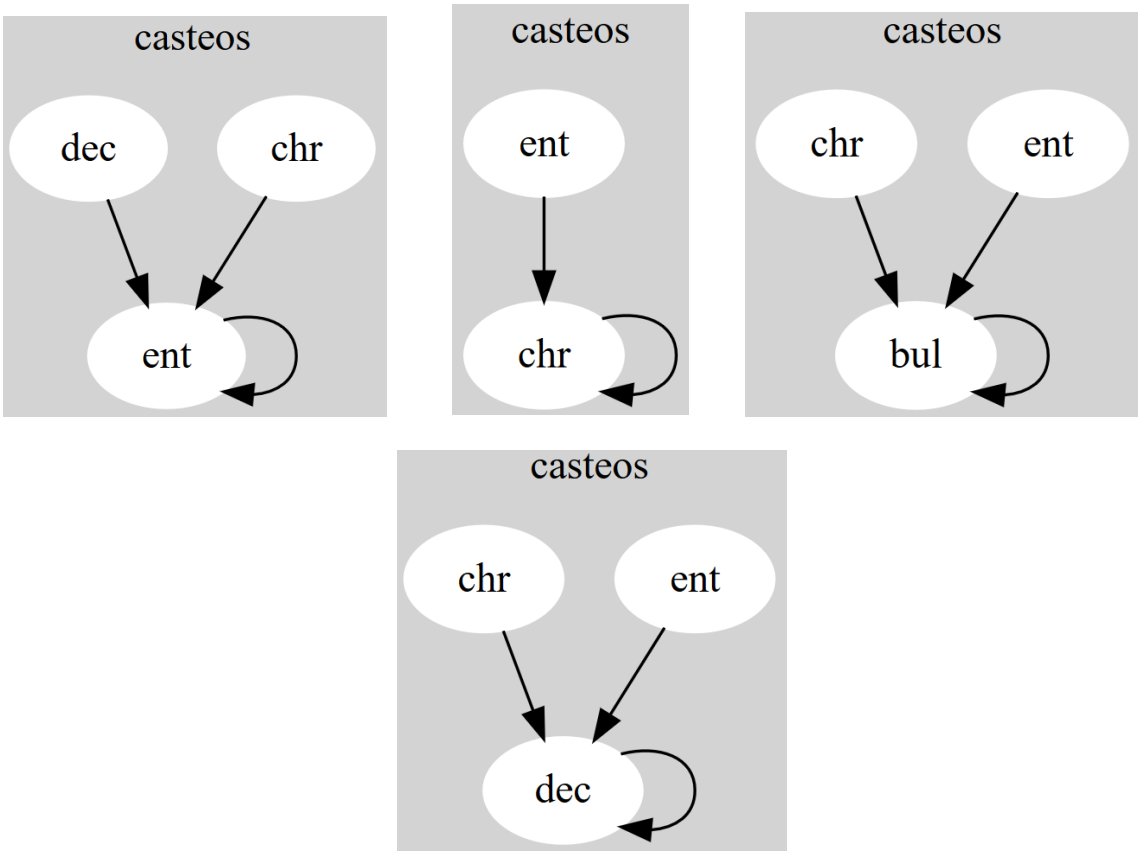
Se utilizarán los siguientes tipos de dato primitivos:

Tipo	Definición	Rango
ent	Acepta valores numéricos enteros	[-2147483648, 2.147.483.647]
dec	Acepta valores numéricos de punto flotante	[-9223372036854775800, 9223372036854775800]
chr	Acepta un único carácter	[0,255] (ASCII)
bul	Valores de verdadero y falso	true, false

nlo	**Representa la ausencia de valor, será colocado en registros sin valor o reserva de espacio**	**-1**

Casteos implícitos

Además de la sobrecarga que posee en operador más (+) en las operaciones aritméticas, RMB permite realizar casteos implícitos al momento de asignar valores a las variables, tanto para variables definidas por el usuario como los parámetros de una función o procedimiento. Se presentan las siguientes gráficas de casteos permitidos:



Arreglos

RMB proporciona una estructura de datos llamada arreglo que puede almacenar una colección secuencial de tamaño fijo de elementos del mismo tipo. En lugar de declarar variables individuales, como número1, número2, ... y número100, declara una variable de matriz como números y usa números [1], números [2] y ..., números [100] para representar variables individuales Se accede a un elemento específico en una matriz mediante un índice.

Ejemplo:

```
...
dec id[10]; // todos los valores estarian en 0.0
ent id_2[x+27];
//si no lleva tamaño de la dimencion explicita
//la asignación de valor es obligatoria
chr id_3[] = {'s','i'};
chr id_4[2] = {'n','i'};
chr Cadena [10] = "hola mundo";
...
```

Arreglos Multidimensionales y Arreglos de arreglos

Los arreglos de arreglos son estructuras de datos estáticas en la que tienen n dimensiones para todo n>1. En RMB, los arreglos de arreglos se declaran de la misma forma que se declararía un arreglo multidimensional convencional en otros lenguajes. Se presenta el siguiente ejemplo:

```
ent miArrydeArray [2][2];
ent valores [2][2][3];
ent valores [][][] = {{{1,2,3},{1,2,3}},{{1,2,3},{1,2,3}}};
```

Acceso a los arreglos

Se accede a un elemento de un arreglo mediante el nombre del arreglo indicando la posición a la que se desea acceder.

```
...

chr a[2][2] = {{'1','2'},{'3','4'}};
chr b[2] = a[0]; // = {'1','2'}
chr z[] = "arreglo de caracteres";
ent b = z[0]; // valor ascii de a

...
```

Consideraciones:

- Para un arreglo de n dimensiones, si se hace un acceso sobre una dimensión m donde m<n, al acceso retornara un arreglo de n-m dimensiones.

Fusion

La principal limitación de un arreglo es que todos los datos que contiene deben ser del mismo tipo. Pero a veces nos interesa agrupar datos de distinta naturaleza, como pueden ser la edad y el nombre de una persona, que serían del tipo entero y un arreglo de caracteres, respectivamente. En ese caso, podemos emplear las fusiones, que se definen indicando el nombre y el tipo de cada dato individual (cada campo), y se accede a estos campos indicando el nombre de la variable y el nombre del campo, separados por un punto:

```
fusion persona {
    ent edad;
    chr nombre[20];
    persona siguiente;
};

zro  unMetodo (ent a, char b[]){

    persona obj; // en este punto con valor nulo
    obj = _reservar(_pesode(persona));
    // a partir de este punto ya se puede acceder a los
    // atributos de obj
    obj.edad = a;
_copi(obj.nombre,b);
// no esta permitiro una asignación del tipo
// obj.nombre = b;

    ...

}
```

Memoria Virtual

RMB permite realizar una manipulación directa de la memoria virtual o como se denomina en el entorno de ejecución el “Heap”. Para la manipulación de dicha memoria el lenguaje contara con 2 funciones propias y únicamente serán aplicables para la manipulación de fusiones.

- ### _pesode

Este método retornara un entero que indicara el tamaño de un registro. El tamaño de un registro se calcula como la suma de cada uno de sus atributos. Si el registro tuviera como atributos otra fusion o un arreglo, también contaría como un espacio ya que únicamente se almacenaría su apuntador.

```
fusion persona {
    ent edad;
    chr nombre[20];
};
ent cant = _pesode(persona); // cant = 2

fusion persona2 {
    ent edad;
    chr nombre[20];
    dec salario;
};
ent cant2 = _pesode(persona2); // cant2 = 3
```

- ### _reservar

Este método recibirá como parámetro un entero, reservará en el “heap” dicha

cantidad de memoria y retornará un puntero hacia la posición que se reservó. Esta operación deberá de realizarse cada vez que se quiera instanciar un registro, en caso de no reservarse memoria, el registro apuntará a nil y si se intenta utilizar se producirá un error en tiempo de ejecución.

```
fusion persona {
    ent edad;
    chr nombre[20];
};

zro  unMetodo (ent a, char b[]){

    persona obj; // en este punto con valor nulo
    obj = _reservar(_pesode(persona));
    // a partir de este punto ya se puede acceder a los
    // atributos de obj
    obj.edad = a;
_copi(obj.nombre,b);
// no esta permitiro una asignación del tipo
// obj.nombre = b;

    ...

}
```

Definición léxica

Comentarios

Un comentario está destinado a incrustar anotaciones legibles al programador en el código fuente de un Programa
Existirán dos tipos de comentarios:

- Los comentarios de una línea que serán delimitados al inicio con el símbolo “//”.
- Los comentarios con múltiples líneas que empezarán con los símbolos “/*” y terminarán con los símbolos “*/”.

```
// este es un ejemplo de un lenguaje de una sola línea

/*

este es un ejemplo de un lenguaje de múltiples líneas.

*/
```

Sensibilidad a mayúsculas y minúsculas

RMB será un lenguaje case sensitive, por lo tanto, nuestro compilador hará distinción entre mayúsculas y minúsculas, tanto para palabras reservadas como para identificadores

Identificadores

Los identificadores son símbolos léxicos que nombran entidades. Estas entidades pueden ser variables, métodos o módulos. 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.

A continuación, se muestran ejemplos de identificadores validos:

```
Identificador_valido_1
identificador1
sale_Compi_en_vacas
```

A continuación, se muestran ejemplos de identificadores no validos:

```
valor.2
104sdls
```

Palabras reservadas

Las palabras reservadas son identificadores reservados predefinidos que tienen un significado especial y no se pueden ser utilizados como identificadores en el programa.

zro	ent	chr	dec	bul
if	while	for	repeat	switch
case	default	romper	siga	definir
fusion	importar	regresar	when	

Literales

Representan el valor de un tipo primitivo

- Enteros: [0-9]+
- Decimales: [0-9]+(“.” [0-9]+)
- Caracter: “ ’ ” <Carácter ASCII> “ ’ ”
- Cadena: “ ” ” <Caracteres ASCII> “ “ ”
- Booleanos: [true, false]
- Nulo: nlo

Secuencias de escape

Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto. Las secuencias de escape disponibles son las

siguientes:

Secuencia de escape	**Significado**
\'	Comillas simple
\"	Comillas doble
\?	Signo de interrogación
\\	Barra invertida
\n	Nueva linea
\%	Signo de porcentaje

Estructura del Lenguaje

Un programa en RMB básicamente consiste en las siguientes partes

- Importe de codigo
- Declaraciones de tipos (Types)
- Declaraciones de constantes
- Declaraciones de variables
- Declaración de funciones y procedimientos
- Bloque del programa principal

Declaración de variables

Una variable no es más que un nombre dado a un área de almacenamiento que nuestros programas pueden manipular. Cada variable en RMB tiene un tipo y un identificador asociado que no puede cambiar a lo largo de la ejecución.

Las variables declaradas fuera de cualquier método (incluyendo el principal) o función serán globales.

```
ent varA = 2;//globales
ent varB = 3;

ent unaSuma (){

    ent varC = varA + varC;
    regresar varC;

}
```

Consideraciones:

- No puede declararse una variable con el mismo identificador de una variable existente en el ámbito actual, de suceder debe reportarse un error.
- La inicialización es opcional y el tipo de la expresión debe ser el mismo que el declarado, de lo contrario debe reportarse un error.
- No pueden definirse nombre de variables con el mismo nombre que una función o procedimiento.

Declaración de constantes

Las constantes se definen fuera de cualquier método (incluyendo el principal) o función, a diferencia de las variables estos no pueden cambiar de valor a lo largo de toda la ejecución del programa. Siempre serán de ámbito global.

```
#definir hola 25
#definir arreglo {{1,2,3},{4,10,8}}
```

Consideraciones:

- No puede declararse más de una constante con el mismo identificador.
- No pueden definirse nombre de constantes con el mismo nombre que una función o procedimiento.

Expresiones

Tendrá soporte a diferentes operaciones, todas estas serán descritas en conjunto como operaciones, las expresiones son:

Operaciones Aritméticas

Contará con un conjunto de operaciones aritméticas. Una operación aritmética es la acción de un operador sobre los elementos de un conjunto. El operador toma los elementos iniciales y los relaciona con otro elemento de un conjunto final que puede ser de la misma naturaleza o no. Para la precedencia de operadores consultar Apéndice A.

- Suma

Consiste en la adición de dos o más elementos para llegar a un resultado final

donde todo se incluye. El símbolo de la suma es el símbolo más (+) y se intercala entre los elementos que se quiere sumar.

Operandos	Tipo resultante	Ejemplos
ent + dec dec + ent dec + chr chr + dec dec + dec	dec	100 + 4.9 = 104.9 17.8 + 'a' = 114.8 2.5 + 2.5 = 5.0
ent + chr chr + ent ent + ent chr + chr	ent	8 + 'a' = 105 10 + 15 = 25 'a' + 'a' = 194

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

- Resta

La resta es una operación aritmética básica a través de la cual se le disminuye al minuendo la cantidad indicada en el sustraendo. El símbolo de la resta es el símbolo menos (-) y se intercala entre los elementos que se quiere restar.

.

Operandos	Tipo resultante	Ejemplos
ent – dec dec - ent dec - chr chr - dec dec - dec	dec	8 - 1.5 = 6.5 200.0 - 'b' = 102.0 3.5 - 3.5 = 0
ent - chr chr - ent ent – ent chr - chr	ent	95 - 'd' = -5 50-10 = 40

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

- Multiplicación

Consiste en calcular el resultado (producto) de sumar un mismo número (multiplicando) tantas veces como indica otro número (multiplicador). La

multiplicación se representa por el símbolo (*) y se intercala entre los elementos que se quiere multiplicar.

Operandos	Tipo resultante	Ejemplos
int * dec dec * ent dec * chr chr * dec dec * dec	dec	10.25 * 12.3 = 1550.72 20.1 * 5 = 80.4 2.5 * 'b' = 240.0 3.5 * 8.5 = 29.75
ent * chr chr * ent ent * ent chr * chr	ent	2 * 'a' = 194 'd' * -3 = -300

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

- Potencia

La potenciación es una operación matemática entre dos términos denominados: base y exponente. Se escribe y se lee normalmente como «a elevado a la n». La potencia se representa por el símbolo (^).

Operandos	Tipo resultante	Ejemplos
ent ^ ent chr ^ ent ent ^ chr chr ^ chr	ent	2^5 = 32 'a'^2 = 9409

Consideraciones:

- El primer operando es la base y el segundo es el exponente.
- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

- Modulo

Operación aritmética que retorna como resultado el residuo de una división. El módulo se representa por el símbolo (%) y se intercala entre los elementos entre los que se quiera obtener el módulo.

Operandos	Tipo resultante	Ejemplos
int % dec dec % ent dec % chr chr % dec dec % dec	dec	16 % 3.5 = 2.0 'a' % 94.5 = 2.5 10.5 % 3.4 = 0.3
ent % chr chr % ent ent % ent chr % chr	ent	10 % 3 = 1 'b' % 100 = 98

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

- División

La división es una operación matemática o aritmética que consiste en averiguar cuántas veces un número (el divisor) está contenido en otro número (el dividendo). La división se representa con el símbolo (/) y se intercala entre los elementos entre los que se quiera obtener el resultado.

Operandos	Tipo resultante	Ejemplos
ent / dec dec / ent dec / chr chr / dec dec / dec	dec	15 / 5.0 = 3.0 250.0/ 'x' = 2.083
ent / chr chr / ent ent / ent chr / chr	ent (se descartan los decimales)	3 / 2 = 1 'a' / 8 = 12

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.
- Si el divisor es 0, se tendrá que reportar un error en tiempo de ejecución.

Operadores relacionales

Una operación relacional es una operación que compara la igualdad, diferencia o comportamiento entre dos valores. Este tipo de comparaciones siempre devuelve un valor lógico según el resultado de la comparación (true | false). Una operación relacional siempre tiene dos operandos y un único operador.

Operador relacional	Operador relacional	Ejemplos de uso
int [>,<,>=,<=] dec dec [>,<,>=,<=] ent dec [>,<,>=,<=] chr chr [>,<,>=,<=]dec ent [>,<,>=,<=] chr chr [>,<,>=,<=] ent dec [>,<,>=,<=]dec ent [>,<,>=,<=]ent chr [>,<,>=,<=]chr	>,<,>=,<=	15 < 10 = false 4.5 < 3 = false 50.58 >= 'b' = false 'b' <= 107 = true 'a' <= 100 = true 200.25 > 52.2 = true 0 <= 0 = true 'a' < 'a' = false
ent [=,<>]dec dec [=,<>] ent dec [=,<>] chr chr [=,<>]dec ent [=,<>] chr chr [=,<>] ent dec [=,<>]dec ent [=,<>] ent chr [=,<>] chr bul [=,<>] bul	=,<>	50 = 50.0 = true 5.5 <> 30 = true 'a' = 'a' = true 1.2 <> 0.2 = false true = true = true true <> true = false

Consideraciones:

- Cualquier otra combinación que no esté listada en la tabla deberá reportarse como un error semántico.

Operaciones lógicas

Las operaciones lógicas son expresiones matemáticas cuyo resultado es un valor booleano (true o false). Estas expresiones se utilizan principalmente en las estructuras de control. Para la precedencia de operadores consultar Apéndice A. Se presentan las operaciones lógicas que RMB posee, así como sus tablas de

verdad:

- AND

Multiplicación lógica o conjunción
Signo utilizado “&&”

Operando A	Operando B	AAND B
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

- OR

Suma lógica o disyunción
Signo utilizado “||”

Operando A	Operando B	A OR B
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE

Consideraciones:

- Ambos operadores deberán ser de tipo booleano.

- NOT

Negación
Signo utilizado “!”

Operando B	!B
FALSE	TRUE
TRUE	FALSE

Funciones y Metodos

En RMB es una función es un subprograma que retorna siempre un valor, mientras que un procedimiento es un subprograma que realiza un conjunto de acciones sin ningún retorno.

- ## Funciones

Una función es un grupo de declaraciones que juntas realizan una tarea. Cada programa tiene al menos una función, que es el programa en sí, y todos los programas más triviales pueden definir funciones adicionales. Una declaración de una función le dice al compilador sobre el nombre, el tipo de retorno y los parámetros de una función. Una definición de función proporciona el cuerpo real de la función.

La forma general de una definición de función es la siguiente:

```
chr  unaFuncion (ent a, char b[]){

    persona obj; // en este punto con valor nulo
    obj = _reservar(_pesode(persona));
    // a partir de este punto ya se puede acceder a los
    // atributos de obj
    obj.edad = a;

    ...

}

ent[] unaFuncion (ent a, char b[]){

    persona obj; // en este punto con valor nulo
    obj = _reservar(_pesode(persona));
    // a partir de este punto ya se puede acceder a los
    // atributos de obj
    obj.edad = a;

    ...

}
```

Consideraciones

- Si el tipo de retorno y el valor retornado no coinciden, se deberá de reportar un error.
- Los parámetros pueden ser de cualquier tipo.
- Las funciones soportan todo tipo de llamadas recursivas
- Las funciones soportan sobrecarga de parámetros
- La función puede no definir ningún argumento.

- ## regresar

En RMB existe una palabra reservada para la sentencia return para la devolución de resultados en una función.

```
chr  funcion (ent a, char b[]){

    ...
    regresar 'z';

}
```

Consideraciones:

- Debe verificarse que el valor de la expresión sea del mismo tipo del que fue declarado en dicha función.

- Metodos

En RMB, un procedimiento se define utilizando la palabra clave de zro. La forma general de una definición de metodo es la siguiente:

```
zro unaFuncion (ent a, char b[]){

    persona obj; // en este punto con valor nulo
    obj = _reservar(_pesode(persona));
    // a partir de este punto ya se puede acceder a los
    // atributos de obj
    obj.edad = a;

    ...

}
```

Consideraciones

- Los parámetros pueden ser de cualquier tipo.
- No debe retornar ningún valor.

- Parámetros

Si un subprograma (función o metodo) va a usar argumentos, debe declarar variables que acepten los valores de los argumentos. Estas variables se denominan parámetros formales del subprograma.
Los parámetros formales se comportan como otras variables locales dentro del subprograma y se crean al ingresar al subprograma y se destruyen al salir.

Paso Por Valor

Este copia el valor real de una variable a otra. En este caso, los cambios realizados en la variable dentro del subprograma no tienen efecto en la variable original.

```
chr a; // ''
chr b = 'y';

a = b; // a = 'y'
a = 'z';
_imp("valor de a: %c",a); //valor de a: z
_imp("valor de b: %c",b); //valor de b: y

```

Paso Por Referencia

Este toma el valor real de una variable a otra. En este caso, los cambios realizados en la variable dentro del subprograma tienen efecto en la variable original.

```
chr a; // ''
chr b = 'y';

a = *b; // a = 'y' <=> b = 'y'
a = 'z';
_imp("valor de a: %c",a); //valor de a: z
_imp("valor de b: %c",b); //valor de b: z

```

Funciones Propias del lenguaje

RMB posee un conjunto de funciones propias del lenguaje que permiten una experiencia más agradable al usuario. Estas son:

- _atxt

Recibe como parámetro un entero, un decimal o un Rstring. Devuelve un arreglo de caracteres

```
chr a[5]; // ''
ent b = 123;

a = _atxt(b); // carga "123" en a :  ['1','2','3','','']
_imp("valor de a: %s",a); //valor de a: 123

```

- _conc

Recibe como parámetro dos cadenas o dos arreglos de caracteres y concatena el valor de los dos parámetros y los almacena en el primer parámetro.

```
chr a[5] = "123"; //a :  ['1','2','3','',''] 
chr b[] = "si"; // b :  ['s','i']
 
_conc(a,b); // concatena los dos arreglos
_imp("valor de a: %s",a); //valor de a: 123si
 
```

- _aent

Recibe como parámetro una cadena. Devuelve el valor entero de la cadena.

```
chr a[5] = "123"; //a :  ['1','2','3','',''] 
ent b; // b = 0
 
b = _aent(a); // carga el valor 123 en b
 
```

- _adec

Recibe como parámetro una cadena. Devuelve el valor decimal de la cadena.

```
chr a[5] = "123.6"; //a :  ['1','2','3','.','6'] 
dec b; // b = 0
 
b = _adec(a); // carga el valor 123.6 en b
 
```

- _eqls

Recibe como parámetro dos cadenas y devuelve un booleano que indica si las cadenas son iguales o no.

```
chr a[5] = "123.6"; //a :  ['1','2','3','.','6'] 
chr b[] = "string"; //b : ['s','t','r','i','n','g']
 
_imp("son iguales\? %b", _eqls(a,b)); // son iguales? false
 
```

Sentencias

Son los elementos básicos en los que se divide el código en un lenguaje de programación.

Asignación de variables

Una asignación de variable consiste en asignar un valor a una variable. A las variables se les asigna un valor con el signo igual, seguido de una expresión. La forma general de asignar un valor es:

```
...

chr a[5] = "123.6";
chr b[] = "string"; // si el tamaño de la dimension
/* en la declaracion no es explicita, la asignacion
   es obligatoria. Esto en el caso de los arreglos o
   arreglos de arreglos
*/

unavariable = 25;

undecimal = tomarDivicion(8,6.0);

...
```

Consideraciones:

- Si se intenta asignar un valor a una variable que no ha sido definida se debería de reportar el error.
- Se debe verificar que el tipo que se está asignado sea el mismo con el que fue definida la variable

Sentencias de transferencia

RMB soportará dos sentencias de salto o transferencia: romper y siga. Estas sentencias transferirán el control a otras partes del programa.

- romper

La sentencia romper sirve para terminar la ejecución de un ciclo.

```
ent I = 0;

while (I < 25){
    _imp("imprimiendo en consola %e", I);
    if (I > 8){
        _imp("entrando al if");
        rompe;
}
    I++;
}
```

Consideraciones:

- Deberá verificarse que la sentencia romper aparezca dentro de un ciclo.
- La sentencia romper únicamente detendrá la ejecución del bloque de

sentencias asociado a la sentencia cíclica que la contiene.

- siga

La sentencia siga se utilizará para transferir el control al principio del ciclo, es decir, continuar con la siguiente iteración.

```
ent I = 0;

while (I < 25){
    _imp("imprimiendo en consola %e", I);
    if (I == 11){
        I = I+2;
        _imp("entrando al if");
        siga;
}
    I++;
}

```

.

Consideraciones:

- Deberá verificarse que la sentencia siga aparezca dentro de un ciclo y afecte solamente las iteraciones de dicho ciclo.

Sentencias de selección

- Sentencia If

Esta sentencia de selección bifurcará el flujo de ejecución ya que proporciona control sobre dos alternativas basadas en el valor lógico de una expresión (condición).

```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
...
else {
   // statement(s)
}

```

- Switch

Esta sentencia de selección sirve para bifurcación múltiple, es decir, proporcionará control sobre múltiples alternativas basadas en el valor de una expresión de control.

La sentencia switch compara con el valor seguido de cada case, y si coincide, se ejecuta el bloque de sentencias asociadas a dicho case, y posteriormente finaliza su ejecución.

```
switch (expression)
{
    case constant1:
      // statements
      romper;
    case constant2:
      // statements
      romper;
    .
    .
    .
    default:
      // default statements
}
```

Consideraciones:

- Si i es el caso actual y n el número de casos, si la expresión del caso i llegara a coincidir con la expresión de control (para $i < n$), se ejecuta el bloque i correspondiente y la evaluación termina.
- Si entra en un caso y este no cuenta con un romper, se debe continuar ejecutando los demás casos hasta encontrar esta instrucción.
- Tanto la expresión de control como las expresiones asociadas a cada uno de los casos deberá ser de tipo primitivo.
- Si ninguno de los casos coincide con la expresión de control se ejecutará el bloque de sentencias asociadas a default.

Ciclos

While

La sentencia cíclica mientras se utilizará para crear repeticiones de sentencias en el flujo del programa.

El bloque de sentencias se ejecutará mientras la condición sea verdadera (0 a n $veces$), de lo contrario el programa continuará con su flujo de ejecución normal.

```
ent digito=0;

while (digito<=9) {
    _imp("%d ",digito);
    digito++;
}
```

Repeat

La sentencia cíclica repeat se utilizará para crear repeticiones de sentencias en el flujo del programa.

El bloque de sentencias se ejecutará una vez y se seguirá ejecutando mientras la condición sea verdadera (1 $a\ n\ veces$), de lo contrario el programa continuará con su flujo de ejecución normal.

```
ent digito=0;

repeat {
    _imp("%d ",digito);
    digito++;
} when (digito<=9) ;
```

For

La sentencia cíclica para permitirá inicializar o establecer una variable como variable de control, el ciclo tendrá una condición que se verificará en cada iteración, luego se deberá definir una operación que actualice la variable de control cada vez que se ejecuta un ciclo para luego verificar si la condición se cumple.

```
ent y;

for(ent x=2;x<20;x=x+2){ // o x++ o x--

    _imp("El contador X vale: %d\n",x);

}
```

Consideraciones:

- Siempre se debe declarar una variable entera dentro del for para su funcionamiento.

Sentencias de entrada/salida

- _imp

Instrucción que permitirá imprimir una expresión en la consola del editor en línea.

```
ent x;
chr y[] = "viernes";

_imp("El contador X vale: %d\n",x);

_imp("texto comun y corriente");


_imp("la fecha es %d/%d/%d dia: %s",1,12,2019,y);
```

Formato	Valor esperado
%d	decimal
%e	entero
%b	booleano
%c	carácter
%s	Cadena (caracteres [])

Consideraciones:

- La impresión en consola no permite el paso de variables Rstring, si se desea imprimir este valor se debe castear a un arreglo de caracteres.

- _write

Instrucción que permitirá escribir un archivo directamente en el disco duro.

```
_write("/midir/hola_mundo.txt");
_wf("hola");
_wf("con formato %d",9.758);
_close();
```

Consideraciones:

- La impresión en un archivo no permite el paso de variables Rstring, si se desea imprimir este valor se debe castear a un arreglo de caracteres.
- Si el archivo no existe, crea el archivo y escribe dentro del mismo.
- Si el archivo ya existe, sobrescribe dentro del mismo. Esto quiere decir que se perderá todo lo que pudiera contener este archivo.
- Solo se puede escribir en un archivo a la vez.

- _apend

Instrucción que permitirá escribir un archivo directamente en el disco duro.

```
1  _apend("/midir/hola_mundo.txt");
2  _wf("hola");
3  _wf("con formato %d",9.758);
4  _close();
5
```

Consideraciones:

- La impresión en un archivo no permite el paso de variables Rstring, si se desea imprimir este valor se debe castear a un arreglo de caracteres.
- Si el archivo no existe, no crea el archivo y despliega un mensaje de error en tiempo de ejecución.
- Si el archivo ya existe, busca la última línea disponible dentro del mismo y continua escribiendo a partir de esta. Esto quiere decir que no se perderá todo lo que pudiera contener este archivo.
- Solo se puede escribir en un archivo a la vez.

- _wf

Instrucción encargada de recibir el texto que será impreso dentro del archivo mientras no se utilice la sentencia _close.

- _close

Instrucción encargada de cerrar el archivo que está activo en cierto punto de la ejecución. Mientas un archivo se encuentre abierto no se puede abrir otro.

- _read

Instrucción que permitirá leer un archivo que se encuentre en el disco duro. Almacenando su contenido en una variable de tipo Rstring;

```
1  Rstring contenedor;
2  _apend("/midir/hola_mundo.txt", contenedor);
3  _close();
4
```

Consideraciones:

- Si el archivo no existe, no crea el archivo y despliega un mensaje de error en tiempo de ejecución.
- La instrucción _close va inmediatamente después de _read.

GUI

El lenguaje RMB permite la generación, manejo e interacción con formularios (ventanas) gráficos para el usuario. Estos formularios son utilizados construidos en tiempo de ejecución y su funcionamiento es síncrono, esto quiere decir que mientras un formulario este abierto no se ejecutara las instrucciones que le sigan a la llamada del formulario.

Los formularios en RMB pueden contener los siguientes componentes:

- Etiqutas
- Inputs
  - Texto
  - Área
  - Numérico
  - Contraseña
- butones

Rlbl

Este componente permite el despliegue de un texto no editable dentro de un formulario.

Input

Este componente permite el despliegue de cuadros de texto no editables dentro de un formulario. Este componente se puede encontrar de distintos formatos:

- Rtxt

Este input permite la entrada de cualquier texto lineal.

- RtxtA

Este input permite la entrada de cualquier texto en formato de múltiples líneas.

- RtxtP

  Este input permite la entrada de cualquier texto lineal con una máscara de seguridad, utilizado para ingreso de contraseñas.

- RtxtN
  Este input permite la entrada de cualquier número únicamente, en formato lineal.

Rbton
Este componente permite el despliegue de botones no editables pero funcionales dentro de un formulario.

Rmensage
RMB GUI cuenta con un ventanas de alerta que permite el despliegue de mensajes en forma gráfica.

Sentencias GUI

RMB reconoce que se trata de una llamada a un formulario grafico gracias a la extensión del archivo que contiene el código. Los archivos encargados de contener la configuración grafica del formulario son los “*.B”.

Eventos
Dentro de este tipo de archivo puede venir cualquier tipo de sentencias RMB y además se pueden configurar los eventos que puede tener el formulario.

Los eventos soportados por RMB GUI son 2:
- Inicar_ventada
- Al_dar_click

- R:iniciar_ventana

Este es uno de los eventos que sirven para cargar y configurar todos los componentes gráficos que se desean. Su sintaxis es la de un método sin parámetros de entrada.

- ID:iniciar_ventana

Este es uno de los eventos que sirven ejecutar código al momento de realizar un click sobre un botón predeterminado dentro del formulario.

Declaración de componente
La declaración de cualquier componente (input, etc…) se debe realizar de manera global. Para poder ser utilizado en cualquier lugar dentro del ámbito del formulario.

Asignacion de componentes
Para poder utilizar y cargar dentro del formulario cualquier componente, es

obligatorio inicializar el componente. Esto es posible gracias al método:

- _Nuevo_GUI(componente);

Métodos y Funciones propias de los componentes

Cada tipo de componente cuenta con funciones y métodos propios, adquiridos al momento de ser inicializados. Muchos de estos son compartidos entre ellos y otros son exclusivos de ciertos componentes. Los cuales son:

- _alto_y_ancho

Este es un método que recibe dos enteros como parámetros y los utiliza para cargarlo en el formulario, el primer parámetro representa el ancho del formulario y el segundo parámetro representa la altura del formulario. Este método es exclusivo para el formulario.

- Settexto

Este es un método que recibe un Rstring como parámetro y lo utiliza para cargarlo en el componente.

- Setancho

Este es un método que recibe un entero como parámetro y lo utiliza para cargarlo como ancho del componente.

- Setalto

Este es un método que recibe un entero como parámetro y lo utiliza para cargarlo como altura del componente.

- Setpos

Este es un método que recibe dos enteros como parámetros y los utiliza para cargarlo en el componente, el primer parámetro representa la posición en el eje X dentro del formulario y el segundo parámetro representa la posición en el eje y dentro del formulario.

- Gettexto

Esta es una función retorna un Rstring cargado con el texto que contiene el componente en ese momento.

- Getancho

Esta es una función retorna un entero cargado con el tamaño del ancho que contiene el componente en ese momento.

- Getalto

Esta es una función retorna un entero cargado con el tamaño de la altura que contiene el componente en ese momento.

- Getpos
  Esta es una función retorna un arreglo de enteros cargado con la posición actual del componente en ese momento.

Ejemplo de archivo con extensión “B”

```
#importar "/codigo/estructura.m"
#definir hola 25

Rtxt miInput; // un cuadro de texto
Rtxt miInput2; // un cuadro de texto
Rbton miBoton; //un boton
Rbton miBoton2; //un boton

zro R:iniciar_ventana(){

    _Nuevo_GUI(miInput); // se inicializa la variable miInput
    _Nuevo_GUI(miBoton); // se inicializa la variable miBoton
    _Nuevo_GUI(miInput2); // se inicializa la variable miInput
    _Nuevo_GUI(miBoton2); // se inicializa la variable miBoton

    /*
    * Si un componente no se ha inicializado no se puede acceder a sus
    * funciones y metodos propios.
    */

    miInput.settexto("este es un texto inicial");
    miInput.setancho(hola);
    miInput.setpos(hola,hola);

    miBoton.settexto("Mensaje");
    miBoton.setancho(hola);
    miBoton.setalto(hola*2);
    miBoton.setpos(hola+100,hola+200);// pos en X y Y dentro de la ventana.
```

```
30      miInput2.setancho(hola);
31      miInput2.setpos(hola,hola+40);
32
33      miBoton2.settexto("calcular");
34      miBoton2.setancho(hola);
35      miBoton2.setalto(hola*2);
36      miBoton2.setpos(hola+160,hola+200);// pos en X y Y dentro de la ventana.
37
38      //la siguiente funcion puede venir o no
39      _alto_y_ancho(600,700);
40  }
41
42  zro miBoton:al_dar_click(){
43
44      _imp("texto con formato %e ", hola); // imprime en consola 25
45
46      Rstring temporal = miInput.gettexto();
47
48      chr casteo[_peso_de(temporal)];
49      casteo = _atxt(temporal);
50      _imp("texto con formato 2 %s ", casteo); // imprime en consola 'este es un texto inicial'
51      _imp("texto con formato 2 %c ", casteo[0]); // imprime en consola 'e'
52
53      Rmensaje("valor de miInput: %s", casteo); //un cuadro de mensaje
54  }
```

```
56  zro miBoton2:al_dar_click(){
57
58      miInput2.settexto(miInput.gettexto()); // copia valor de un cuadro a otro
59
60      _imp("texto con formato %e ", hola); // imprime en consola 25
61
62      ent miSalida = miFactorial(1*1+2*0+5); //llamada a funcion importada '...estructura.h'
63
64      Rmensaje("valor de miInput: %d", miSalida); //un cuadro de mensaje
65  }
66
```

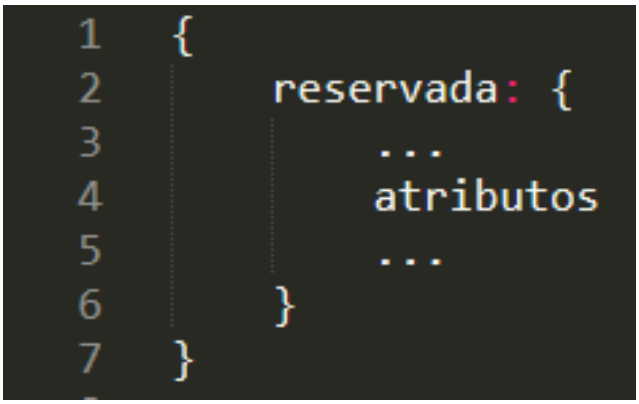
Archivo de configuración

El IDE de RMB permite generar un archivo que contiene toda la estructura de un proyecto desarrollado en el lenguaje RMB.
Este archivo tiene una configuración parecida a un archivo json. Lo cual nos permite representar de manera ordenada la jerarquía del proyecto desarrollado.
En resumen el archivo de configuración contendrá meta data del proyecto.

Estos archivos tendrán una extensión ".olc", el nombre del archivo debe ser el mismo que el nombre del proyecto.

Sintaxis olc

El archivo olc se compone de objetos, descritos de la forma:



Los atributos dentro de un objeto pueden ser simples o también puede contener objetos en su interior.

Los principales objetos utilizados son:

- Proyecto
- Archivo
- Carpeta

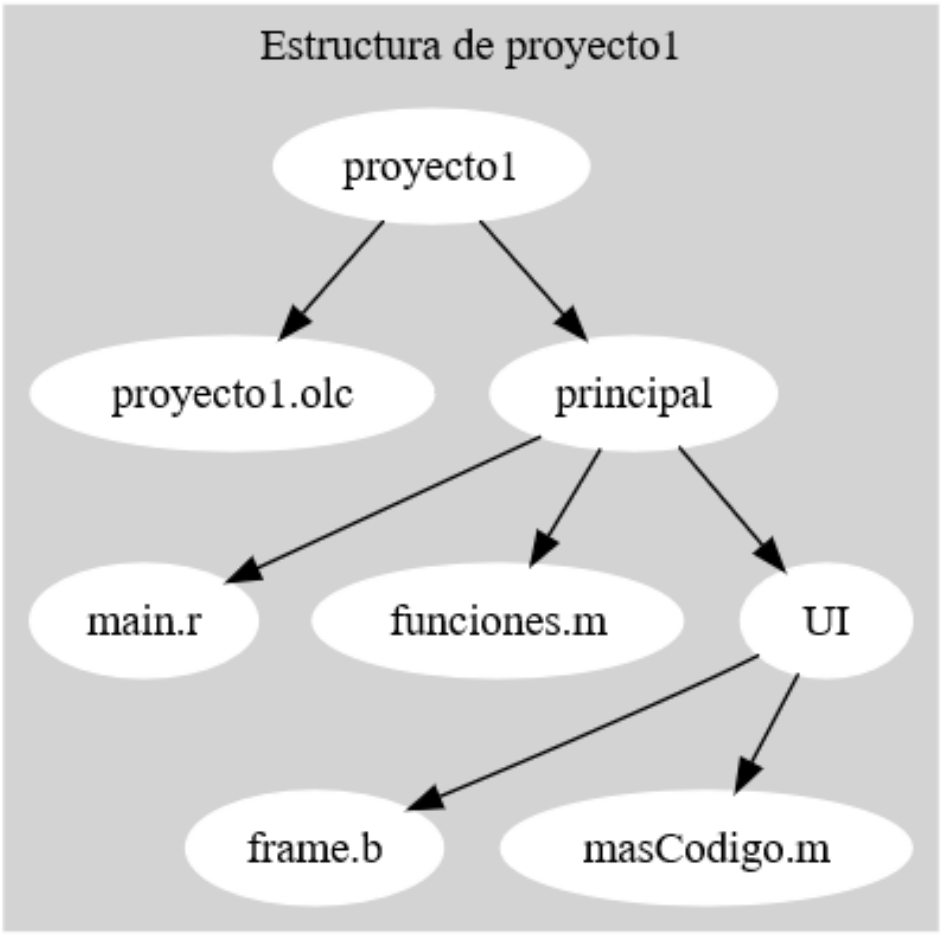
Los principales atributos utilizados son:

- Ruta : contiene la ruta absoluta actual del proyecto
- Nombre: contiene el nombre del objeto.
- Correr: contiene el archivo "R" que se correrá de manera automática.
- Configuración: contiene la estructura general del proyecto tanto en archivos como en carpetas.
- Fecha de modificación: contiene la última fecha de modificación, también la hora en la que realizo dicha modificación.

Ejemplo de un archivo de configuración.

```
{
    proyecto:{
        ruta:"/documentos/Entrada1/",
        nombre:"proyecto1",
        correr:"principal/main.r",
        configuracion:{
            archivo:{
                nombre:"proyecto1.olc",
                fecha_mod:"dd/mm/yyyy hh:mm:ss"
            },
            carpeta:{
                nombre:"principal",
                archivo:{
                    nombre:"main.r",
                    fecha_mod:"dd/mm/yyyy hh:mm:ss"
                },
                archivo:{
                    nombre:"funciones.m",
                    fecha_mod:"dd/mm/yyyy hh:mm:ss"
                },
                carpeta:{
                    nombre:"UI",
                    archivo:{
                        nombre:"frame.b",
                        fecha_mod:"dd/mm/yyyy hh:mm:ss"
                    },
                    archivo:{
                        nombre:"masCodigo.m",
                        fecha_mod:"dd/mm/yyyy hh:mm:ss"
                    }
                }
            }
        }
    }
}
```

Estructura obtenida del archivo de configuración:



- Si el archivo de configuración no existe, IDE RMB debe poder generar el archivo de configuración.
- IDE RMB debe poder actualizar el archivo de configuración en todo momento.

Entregables y Restricciones

Entregables

- Código fuente
- Aplicación funcional
- Gramáticas
- Manual técnico
- Manual de usuario

Deberán entregarse todos los archivos necesarios para la ejecución de la aplicación, así como el código fuente, la gramática y la documentación. La calificación del proyecto se realizará con los archivos entregados en la fecha establecida. El usuario es completa y únicamente responsable de verificar el contenido de los entregables. La calificación se realizará sobre archivos ejecutables. Se proporcionarán archivos de entrada al momento de calificación.

Restricciones

La aplicación deberá ser desarrollada utilizando el lenguaje Java haciendo uso de flex o jflex y cup para la generación de analizadores léxicos y sintácticos.

Copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ingeniería en Ciencias y Sistemas los involucrados.

Consideraciones

El auxiliar podrá realizar cambios al enunciado de forma oficial en un documento de cambios.

El estudiante deberá realizar lo que se exprese en este documento considerando los cambios que estén plasmados en el documento de cambios.

Durante la calificación se realizarán preguntas sobre el código para verificar la autoría del mismo, de no responder correctamente la mayoría de las preguntas se reportará la copia.

Requisitos mínimos

Los requerimientos mínimos del proyecto son funcionalidades del sistema que permitirán un ciclo de ejecución básica, para tener derecho a calificación se deben cumplir con lo siguiente:

- Descripción
- Descripción de la aplicación
- DosIDE
- Consola

- Reportes
- Generalidades del lenguaje RMB
- Tipos de dato
- Tipos primitivos
- Tipos definidos por el usuario
- Arreglos
- Arreglos de arreglos
- Acceso a los arreglos
- Registros
- Memoria Virtual
- _reservar
- _pesode
- Definición léxica
- Comentarios
- Sensibilidad a mayúsculas y minúsculas
- Identificadores
- Palabras reservadas
- Literales
- Secuencias de escape
- Estructura del Lenguaje
- Declaración de variables
- Declaración de constantes
- Expresiones
- Funciones y Metodos
- Funciones
- Metodos
- Paso de Parámetros
- Sentencias
- Asignación de variables
- Sentencias de transferencia
- Romper
- Siga
- Sentencias de selección
- Sentencia If
- Ciclos
- Sentencia Write
- Entorno de ejecución
- Estructuras del entorno de ejecución
- Acceso a estructuras del entorno de ejecución

- RMB GUI
- Archivo de configuracion
- Manejo de errores
- Tipos de errores
- Contenido mínimo de tabla de errores
- Precedencia y asociatividad de operadores

Entrega del proyecto

- La entrega será virtual.
- La entrega de cada uno de los proyectos es individual.
- Para la entrega del proyecto se deberá cumplir con todos los requerimientos mínimos.
- No se recibirán proyectos después de la fecha ni hora de la entrega estipulada.
- La entrega del proyecto será mediante un archivo comprimido de extensión zip, el formato: <carnet>.zip.
- Entrega del proyecto:
    - Viernes 20 de diciembre a las 23:59 horas