UNIVERSIDAD DE LA EMPRESA

CalcuSimple

Taller 1

Trabajo de Laboratorio

Curso nocturno 2017/18

Licenciatura en Informática

Tutor:	**Gómez, Federico.**	
Integrantes:	**Pías, Richard**	**- 1.924.591-2**
	Segovia, Joaquín	**- 4.739.544-4**
	Torres, Mathias	**- 4.223.291-4**

Tabla de contenido

1) Definición de estructuras _____ 3

2) Diagrama de jerarquía de módulos _____ 6

3) Pseudocódigo detallado para los tres comandos _____ 7

4) Cabezales de procedimientos y funciones _____ 8

5) Cronograma y planificación _____ 16

6) Decisiones y consideraciones del diseño de la solución _____ 18

1) Definición de estructuras

Para este trabajo de laboratorio, una de las tareas principales es el diseño de las estructuras de datos necesarios para representar el problema.

A continuación, se muestra tales estructuras, escritas en el lenguaje de programación C++, de todos los tipos de datos correspondientes, ordenados alfabéticamente por nombre del archivo, ya que en el siguiente punto, se podrá ver las inclusiones mediante el diagrama.

```
1 typedef struct nodoVariableAlias
2 {
3     variable info;
4     nodoVariableAlias * Hizq;
5     nodoVariableAlias * Hder;
6 } nodoVariable;
7
8 typedef nodoVariable *arbolVariables;
```

```
1 typedef struct
2 {
3     enumOpsArits func;
4     int valor1;
5     int valor2;
6 } AS3;
```

```
1 typedef struct
2 {
3     enumOpsArits func;
4     int valor1;
5     StringDyn nomVariable;
6 } AS4;
```

```
1 typedef struct
2 {
3     enumOpsArits func;
4     StringDyn nomVariable;
5     int valor1;
6 } AS5;
```

```
1 typedef struct
2 {
3     enumOpsArits func;
4     StringDyn nomVariable1;
5     StringDyn nomVariable2;
6 } AS6;
```

```
1 typedef enum{FALSE,TRUE}boolean;
```

```
typedef enum{SUM, RES, MUL, DIV}enumOpsArits;
```

```
typedef enum {LEER, MOSTRAR, enum_AS1, enum_AS2, enum_AS3, enum_AS4, enum_AS5, enum_AS6} enumOpsBasicas;
```

```
typedef struct
{
    StringDyn nombreVar;
    enumOpsBasicas tipoInstruccion;
    union
    {
        int numeroEntero;
        StringDyn nomVar1;
        AS3 funcIntInt;
        AS4 funcIntVar;
        AS5 FuncVarInt;
        AS6 FuncVarVar;
    } discInstruccion;
} instruccion;
```

```
typedef struct nodoInstruccionAlias
{
    instruccion info;
    nodoInstruccionAlias * sig;
} nodoInstruccion;

typedef nodoInstruccion * listaInstrucciones;
```

```
typedef struct nodoListaStringAlias
{
    StringDyn info;
    nodoListaStringAlias * sig;
} nodoListaString;

typedef nodoListaString * listaStrings;
```

```
1 const int MAX = 80;
2 typedef char * StringDyn;
```

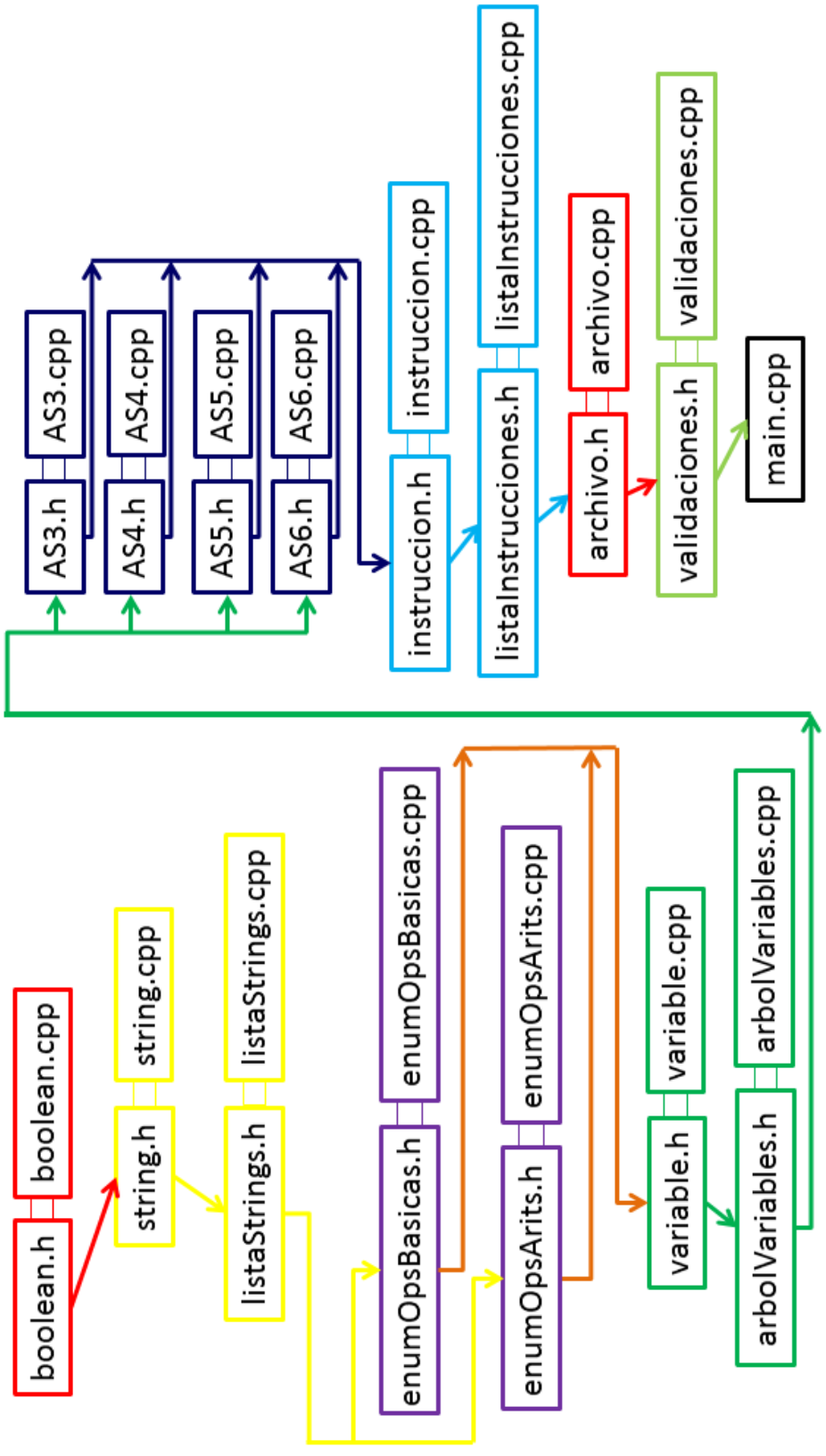
```
1 typedef struct
2 {
3     StringDyn nombre;
4     int valor;
5 } variable;
```

En el siguiente punto, veremos como estas estructuras, en sus respectivos pares de archivos (.h y .cpp), se relacionan con inclusiones. Para ello, se utiliza el diagrama de jerarquía donde la inclusión queda representada en forma de flecha.

Cabe destacar, que se verán además, módulos como el de validaciones y archivos, los cuales son utilizados para separar el trabajo a implementar, y poder así, organizar aún mejor el código.

Por último, podremos encontrarnos con el main.cpp, mediante el cual, el software iniciará.

2) Diagrama de jerarquía de módulos



3) Pseudocódigo detallado para los tres comandos

Se adjuntará al final de este documento, el pseudocódigo pertinente utilizado como base fundamental para el diseño y cubrimiento de las posibilidades, para la creación del main de nuestro programa, el cual contiene los tres comandos principales (*compilar, ejecutar, salir*).

Para su mejor lectura, se desplegara como anexo a este documento, ya que se presentara en forma horizontal y con sus números de líneas correspondientes.

4) Cabezales de procedimientos y funciones

Se exponen en esta sección, los cabezales que se encontraran en los archivos con extensión .h.

Análogamente al punto 1 de este documento, se presentan en forma alfabética al nombre de archivo.

```
//arbolVariables.h

//crea el abb en null, como modifica el mismo abb hay que pasarlo por referencia
void arbolVariablesPROC_crearABB(arbolVariables &abb);//1

void arbolVariablesPROC_destruirABB(arbolVariables &abb);//2

boolean arbolVariablesFUNC_isArbolVacio(arbolVariables av);//3

void arbolVariablesPROC_agregarVariableAlArbol(arbolVariables &av, variable v);//4

void arbolVariablesPROC_modificarVariableEnABB(arbolVariables &abb, StringDyn nombreVar, int nuevoValor);//5

boolean arbolVariablesFUNC_verificarExistenciaString(arbolVariables av, StringDyn str);//6

variable arbolVariablesFUNC_darRaiz(arbolVariables abb);//7

arbolVariables arbolVariablesFUNC_darHizq(arbolVariables abb);//8

arbolVariables arbolVariablesFUNC_darHder(arbolVariables abb);//9

void arbolVariablesPROC_PrintNombreValor(arbolVariables abb, StringDyn nombreVar);//10

int arbolVariablesFUNC_valorDeVariable(arbolVariables abb, StringDyn nombreVar);//11

void arbolVariablesPROC_obtenerListaDeNombres(arbolVariables abb, listaStrings &variablesUSADAS);//12
```

```
//archivo.h

// Vetrifica la Existencia de un archivo pasado por parametro
boolean archivoFUNC_existeArchivo(StringDyn strNombreArchivo_con_extension);//1

// Crea un Archivo, recibe el nombre del archivo con extension incluida
void archivoPROC_crearArchivo(FILE * &archivo, StringDyn nombreArchivo);//2

// Elimina un Archivo, recibe el nombre del archivo con extension incluida
void archivoPROC_borrarArchivo(StringDyn nombreArchivo);//3

// Abre el archivo y lo deja abierto
void archivoPROC_abrirArchivo(FILE * &archivo, StringDyn nombreArchivo, StringDyn como);//4

// Cierra el archivo
void archivoPROC_cerrarArchivo(FILE * &archivo);//5

// Verifica si un archivo esta vacío
boolean archivoFUNC_isArchivoVacio(StringDyn nombreArchivo_con_extension);//6

// si es EOF devolver linea = NULL
StringDyn archivoFUNC_cargarLaSiguienteLinea(FILE * archivo);//7

// Baja el arbol de variables al archivo
void archivoPROC_bajarABB_de_Variables_al_Archivo(arbolVariables abb, FILE * archivo);//8

// Levanta el arbol de variables a memoria
void archivoPROC_levantoABB_de_Variables_a_Memoria(arbolVariables &abb,FILE * archivo);//9

// Baja la estructura de variable al archivo
void archivoPROC_bajarVariable(variable v, FILE * archivo);//10

// Levanta la estructura de variable
void archivoPROC_levantarVariable(variable &v, FILE * archivo);//11

// Baja la lista de instrucciones al archivo
void archivoPROC_listaInstrucciones_BajarLista_al_Archivo(listaInstrucciones li, FILE * archivo);//12

// Levanta la lista de instrucciones a memoria
void archivoPROC_listaInstrucciones_LevantarLista_a_Memoria(listaInstrucciones &li, FILE * archivo);//13

// Bajar la estructura Instrucciones al archivo
void archivoPROC_listaInstrucciones_bajarInstruccion(instruccion i, FILE * archivo);//14

// Levantar la estructura Instrucciones al archivo
void archivoPROC_listaInstrucciones_levantarInstruccion(instruccion &i, FILE * archivo);//15

// Baja la estructura AS3 al archivo
void archivoPROC_listaInstrucciones_bajarAS3(AS3 as3, FILE * archivo);//16

// Levanta la estructura AS3
void archivoPROC_listaInstrucciones_levantarAS3(AS3 &as3, FILE * archivo);//17

// Baja la estructura AS4 al archivo
void archivoPROC_listaInstrucciones_bajarAS4(AS4 as4, FILE * archivo);//18

// Levanta la estructura AS4
void archivoPROC_listaInstrucciones_levantarAS4(AS4 &as4, FILE * archivo);//19

// Baja la estructura AS5 al archivo
void archivoPROC_listaInstrucciones_bajarAS5(AS5 as5, FILE * archivo);//20

// Levanta la estructura AS5
void archivoPROC_listaInstrucciones_levantarAS5(AS5 &as5, FILE * archivo);//21

// Baja la estructura AS6 al archivo
void archivoPROC_listaInstrucciones_bajarAS6(AS6 as6, FILE * archivo);//22

// Levanta la estructura AS6
void archivoPROC_listaInstrucciones_levantarAS6(AS6 &as6, FILE * archivo);//23

// Baja la estructura Operaciones Basicas al archivo
void archivoPROC_bajarEnumOpsBasicas(enumOpsBasicas ob, FILE * archivo);//24

// Levanta la estructura Operaciones Basicas
void archivoPROC_levantarEnumOpsBasicas(enumOpsBasicas &ob, FILE * archivo);//25

void archivoPROC_bajarEnumArits(enumOpsArits oa, FILE * archivo);//26

void archivoPROC_levantarEnumArits(enumOpsArits &oa, FILE * archivo);//27

void archivoPROC_bajarString(StringDyn s, FILE * archivo);//30

void archivoPROC_levantarString(StringDyn &s, FILE * archivo);//31

void archivoPROC_bajarBoolean(boolean b, FILE * archivo);//32

void archivoPROC_levantarBoolean(boolean &b, FILE * archivo);//33

void archivoPROC_guardarLog(StringDyn log);//34

void archivoPROC_agregarSaltoLineaAlFinal(StringDyn nombreConcatenado);//35
```

```h
//AS3.h

void AS3_FUNC_carga(AS3 &asignacion, enumOpsArits func, int valor1, int valor2);//1

enumOpsArits AS3_FUNC_darTipoFUNC(AS3 as3);//2

int AS3_FUNC_darValor1(AS3 as3);//3

int AS3_FUNC_darValor2(AS3 as3);//4
```

```h
//AS4.h

void AS4_FUNC_carga(AS4 &as, enumOpsArits func, int valor1, StringDyn nomVariable);//1

enumOpsArits AS4_FUNC_darTipoFUNC(AS4 as4);//2

int AS4_FUNC_darValor1(AS4 as4);//3

void AS4_PROC_darNombreVariable(AS4 as4, StringDyn &nombreVar);//4

StringDyn AS4_FUNC_darNombreVariable(AS4 as4IN);//5
```

```h
//AS5.h

void AS5_FUNC_carga(AS5 &as, enumOpsArits func, StringDyn nomVariable, int valor1);//1

enumOpsArits AS5_FUNC_darTipoFUNC(AS5 as5);//2

void AS5_PROC_darNombreVariable(AS5 as5, StringDyn &nombreVar);//3

int AS5_FUNC_darValor1(AS5 as5);//4

StringDyn AS5_FUNC_darNombreVariable(AS5 as5IN);//5
```

```h
//AS6.h

void AS6_FUNC_carga(AS6 &as, enumOpsArits func, StringDyn nomVariable1, StringDyn nomVariable2);//1

enumOpsArits AS6_FUNC_darTipoFUNC(AS6 as6);//2

void AS6_PROC_darNombreVariable1(AS6 as6, StringDyn &nombreVar1);//3

void AS6_PROC_darNombreVariable2(AS6 as6, StringDyn &nombreVar2);//4

StringDyn AS6_FUNC_darNombreVariable1(AS6 as6IN);//5

StringDyn AS6_FUNC_darNombreVariable2(AS6 as6IN);//6
```

```
1 //boolean.h
2
3 // Carga de un booleano por teclado
4 void booleanPROC_carga(boolean &booleanoAcargar);//1
5
6 // Despliegue de un booleano por pantalla
7 void booleanPROC_mostrar(boolean booleanoAcargar);//2
```

```
1 //enumOpsArits.h
2
3 // Carga de una FUNC por comparacion de strDyn: recibe el tercer substring y retorna una variable FUNC con el tipo de FUNC para
  guardarla en el struct (del AS3 al AS6) dentro de la union
4 // precondicion, debe validarse con StringDyn_equalAnyFUNCarits que sea una de las 4 funcs
5 void enumOpsAritsFUNC_carga(enumOpsArits &oa, StringDyn strDynFUNC_IN);//1
```

```
1 //enumOpsBasicas.h
2
3 typedef enum {LEER, MOSTRAR, enum_AS1, enum_AS2, enum_AS3, enum_AS4, enum_AS5, enum_AS6} enumOpsBasicas;
4 //asignacion se divide en 6    //--> 8 ops basica
5
6 // Carga de una enumOpsBasicas por numero del 0 al 7: recibe un numero codigo 0-7 y retorna una variable enumOpsBasicas con el
  tipo de operacion para guardar lo correspondiente segunda, incluyendo a la union, si es del AS1 al AS6
7 void enumOpsBasicasFUNC_carga(enumOpsBasicas &ob, int codigoCeroSieteIN);//1
```

```
1 //instruccion.h
2
3 void instruccionFUNC_cargar(instruccion &i, StringDyn nombreVar, enumOpsBasicas tipoInstruccion, int numeroEntero, StringDyn
  nomVar1, AS3 AS3_funcIntInt, AS4 AS4_funcIntVar, AS5 AS5_FuncVarInt, AS6 AS6_FuncVarVar);//1
4
5 void instruccionPROC_darNombreVar0(instruccion instIN, StringDyn &nombreVar1LadoIzq);//2
6
7 enumOpsBasicas instruccionFUNC_darTipoInstruccion(instruccion instIN);//3
8
9 int instruccionFUNC_darNumeroEntero(instruccion instIN);//4
10
11 void instruccionPROC_darNombreVar1(instruccion instIN, StringDyn &nombreVar1LadoDer);//5
12
13 AS3 instruccionFUNC_darAS3_funcIntInt(instruccion instIN);//6
14
15 AS4 instruccionFUNC_darAS4_funcIntVar(instruccion instIN);//7
16
17 AS5 instruccionFUNC_darAS5_FuncVarInt(instruccion instIN);//8
18
19 AS6 instruccionFUNC_darAS6_FuncVarVar(instruccion instIN);//9
20
21 StringDyn instruccionFUNC_darNombreVar0(instruccion instIN);//10
22
23 StringDyn instruccionFUNC_darNombreVar1(instruccion instIN);//11
```

```
//listaInstrucciones.h

void listaInstruccionesPROC_crearLI(listaInstrucciones &li);//1

void listaInstruccionesPROC_destruirLI(listaInstrucciones &li);//2

boolean listaInstruccionesFUNC_isNull(listaInstrucciones li);//3

int listaInstruccionesFUNC_largo(listaInstrucciones li);//4

void listaInstruccionesPROC_instertFront(listaInstrucciones &li, instruccion info);//5

//utilizada para el armado de la estructura lista de instrucciones
void listaInstruccionesPROC_instertBack(listaInstrucciones &li, instruccion info);//6

//el contador se suma afuera y le pasa para que buclee n veces hasta que da el siguiente elemento
//util para ver todas las instrucciones en la estructura lista de instrucciones cuando se carge para usar en ejecucion
instruccion listaInstruccionesFUNC_darInstruccion(listaInstrucciones li, int numero);//8

//guarda una instrucicon a la lista con el discriminante- tipo del enumeradoopsBasi leer y el nombre de la variable
void listaInstruccionesPROC_agregarInstruccionLEER(StringDyn str, listaInstrucciones &li);//9

//guarda una instrucicon a la lista con el discriminante- tipo del enumeradoopsBasi mostrar y el nombre de la variable
void listaInstruccionesPROC_agregarInstruccionMOSTRAR(StringDyn str, listaInstrucciones &li);//10

void listaInstruccionesPROC_agregarInstruccionAScompuesta(arbolVariables abb, listaInstrucciones &li, listaStrings ls, int
nroLinea, boolean &huboError, int &nroLineaWarning, boolean &compiloConCero);//11
//guarda una instrucicon a la lista analizando cual de las combinaciones de 5 elementos es para guardarla con su respectivo
structura datos de la union y discriminante
//de existir el error, tirarlo en el formato : printf("\n***Error de compilaci%sn - linea %d: para la instruccion de la
asignacion con FUNC con 2 argumentos",nroLinea); //tilde

//   Guardo las intruccion en la lista de instrucciones, le paso la lista de substring y se debera validar correctamente todas
las combinaciones, tanto LEER, MOSTRAR y asiganciones, dadas en el diseño.
//Utilizando la estructura y haciendo énfasis en la union para las combinaciones en la asignacion.
//tiene q retornar una bool para ver si hubo algun error, asi puedo saber si bajo todo o no

//12
void listaInstruccionesPROC_agregarInstruccionASsimpple(arbolVariables abb, listaInstrucciones &li, listaStrings ls, int
nroLinea, boolean &huboError);//12
```

```
//listaStrings.h

void listaStringsPROC_crearLista(listaStrings &ls);//1

void listaStringsPROC_destruirLista(listaStrings &ls);//2

int listaStringsFUNC_largo(listaStrings ls);//3

boolean listaStringsFUNC_isNull(listaStrings ls);//4 //o FUNC_isVacia

void listaStringsPROC_instertFront(listaStrings &ls, StringDyn info);//5

//utilizada para la particion de substrings
void listaStringsPROC_instertBack(listaStrings &ls, StringDyn info);//6

boolean listaStringsFUNC_isNodoPertenece(listaStrings ls, StringDyn info);//7

//precondicion: se sabe que tiene 5 justo
//estos estan para que devuelva el string directo como info

StringDyn listaStringsFUNC_darPRIMERstr(listaStrings ls);//8

StringDyn listaStringsFUNC_darSEGUNDOstr(listaStrings ls);//9

StringDyn listaStringsFUNC_darTERCERstr(listaStrings ls);//10

StringDyn listaStringsFUNC_darCUARTOstr(listaStrings ls);//11

StringDyn listaStringsFUNC_darQUINTOstr(listaStrings ls);//12

//el contador se suma afuera y le pasa para que buclee n veces hasta que da el siguiente elemento
//util para ver todas las variables en la seccion variables
//precondicion: no debe estar vacia
StringDyn listaStringsFUNC_darSIGUIENTEstr(listaStrings ls, int numero);//13

void PROC_partirStrEnSubStrs(StringDyn linea, listaStrings &lsSubsString);//14

//PRECONDICIONES PARA LOS SIGUIENTES 3 es que no este vacia

listaStrings listaStringsFUNC_listaSinPrimerElemento(listaStrings lsCompilacion);//15

void listaStringsPROC_resto(listaStrings &ls);//16

void listaStringsPROC_obtenerUltimo(listaStrings ls, StringDyn &ultimoINFO);//17

int listaStringsFUNC_ocurrenciasDeInfo(listaStrings ls, StringDyn info);//18

void listaStringsPROC_print(listaStrings ls);//19
```

```
//String.h

void StringDyn_crear(StringDyn &s);//1
/* crea un string vacío */

void StringDyn_crearDadoLargo(StringDyn &s, int largo);//2

void StringDyn_destruir(StringDyn &s);//3
/* libera la memoria usada por el string */

int StringDyn_largo(StringDyn s);//4
/* devuelve el largo del string s */

void StringDyn_copiar(StringDyn &s1, StringDyn s2);//5
/* copia el contenido del string s2 en s1 */

void StringDyn_scan(StringDyn &s);//6
/* lee el string s desde teclado */

void StringDyn_concatenar(StringDyn &s1, StringDyn s2);//7
/* concatena el contenido de s2 al final de s1 */

void StringDyn_swap(StringDyn &s1, StringDyn &s2);//8
/* intercambia los contenidos de s1 y s2 */

void StringDyn_print(StringDyn s);//9
/* imprime el string s por pantalla */

boolean StringDyn_menor(StringDyn s1, StringDyn s2);//10
//es en abc, testear porque se usara para las id del ABB
/* determina si s1 es alfabéticamente menor que s2 */

boolean StringDyn_equal(StringDyn s1, StringDyn s2);//11
/* determina si los strings s1 y s2 son iguales */

boolean StringDyn_contiene_MAYUSCULAS(StringDyn s1);//12
/* determina si un strings s1 contiene letras MAYUSCULAS */

boolean StringDyn_contiene_minusculas(StringDyn s1);//13
/* determina si un strings s1 contiene letras minusculas */

boolean StringDyn_contiene_numeros(StringDyn s1);//14
/* determina si un strings s1 contiene numeros */

int StringDyn_cantidad_de_SubString(StringDyn s1);//15
/* devuelve la cantidad de subString de una cadena */

//valida que el s1 sea el signo =
boolean StringDyn_equalSignoIgual(StringDyn s1);//16

//valida que el s1 sea igual a una de las funcs, SUM DIV RES MUL
boolean StringDyn_equalAnyFUNCarits(StringDyn s1);//17

//valida que el s1 sea igual a un numero entero
boolean StringDyn_equalNumeroEntero(StringDyn s1);//18

//19
void StringDynPROC_iniStrDynVarsCompilar(StringDyn &PROGRAMA, StringDyn &VARIABLES, StringDyn &INSTRUCCIONES, StringDyn &LEER,
StringDyn &MOSTRAR);//19

//20
void StringDynPROC_iniStrDynVarsMain(StringDyn &salir, StringDyn &compilar, StringDyn &ejecutar);//20

//21
void StringDynPROC_iniStrDynVarsOpsArits(StringDyn &SUM, StringDyn &RES, StringDyn &MUL, StringDyn &DIV);//21

//Devuelve Cargado en un StringDyn las exttensiones de los archivos .csim .vars .inst
//22
void StringDynPROC_iniStrDynExtensionesArchivos(StringDyn &ext_csim, StringDyn &ext_vars, StringDyn &ext_inst);//22

void StringDynPROC_separarPrimerPalabra(StringDyn linea, StringDyn &palabra, StringDyn &restoLinea);//23

//Funcion que compara un string con Ascii para determinar si el string = SUM
boolean StringDyn_equalSUM(StringDyn s1);//24

//Funcion que compara un string con Ascii para determinar si el string = RES
boolean StringDyn_equalRES(StringDyn s1);//25

//Funcion que compara un string con Ascii para determinar si el string = MUL
boolean StringDyn_equalMUL(StringDyn s1);//26

//Funcion que compara un string con Ascii para determinar si el string = DIV
boolean StringDyn_equalDIV(StringDyn s1);//27

void StringDyn_copiarDadoLargo(StringDyn &s1,StringDyn s2, int largo, int inicio);//28

StringDyn StringDyn_concatenarFUNC(StringDyn s1, StringDyn s2);//29
/* concatena el contenido de las entradas y lo retorna */

int StringDynFUNC_stringToNumeric(StringDyn s1);//30

void StringDynPROC_limpiarFinalLinea(StringDyn &s1);//31

void StringDynPROC_limpiarTabsLinea(StringDyn &s1);//32

void StringDynPROC_siSoloEspaciosCaracterNulo(StringDyn &s1);//33

void StringDynPROC_quitarSimboloSignoNeg(StringDyn &s1);//34

boolean StringDyn_contiene_minusculasSolo(StringDyn s1);//35
```

```
//validaciones.h

void validacionesPROC_guardarVariablesEnABB(listaStrings ls,arbolVariables &abb,boolean &huboError);//1

void validacionesPROC_bucleInstruccionesEjecucion(arbolVariables abb, listaInstrucciones li, int &nroLinea, boolean
&huboError);//2

void PROC_compilar(StringDyn nombrePrograma);//3

void PROC_ejecutar(StringDyn nombrePrograma);//4

void PROC_LimpiarYPausa();//5

void cierrePrograma(int opcion);//6

void validacionesPROC_variablesSinUso(arbolVariables abb, listaInstrucciones li, listaStrings &listaNombresOrdenEncontradas,
boolean &hayVarsSinUso);//7

boolean validacionesFUNC_isComentario(listaStrings ls);//8
```

```
//variable.h

//carga el nombre de una variable pasada por parametro y devuelve una variable con el nombre y valor = 0
void variablePROC_carga(variable &v, StringDyn nombreVar);//1

//recibe dos variables y determina si se llaman igual
boolean variableFUNC_areIguales(variable v1, variable v2);//2

//selectoras
void variablePROC_darNombre(variable v, StringDyn &nombreVar);//3

int variableFUNC_darValor(variable v);//4

// carga una variable a partir de los parametros de entrada
void variableFUNC_PARAM_cargaVariable(variable &v, StringDyn nombreVariable, int valor);//5

StringDyn variablePROC_darNombre_PARAMETRO(variable v);//6

 //significa que v1 es menor que v2
boolean variableFUNC_MenorQue(variable v1, variable v2);//7

void variablePROC_modificarValor(variable &v, int newValor);//8
```

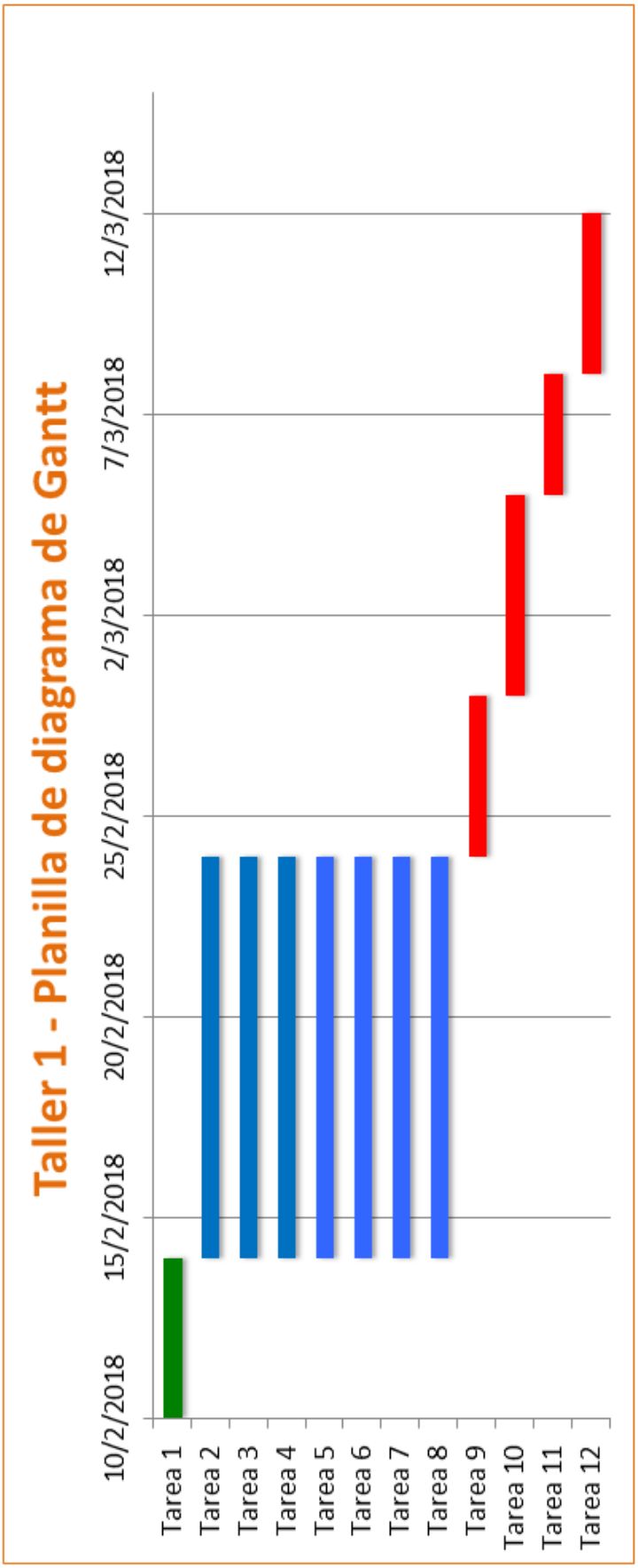
5) Cronograma y planificación

Para la implementación, y las tareas relacionadas a la defensa, se dispuso la utilización del diagrama Gantt, mediante una planilla de cálculo.

Dicha planilla contiene fórmulas. Partiendo de las definiciones de las tareas, junto a su rango de tiempo correspondiente, se consolida en una línea de tiempo, en forma de gráfica, permitiéndonos visualizar la cantidad de días y la distribución de las tareas.

A continuación, el contenido de dicha planificación.

Nombre de la tarea	Fecha de inicio	Fecha final	Duración (días)	Descripción
Tarea 1	10/2/2018	14/2/2018	4	Desarrollo del main.cpp, contemplando los tres comandos y los cabezales
Tarea 2	14/2/2018	24/2/2018	10	Desarrollo del módulo string y listaStrings, incluyendo el testing unitario
Tarea 3	14/2/2018	24/2/2018	10	Desarrollo del módulo enumOpsBasicas y enumOpsArits, incluyendo el testing unitario
Tarea 4	14/2/2018	24/2/2018	10	Desarrollo del módulo variable y arbolVariables, incluyendo el testing unitario
Tarea 5	14/2/2018	24/2/2018	10	Desarrollo del módulo AS3, AS4, AS5 y AS6, incluyendo el testing unitario
Tarea 6	14/2/2018	24/2/2018	10	Desarrollo del módulo instruccion y listaInstrucciones, incluyendo el testing unitario
Tarea 7	14/2/2018	24/2/2018	10	Desarrollo del módulo validaciones, incluyendo el testing unitario
Tarea 8	14/2/2018	24/2/2018	10	Desarrollo del módulo archivo, incluyendo el testing unitario
Tarea 9	24/2/2018	28/2/2018	4	Testing regresivo y testing completo
Tarea 10	28/2/2018	5/3/2018	5	Ajustes y correcciones de issues/bugs reportados en el testing
Tarea 11	5/3/2018	8/3/2018	3	Ajustes de documentación y armado de zip entregable
Tarea 12	8/3/2018	12/3/2018	4	Armado de presentación (ppt), speeches y DEMO para la defensa


Taller 1 - Planilla de diagrama de Gantt
10/2/2018
15/2/2018
20/2/2018
25/2/2018
2/3/2018
7/3/2018
12/3/2018
Tarea 1
Tarea 2
Tarea 3
Tarea 4
Tarea 5
Tarea 6
Tarea 7
Tarea 8
Tarea 9
Tarea 10
Tarea 11
Tarea 12

6) Decisiones y consideraciones del diseño de la solución

Se exponen a continuación, las consideraciones y decisiones, evaluadas por el grupo.

Como primera instancia, tras analizar la letra pudimos reconocer dos usuarios o roles en torno a esta realidad. Por un lado tenemos al 'Usuario Programador' el cual, creara el código fuente. También, este último, podría comportarse como 'Usuario Final' el cual podrá compilar y ejecutar un programa compilado. En este caso, el 'Usuario Final' interactuara con el programa desarrollado por el 'Usuario Programador'.

Validaciones de COMPILACION:

a. Cada error de compilación que se encuentre, se deberá detener la compilación y mostrar un mensaje de error lo más aproximado y especifico posible, de modo que el 'Usuario Programador' pueda identificar y solucionar el problema correspondiente. En el mismo orden, se liberara toda memoria dinámica reservada, como Listas y ABBs.
b. El orden de las secciones tienen solo un orden definido.
c. Asumimos que los usuarios no sobrepasaran a MAX (para los strings).
d. En todo el código fuente y prompt puede existir varios espacios entre palabras.
e. Las secciones siempre deberán ser escritas en MAYUSCULA y sin números.
f. Los nombres de programa siempre deberán ser escritos en minúscula y sin números, una sola palabra. Los nombres de programa debe coincidir con el nombre del archivo y el nombre escrito luego del comando 'compilar' en el prompt de nuestro compilador.
g. Las variables siempre deberán ser escritos en minúscula y sin números; y serán separadas por un espacio dentro de la sección VARIABLES. No se podrán repetir. Las variables solo guardaran valores enteros (int). Serán inicializadas con valor cero (sin basura), cuando se guarden en el ABB, con su nombre como ID. Como las variables son en minúscula, puede existir variables con nombre de secciones y comandos.
h. Las instrucciones (operaciones básicas) dentro de la sección INTRUCCIONES deben ser LEER, MOSTRAR o una asignación a una variable declarada (verificando en el ABB). La ultima instrucción del código fuente determinara el fin del programa; si existe texto libre al final del archivo, el compilador intentara validar el comienzo de alguna instrucción y esta podría dar error de compilación. No puede haber líneas vacías.

FORMATO DE INSTRUCCIONES:

1	LEER	var			
2	MOSTRAR	var			
3(AS1)	var	=	int		
4(AS2)	var	=	var		
5(AS3)	var	=	FUNC	int	int
6(AS4)	var	=	FUNC	int	var
7(AS5)	var	=	FUNC	var	int
8(AS6)	var	=	FUNC	var	var

*FUNC = {SUM, RES, MUL, DIV}

i. Hay únicamente seis formas de asignación, si no se respetan se producirá un error de compilación. Solo se permite una FUNC por instrucción.
j. Si la compilación fue exitosa, se procederá a guardar el ABB con las variables, y la lista con las instrucciones en sus respectivos archivos, luego se libera la memoria pertinente.

Validaciones de PROMPT y EJECUCION:

a) Se pedirá el ingreso de un comando y se controlara que efectivamente sea uno de los tres (“compilar”, “ejecutar”, “salir”).
b) Mientras el usuario no ingrese el comando “salir”, el programa pedirá el ingreso de un comando.
c) El comando “salir” es el único que no necesita ningún argumento después de su escritura.
d) Los comandos “compilar” y “ejecutar”, son los únicos que necesitan solo, y solamente un argumento luego de su escritura. Este argumento, debe ser un nombre de programa valido, en minúsculas y sin números, una sola palabra, sin extensión. Para compilar la valides depende si existe el archivo con la extensión pertinente, con el código fuente. Para ejecutar se validara la existencia de los dos archivos con su extensión correspondiente, generados luego de la correcta compilación.
e) Se asume que los archivos están en la misma raíz de nuestro programa, y que si están los archivos de compilación, el programa fue compilado correctamente.
f) Ante cualquier error en tiempo de ejecución, se deberá detener la ejecución y mostrar un mensaje de error lo más aproximado y especifico posible, de modo que el usuario que está ejecutando pueda identificar el error, y en otra disposición, el ‘Usuario Programador’ pueda identificar y solucionar el problema correspondiente. En el mismo orden, se liberara toda memoria dinámica reservada.
g) Al ejecutar, se cargaran las estructuras de los archivos en memoria y se cerraran estos archivos para no modificarlos posteriormente.
h) En tiempo de ejecución, con la instrucción LEER, se le pedirá al usuario un entero y se leerá como cadena se strings, de modo que se convertirá y si el usuario no ingreso un entero no se producirá un error de C++. Se detendrá la ejecución y se mostrara el mensaje de error.
i) En tiempo de ejecución de un programa, se validara y controlara la división por cero, de modo que si se presenta una instrucción de este tipo, se producirá un error y se mostrara el mensaje pertinente. Así, nunca se producirá un error de lenguaje C++, sino que advertirá antes mediante nuestro software.
j) Luego de la correcta ejecución de las instrucciones, se liberara toda memoria dinámica de estructuras como strings dinámicos, listas y ABBs.


UDE
UNIVERSIDAD DE LA EMPRESA