Práctica 4: TABLA DE SÍMBOLOS

Fecha de entrega según calendario publicado en moodle o comentado en clase

Objetivo de la práctica:

El objetivo de la práctica es la codificación de una librería en la que implementes la tabla de símbolos con la funcionalidad adecuada para atender las restricciones asociadas con el lenguaje de programación orientado a objetos ómicron que estás desarrollando en el laboratorio este curso.

La especificación del lenguaje está contenida en el material que lo describe y que se te ha ido proporcionando y explicando en las sesiones dedicadas a estos contenidos en los laboratorios.

El objetivo es que tu librería sea compatible con el uso descrito en los laboratorios y ejemplificado en estas páginas.

Más adelante encontrarás la especificación de las funciones de alto nivel que tienes que respetar así como ejemplos.

Así mismo, en este enunciado se describen mediante ejemplos los casos de uso que tu código tiene que ser capaz de atender.

Para desarrollar la práctica se te pedirá que desarrolles un programa principal que lea de un fichero indicaciones de operaciones sobre la tabla de símbolos y que genere un salida en un fichero de texto.

Para todo ello debes seguir las indicaciones que se describen a continuación.

Descripción de la librería:

Las funciones que debes implementar junto con su descripción se muestran a continuación. Su explicación detallada es objeto de las diferentes semanas dedicadas a la solución de esta práctica. Ten esto en cuenta porque no vas a encontrar aquí todos los detalles de su funcionalidad.

Como se ha indicado previamente, esta práctica se va a evaluar utilizando un programa principal que a partir de ciertas indicaciones de operaciones sobre la tabla de símbolos, realice las llamadas correspondientes a tus funciones y genere la salida adecuada. Eso significa que **a pesar de que hemos**

intentado ser lo más precisos posible a la hora de describir las funciones (dada su complejidad) tanto las cabeceras como incluso las funciones en sí mismas SON SUGERENCIAS pero representan la funcionalidad "de referencia" que articula nuestras explicaciones. Tienes, por tanto, libertad para organizar tu librería como consideres más oportuno asegurándote de que IMPLEMENTAS TODA LA FUNCIONALIDAD QUE SE TE PIDE y de que, además, MANTIENES LAS SALIDAS EQUIVALENTES A LAS DESCRITAS EN ESTA PRÁCTICA.

Agruparemos las funciones de la siguiente manera:

- Funciones de gestión global de las tablas y los ámbitos.
- Funciones de búsqueda de identificadores cuando aparecen en la parte de sentencias para ser usados.
- Funciones de búsqueda de identificadores cuando aparecen en la parte de declaraciones para ser declarados.
- Funciones de inserción en la tabla de símbolos que no involucren la creación de un ámbito.
- Funciones que complementan la funcionalidad de la tabla de símbolos.

Funciones de gestión de tablas y ámbitos

```
int iniciarTablaSimbolosClases(tablaSimbolosClases** t, char * nombre);
```

- Reserva todos los recursos para crear una tabla de símbolos basada en un grafo e identificada con el nombre proporcionado como argumento.
- La tabla se deja en el primer argumento.

```
int abrirClase(tablaSimbolosClases* t, char* id_clase);
```

- Realiza las tareas de añadir al grafo una nueva raíz.

```
int abrirClaseHereda(tablaSimbolosClases* t, char* id_clase, ...);
o
int abrirClaseHeredaN (tablaSimbolosClases* t,
                       char* id_clase,
                       int num_padres,
                       char** nombres_padres);
```

- Realiza las tareas de añadir al grafo un nuevo nodo que debe conectarse a los nombres identificados mediante los últimos argumentos.
- Elige una versión:
    - La primera tiene un número variable de padres y el último es NULL, su significado es el siguiente:
        - El primero de ellos (id_clase) es el de la clase nueva
        - El resto son todos char * que deben contener los nombres de las clases de las que hereda id_clase en el mismo orden de la cláusula inherits correspondiente
        - Para indicar el fin de la serie de padres, se utilizar un último argumento que siempre valdrá NULL
    - La segunda versión incluye explícitamente el número de padres y sus nombres en un vector de char*.

```
int cerrarClase(tablaSimbolosClases* t,
                char* id_clase,
                int num_atributos_clase,
                int num_atributos_instancia,
                int num_metodos_sobreescribibles,
                int num_metodos_no_sobreescribibles);
```

- Realiza tareas asociadas con el final de la clase identificada mediante el segundo argumento.

- Los datos numéricos son necesarios para actualizar la información que se habrá calculado a lo largo del proceso de la clase.
- Es posible que en este momento no tenga significado para ti estos argumentos, en ese caso, utiliza el valor 0 en su llamada.
- Es posible que en el futuro decidas añadir más información necesaria para el proceso de actualización, en ese caso simplemente has de añadirlos.

```
int cerrarTablaSimbolosClases(tablaSimbolosClases* t);
```

- Realiza tareas asociadas con el final de la clase identificada mediante el segundo argumento
- Lo normal es que uses esta función al final del programa.

```
int abrirAmbitoPpalMain(tablaAmbitos** t);
```

- Realiza tareas asociadas con la apertura del ámbito principal de la tabla de símbolos por ámbitos de main.

```
int abrirAmbitoMain(tablaAmbitos ** t,
                    char* id_ambito,
                    int categoria_ambito,
                    int tipo_ambito,···
                    int tamanio);
```

- Realiza tareas asociadas con la apertura del ámbito asociado con una función global dentro del ámbito main:
    - Su tabla de símbolos por ámbitos se proporciona en t.
    - Su nombre es id_ambito.
    - El tipo de retorno es tipo_ambito.
    - Y debes añadir la información que necesites para completar la operación.

```
int cerrarAmbitoMain(tablaSimbolosAmbitos* t)
```

- Realiza tareas asociadas con el cierre del ámbito asociado con una función global dentro del ámbito main, cierra el ámbito actual, de hecho.

```
int tablaSimbolosClasesAbrirAmbitoEnClase(     tablaSimbolosClases * grafo,
                                        char * id_clase,
                                        char* id_ambito,
                                        int categoria_ambito,
                                        int tipo_ambito,
                                        int acceso_ambito,
                                        int tipo_miembro,
                                        int posicion_metodo_sobre,
                                        int tamanio,...);
```

- Realiza tareas asociadas con la apertura del ámbito asociado con un método:
    - En grafo, y
    - en la clase id_clase
    - de nombre id_ambito
    - con la información de símbolo que lo describe, por ejemplo el conjunto de argumentos categoria_ambito,...,posicion_metodo_sobre.
    - El último argumento, si te es necesario, sería el tamaño inicial de la tabla del ámbito.

```
int tablaSimbolosClasesCerrarAmbitoEnClase(tablaSimbolosClases * grafo,
                                           char * id_clase);
```

- Realiza tareas asociadas con el cierre del ámbito asociado con un método:
    - En la tabla grafo,
    - y en la clase id_clase.

Búsqueda de identificadores en la parte de sentencias

```
int buscarIdEnJerarquiaDesdeClase(tablaSimbolosClases *t,
                                  char * nombre_id,
                                  char * nombre_clase_desde,
                                  elementoTablaSimbolos ** e,
                                  char * nombre_ambito_encontrado);
```

Que

- Busca en la jerarquía de clases (en las tablas tablaAmbitos de cada clase)
- a partir de la clase cuyo nombre es `nombre_clase_desde`
- el símbolo `nombre_id`.
- El argumento `e` es de salida y contendrá la información asociada a `nombre_id` de la tabla de símbolos en caso de haberlo encontrado o NULL en caso contrario.
- El argumento `nombre_ambito_encontrado` es un argumento de salida que tendrá el nombre del ámbito donde se ha encontrado el símbolo en el caso de que se haya encontrado o NULL en caso contrario.
- Debe devolver OK si lo encuentra o ERR en caso contrario.

```
int buscarIdNoCualificado( tablaSimbolosClases *t,
                                  tablaAmbitos *tabla_main,
                                   char * nombre_id,
                                  char * nombre_clase_desde,
                                  elementoTablaSimbolos ** e,
                                  char * nombre_ambito_encontrado);
```

Es la función que se responsabiliza de la búsqueda de un identificador cuando aparece en las sentencias y se quiere usar. El identificador no debe ir cualificado. Más en concreto:

- Busca `nombre_id`
- en la tabla de símbolos de clases proporcionada,
- y si es necesario también en la del ámbito main (`tabla_main`),
- a partir de la clase de nombre `nombre_clase_desde`.
- El resto de argumentos se ha descrito previamente.
- Debe devolver OK si lo encuentra o ERR en caso contrario.

```
int buscarIdIDCualificadoClase(  tablaSimbolosClases *t,
                                  char * nombre_clase_cualifica,
                                  char * nombre_id,
                                  char * nombre_clase_desde,
                                  elementoTablaSimbolos ** e,
                                  char * nombre_ambito_encontrado)


int buscarIdCualificadoInstancia(tablaSimbolosClases *t,
                                  tablaSimbolosAmbitos * tabla_main,
                                  char * nombre_instancia_cualifica,
                                  char * nombre_id,
                                  char * nombre_clase_desde,
                                  elementoTablaSimbolos ** e,
                                  char * nombre_ambito_encontrado)
```

Estas funciones buscan identificadores cualificados cuando aparecen así en la parte de sentencias (expresiones del tipo <identificador>.<identificador>). La primera se utiliza cuando se cualifica con el nombre de una clase y la segunda cuando se cualifica con el nombre de una instancia. Más en concreto:

- Busca `nombre_id`
- en la tabla de símbolos de clases proporcionada

- y, si es necesario, también en la del ámbito main `tabla_main` (sólo en el segundo caso),
- cualificado respectivamente por `nombre_clase_cualifica` o `nombre_instancia_cualifica`,
- a partir de la clase de nombre `nombre_clase_desde`.
- El resto de argumentos se ha descrito previamente.
- Debe devolver OK si lo encuentra o ERR en caso contrario.

Búsqueda de identificadores en la parte de declaraciones

```
int buscarParaDeclararMiembroClase( tablaSimbolosClases *t,
                                    char * nombre_clase_desde,
                                    char * nombre_miembro,
                                    elementoTablaSimbolos ** e,
                                    char * nombre_ambito_encontrado)

int buscarParaDeclararMetodoClase(      tablaSimbolosClases *t,
                                        char * nombre_clase_desde,
                                        char * nombre_miembro,
                                        elementoTablaSimbolos ** e,
                                        char * nombre_ambito_encontrado)
```

Estas funciones realizan la búsqueda de identificadores cuando, al declararlos, se intenta insertarlos en la tabla de símbolos. La primera se utiliza cuando se quiere declarar un miembro de clase (ya sea método o atributo) la segunda cuando se quiere declarar uno de instancia. Más en concreto:

- Busca `nombre_miembro`
- en la tabla de símbolos de clases `t` proporcionada como argumento
- a partir de la clase `nombre_clase_desde`.
- El resto de los argumentos se ha explicado ya.
- Debe devolver OK si lo encuentra y ERR si no.
- Es muy importante que te des cuenta de **que estas funciones no insertan los símbolos** sólo realizan la búsqueda previa a su inserción.

```
int buscarTablaSimbolosAmbitoActual(tablaAmbitos * t,
                                    char* id,
                                    elementoTablaSimbolos** e,
                                    char* id_ambito);
```

Recuerda que las tablas de símbolos de cada nodo (clase) y del main están representadas por el tipo de dato tablaAmbitos. Para la búsqueda previa a la declaración de un símbolo en las clases, queda como funcionalidad interna la búsqueda que se realiza en el ámbito actual. Sin embargo, cuando gestiones esa situación en el ámbito main tienes que realizar de forma explícita la búsqueda. Por eso declaramos como una función de alto nivel esta función que:

- Busca el símbolo `id`
- en la tabla de símbolos `t` (especialmente pensada para main).
- El resto de argumentos ya ha sido descrito.
- Devuelve OK si lo encuentra y ERR si no.
- Es muy importante que te des cuenta de **que estas funciones no insertan los símbolos** sólo realizan la búsqueda previa a su inserción.

Observa que cuando estás en la tabla de símbolos de ámbitos de main, puede ser que quieras declarar:

- Si estás en el ámbito principal de la tabla de main:
    - Una variable global
    - Una función, antes de abrir su ámbito
- Si estás en el ámbito local de una función de main:

- Una variable local
- Un parámetro

Ésta es la función que debes utilizar para comprobar si puedes declarar ese nuevo símbolo en todos los casos.

```
int buscarTablaSimbolosClasesAmbitoActual( tablaSimbolosClases *t,
                                           char * nombre_clase,
                                           char * nombre_id,
                                           elementoTablaSimbolos ** e,
                                           char * nombre_ambito_encontrado)
```

Esta función se utilizará en situaciones similares a la anterior pero cuando se está en un método de una clase.

Observa que las peculiaridades de la declaración de un miembro de clase (atributo o método) y de un miembro de instancia (atributo o método) han motivado funciones específicas descritas anteriormente.

Las situaciones que no se han cubierto aún son precisamente:

- La declaración de un parámetro de un método
- La declaración de una variable local de un método

Ésta es la función que debes utilizar en esos casos.

Esta función:

- busca en la tabla de símbolos de ámbitos de la clase argumento `nombre_clase`
- el id `nombre_id` (que debe ir precedido por su correspondiente prefijo de ámbito).
- Los otros argumentos ya se han explicado previamente.
- Devuelve OK si lo encuentra y ERR si no.
- Es muy importante que te des cuenta de **que estas funciones no insertan los símbolos,** sólo realizan la búsqueda previa a su inserción.

Funciones inserción símbolos que no involucren creación de ámbitos

```
int insertarTablaSimbolosClases( tablaSimbolosClases * grafo,
                                 char * id_clase,
                                 char* id,
                                 int clase,
                                 int tipo,
                                 int estructura,
                                 int direcciones,
                                 int numero_parametros,
                                 int numero_variables_locales,
                                 int posicion_variable_local,
                                 int posicion_parametro,
                                 int dimension,
                                 int tamanio,
                                 int filas,
                                 int columnas,
                                 int capacidad,
                                 int numero_atributos_clase,
                                 int numero_atributos_instancia,
                                 int numero_metodos_sobreescribibles,
                                 int numero_metodos_no_sobreescribibles,
                                 int tipo_acceso,
                                 int tipo_miembro,
                                 int posicion_atributo_instancia,
                                 int posicion_metodo_sobreescribible,
                                 int num_acumulado_atributos_instancia,
                                 int num_acumulado_metodos_sobreescritura,
                                        int
```

```
posicion_acumulada_atributos_instancia,
                                        int
posicion_acumulada_metodos_sobreescritura,
                                  int * tipo_args)
```
Realiza las tareas de inserción de un símbolo:

- En la tabla de clases `grafo`
- En la clase `id_clase`
- del elemento de nombre `id`,
- descrito por el resto de parámetros que sólo son ejemplos de los que tú tal vez pudieras usar.

Funcionalidad variada

```
int aplicarAccesos( tablaSimbolosClases *t,
                    char * nombre_clase_ambito_actual,
                    char * clase_declaro,
                    elementoTablaSimbolos * pelem);
```
Esta función realiza el control de acceso aplicando la política de cualificadores de acceso (hidden, secret y exposed)

- Para la tabla de clases proporcionada como argumento,
- desde la clase `nombre_clase_ambito_actual`,
- para un símbolo definido en `clase_declaro`
- y cuya información de la tabla de símbolos está incluida en `pelem`.

```
tablaSimbolosClases * tablaSimbolosClasesToDot(tablaSimbolosClases * grafo);
```
Esta función genera el fichero que contiene el gráfico en formato dot de la tabla de símbolos de clases como se ha descrito en el laboratorio y que tiene como nombre el nombre de la tabla (con el que se creó) con la extensión .dot.

Descripción del programa de prueba que tienes que codificar:

Analiza el siguiente programa principal ómicron.

```
main
{

    int v1;

    class AA {
        exposed int a1;
        secret unique int sa1;
        exposed unique int a2;

        function exposed int mA1(int pmA1) {
            a1=5;
            printf v1;
            return v1*pmA1;

        }
    };

    class BB inherits AA {
        int b1;
```

```
              {AA} aa1;
              function exposed int mB1(int pmB1){
            printf aa1.a1;
            printf aa1.sa1;
              }
    };

    {AA} a;

    a = instance_of (AA);

    v1=3;
    v1 = a.mA1(2);
    printf v1;
    printf a.a1;
    printf x;

    printf AA.sa1;

    discard a;
}
```

A continuación se describe cómo se indicará a tu programa de prueba que realice estas operaciones:

FORMATO DE LA OPERACIÓN	SIGNIFICADO
inicia_tsa_main	Inicia la tabla de ámbitos de main
abrir_ambito_ppal_main	Abre el ámbito ppal de main
buscar declarar_main nombre_id	Busca un símbolo para ser declarado en la tabla de ámbitos de main
buscar declarar_miembro_clase nombre_clase_desde nombre_miembro	Busca para declarar como miembro de clase un símbolo en la tabla de clases a partir de una clase
buscar declarar_miembro_instancia nombre_clase_desde nombre_miembro	Busca para declarar como miembro de instancia un símbolo en la tabla de clases a partir de una clase
buscar declarar_id_local_metodo nombre_clase nombre_id	Busca para declarar como variable local o parámetro un símbolo, en el método actual de la clase indicada
buscar jerarquia nombre_id nombre_clase	Función interna que busca en la jerarquía un id a partir de una clase
buscar id_no_cualificado nombre_id nombre_clase_desde	Búsqueda para ser usado en alguna sentencia de un id no cualificado desde cualquier lugar del programa
buscar id_cualificado_instancia nombre_instancia nombre_id nombre_clase_desde	Búsqueda para ser usado en alguna sentencia de un id cualificado a través de una instancia desde cualquier lugar del programa
buscar id_cualificado_clase nombre_clase_cualifica nombre_id nombre_clase_desde	Búsqueda para ser usado en alguna sentencia de un id cualificado a través de una clase desde cualquier lugar del programa

insertar_tsa_main nombre_id categoria tipo_basico estructura tipo_acceso tipo_miembro	Insertar un símbolo cuando se está en los ámbitos de main
abrir_ambito_tsa_main nombre_ambito tipo_basico	Apertura del ámbito asociado con una nueva función global en el ámbito principal de main
cerrar_ambito_tsa_main	Cerrar el ámbito asociado con una función global en el ámbito principal de main
inicia_tsc	Iniciar la tabla de símbolos de clases
abrir_clase nombre_clase	Abrir una nueva clase que no hereda de ninguna
abrir_clase_hereda nombre_clase nombre_padre1 ... nombre_padren	Abrir una nueva clase que hereda de una lista
insertar_tsc nombre_clase nombre_simbolo categoria tipo_basico estructura tipo_acceso tipo_miembro	Insertar un símbolo desde cualquier lugar de las clases del programa
abrir_ambito_tsc nombre_clse nombre_ambito categoria tipo_basico tipo_acceso tipo_miembro	Abrir el ámbito de un nuevo método de una clase
cerrar_ambito_tsc nombre_clase	Cerrar el ámbito de un método de una clase
cerrar_clase nombre_clase	Cerrar una clase
cerrar_ambito_tsa_main	Cerrar el ámbito ppal del main
cerrar_tsa_main	Dar por terminada la gestión de la tabla de ámbitos del main
cerrar_tsc	Dar por terminada la gestión de la tabla de clases

Basándonos en las funciones descritas en el laboratorio y resumidas al comienzo de este enunciado, se te indica una posible implementación de la gestión de cada operación en la siguiente tabla:

Es un esqueleto que sólo te indica la tarea que “conceptualmente” deberías hacer.

No olvides realizar toda la funcionalidad indicada con tus propias funciones.

No olvides que los mensajes que debes escribir se describen de manera conjunta un poco más tarde.

FORMATO DE LA OPERACIÓN	POSIBLE IMPLEMENTACIÓN DE SU GESTIÓN
inicia_tsa_main	`iniciarTablaSimbolosAmbitos(&tabla_main);`
abrir_ambito_ppal_main	`abrirAmbitoPpalMain(&tabla_main);`
buscar declarar_main nombre_id	`buscarParaDeclararIdTablaSimbolosAmbitos(tabla_main, “main_v1”, &pelem, id_ambito);`

buscar declarar_miembro_clase nombre_clase_desde nombre_miembro	buscarParaDeclararMiembroClase(tabla_clases, "AA", "AA_sA1", &pelem, id_ambito) == ERR)
buscar declarar_miembro_instancia nombre_clase_desde nombre_miembro	buscarParaDeclararMiembroInstancia(tabla_clases, "AA", "AA_sa2", &pelem, id_ambito)
buscar declarar_id_local_metodo nombre_clase nombre_id	if (buscarParaDeclararIdLocalEnMetodo(tabla_clases, nombre_clase_desde, id, &pelem, id_ambito) == OK)
buscar jerarquia nombre_id nombre_clase	buscarIdEnJerarquiaDesdeClase(ej_tabla_clases, "pmA1", "AA", &pelem, id_ambito)
buscar id_no_cualificado nombre_id nombre_clase_desde	buscarIdNoCualificado(ej_tabla_clases, tabla_main, "v1", "AA", &pelem, id_ambito)
buscar id_cualificado_instancia nombre_instancia nombre_id nomber_clase_desde	buscarIdCualificadoInstancia(ej_tabla_clases, tabla_main, "MainMiBB2", "BmiAA3", "JJ", &pelem, id_ambito);
buscar id_cualificado_clase nombre_clase_cualifica nombre_id nombre_clase_desde	buscarIdIDCualificadoClase(ej_tabla_clases, "AA", "clase-prot", "AA", &pelem, id_ambito)
insertar_tsa_main nombre_id categoria tipo_basico estructura tipo_acceso tipo_miembro	insertarTablaSimbolosAmbitos(&tabla_main, "main_v1",VARIABLE, INT, ESCALAR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0, 0, ACCESO_CLASE, MIEMBRO_UNICO, 0, 0, 0, 0, **0**,**0**, NULL);
abrir_ambito_tsa_main nombre_ambito tipo_basico	abrirAmbitoMain(&tabla_main,ambito,FUNCION, tipo_basico, NINGUNO, NINGUNO, 0, TAMANIO_TABLA);
cerrar_ambito_tsa_main	cerrarAmbitoMain(&tabla_main);
inicia_tsc	iniciarTablaSimbolosClases(&tabla_clases, token);
abrir_clase nombre_clase	abrirClase(ej_tabla_clases,nombre_clase); graph_enrouteParentsLastNode(ej_tabla_clases) ;
abrir_clase_hereda nombre_clase nombre_padre1 ... nombre_padren	while ((token=strtok(NULL," \n\t"))!=NULL) { num_padres++; nombres_padres = (char **) realloc(nombres_padres, sizeof(char*)*num_padres); nombres_padres[num_padres-1]= (char*)malloc((strlen(token)+1)*sizeof(char)) ; strcpy(nombres_padres[num_padres-1], token); } abrirClaseHeredaN(tabla_clases, nombre_clase_declarandose,

	num_padres,nombres_padres); graph_enrouteParentsLastNode(tabla_clases);
insertar_tsc nombre_clase nombre_simbolo categoria tipo_basico estructura tipo_acceso tipo_miembro	insertarTablaSimbolosClases(ej_tabla_clases,"AA","AA_a1", ATRIBUTO_CLASE, INT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0, 0, ACCESO_TODOS, MIEMBRO_UNICO, 0, 0, 0, 0, **0,0**,NULL);
abrir_ambito_tsc nombre_clse nombre_ambito categoria tipo_basico tipo_acceso tipo_miembro	tablaSimbolosClasesAbrirAmbitoEnClase(ej_tabla_clases,"AA","AA_mA0", METODO_SOBREESCRIBIBLE, ACCESO_TODOS, MIEMBRO_NO_UNICO, 0, TAMANIO_TABLA);
cerrar_ambito_tsc nombre_clase	tablaSimbolosClasesCerrarAmbitoEnClase(ej_tabla_clases, "AA");
cerrar_clase nombre_clase	cerrarClase(ej_tabla_clases,"AA",0,0,0,0);
cerrar_ambito_tsa_main	cerrarAmbito(&tabla_main);
cerrar_tsa_main	cerrarAmbitoMain(&tabla_main); liberarTablaSimbolosAmbitos(&tabla_main);

cerrar_tsc	cerrarTablaSimbolosClases(ej_tabla_clases);
	tablaSimbolosClasesToDot(ej_tabla_clases);
	graph_destroy(ej_tabla_clases);

La salida que debe escribirse ante cada petición será la siguiente:

- Se copiará la petición tal y como está en el fichero.
- A continuación se describirá brevemente el resultado de la operación, sobre todo en lo relativo a las funciones de búsqueda. Por favor, utiliza el ejemplo de salida que tienes más abajo para dar formato a los mensajes de tu programa.
    - En los casos en los que haya información adicional asociada con el retorno se resumirá la misma.
    - Si es un error asociado con una búsqueda de un símbolo que no existe y se quiere declarar
        - `No existe id: se puede declarar`
    - Si en esta misma situación el símbolo no existe
        - `Existe id: no se puede declarar`
    - Si es un error asociado con accesos
        - `No accesible <tipo acceso>`
    - Si es un error asociado con una búsqueda de un símbolo para ser usado y no existe o no es accesible
        - `No existe id`
    - Si en esa misma situación el símbolo existe
        - `Existe id`

- Las operaciones que impliquen un cambio en la tabla de símbolos (ya sea la del ámbito main o el grafo de clases) tienen que terminar su salida mostrando la tabla de símbolos, al menos
    - La tabla de ámbitos de la clase involucrada (si es una operación sobre una clase)
    - La tabla de ámbitos de main (si es ella la involucrada)
- El formato de la salida depende fuertemente de la información que hayas decidido guardar hasta este instante, por ello, fíjate en la salida del ejemplo para imitar su formato en tus mensajes.

A continuación se muestra un ejemplo de fichero de entrada que corresponde al fuente ómicron mostrado antes:

Observa en la línea resaltada que los valores numéricos hacen referencia a las constantes que aparecen en el anexo al final de este enunciado.

```
inicia_tsc    grafo_enunciado
inicia_tsa_main
abrir_ambito_ppal_main
buscar declarar_main  main_v1
insertar_tsa_main     main_v1 1       1       1       3       2
abrir_clase   AA
buscar declarar_miembro_instancia   AA     AA_a1
insertar_tsc  AA     AA_a1  8       1       1       3       2
buscar declarar_miembro_instancia   AA     AA_sa1
insertar_tsc  AA     AA_sa1 7       1       1       1       1
buscar declarar_miembro_clase       A      A_a2
insertar_tsc  AA     AA_a2  7       1       1       3       1
buscar declarar_miembro_instancia   AA     AA_mA1@1
abrir_ambito_tsc      AA     AA_mA1@1       5       3       3       2
buscar declarar_id_local_metodo     AA     mA1@1_pmA1
insertar_tsc  AA     mA1@1_pmA1     2       1       1       3       2
buscar id_no_cualificado     a1     AA
buscar id_no_cualificado     v1     AA
buscar id_no_cualificado     pmA1   AA
cerrar_ambito_tsc     AA
cerrar_clase  AA
abrir_clase_hereda    BB     AA
buscar declarar_miembro_instancia   BB     BB_b1
insertar_tsc  BB     BB_b1  8       1       1       3       2
buscar declarar_miembro_instancia   BB     BB_aa1
insertar_tsc  BB     BB_aa1 8       0       1       3       2
buscar declarar_miembro_instancia   BB     BB_mB1@1
abrir_ambito_tsc      BB     BB_mB1@1       5       1       3       2
buscar declarar_id_local_metodo     BB     mB1@1_pmB1
insertar_tsc  BB     mB1@1_pmB1     2       1       1       3       2
buscar id_cualificado_instancia     aa1    a1     BB
buscar id_cualificado_instancia     aa1    sa1    BB
cerrar_ambito_tsc     BB
cerrar_clase  BB
buscar declarar_main  main_a
insertar_tsa_main     main_a         1       0       1       3       2
buscar id_no_cualificado     a      main
buscar id_no_cualificado     v1     main
buscar id_no_cualificado     v1     main
buscar id_cualificado_instancia     a      mA1@1  main
buscar id_no_cualificado     v1     main
buscar id_cualificado_instancia     a      a1     main
buscar id_no_cualificado     x      main
buscar id_cualificado_clase  AA     a1     main
buscar id_cualificado_clase  AA     sa1    main
buscar id_no_cualificado     a      main
cerrar_tsa_main
cerrar_tsc
```

Y que generaría una salida similar a la siguiente (observa que no hemos fijado el formato con el que se imprime la tabla de símbolos, aquí, por ejemplo, puedes ver el nombre de cada ámbito y luego cada símbolo con la posición que ocupa en la tabla hash y un resumen de la información guardada):

```
inicia_tsc
inicia_tsa_main
abrir_ambito_ppal_main
buscar declarar_main main_v1: No encontrado: se puede declarar
insertar_tsa_main main_v1 1 1 1 3 2

=================== main =================

****************Posicion 50 ******************
main_v1 VARIABLE       POS_LOCAL: 0   POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR
TIPO: ENTERO   DIR: 0  ACCESO: 3      MIEMBRO: 2

****************Posicion 93 ******************
main_main      CLASE   ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO
SOBR. 0 ACUM ATR INS Y 0 ACUM MET SOBR.

abrir_clase AA.
buscar declarar_miembro_instancia AA AA_a1: No encontrado: se puede declarar
insertar_tsc AA AA_a1 8 1 1 3 2

=================== AA =================

****************Posicion 144 ******************
AA_AA   CLASE   ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 160 ******************
AA_a1   ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO  DIR: 0  ACCESO: 3      MIEMBRO: 2
buscar declarar_miembro_instancia AA AA_sa1: No encontrado: se puede declarar
insertar_tsc AA AA_sa1 7 1 1 1 1

=================== AA =================

****************Posicion 64 ******************
AA_sa1  ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0  ACCESO: 1      MIEMBRO: 1

****************Posicion 144 ******************
AA_AA   CLASE   ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 160 ******************
AA_a1   ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO  DIR: 0  ACCESO: 3      MIEMBRO: 2
buscar declarar_miembro_clase A A_a2: No encontrado: se puede declarar
insertar_tsc AA AA_a2 7 1 1 3 1

=================== AA =================

****************Posicion 64 ******************
AA_sa1  ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0  ACCESO: 1      MIEMBRO: 1

****************Posicion 144 ******************
AA_AA   CLASE   ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 160 ******************
AA_a1   ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO  DIR: 0  ACCESO: 3      MIEMBRO: 2
```

```
****************Posicion 161 ******************
AA_a2  ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0 ACCESO: 3      MIEMBRO: 1
buscar declarar_miembro_instancia AA AA_mA1@1: No encontrado: se puede declarar
abrir_ambito_tsc AA AA_mA1@1 5 3 3 2

=================== mA1@1 =================

****************Posicion 139 ******************
AA_mA1@1       METODO SOBREESCRIBIBLE       TIPO: BOOLEAN  CLASE: (null)  #PAR: 0 #LOCAL: 0
       ACCESO: 3      MIEMBRO: 2

=================== AA =================

****************Posicion 64 ******************
AA_sa1 ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0 ACCESO: 1      MIEMBRO: 1

****************Posicion 139 ******************
AA_mA1@1       METODO SOBREESCRIBIBLE       POS METODO: 0 Y ACUMULADA 0  TIPO: BOOLEAN
CLASE: (null)  #PAR: 0 #LOCAL: 0      ACCESO: 3      MIEMBRO: 2

****************Posicion 144 ******************
AA_AA  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 160 ******************
AA_a1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO  DIR: 0  ACCESO: 3      MIEMBRO: 2

****************Posicion 161 ******************
AA_a2  ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0 ACCESO: 3      MIEMBRO: 1
buscar declarar_id_local_metodo AA mA1@1_pmA1: No encontrado: se puede declarar
insertar_tsc AA mA1@1_pmA1 2 1 1 3 2

=================== mA1@1 =================

****************Posicion 133 ******************
mA1@1_pmA1     PARAMETRO      POS_PAR: 0     POS_LOCAL: 0   CLASE: ESCALAR TIPO: ENTERO
DIR: 0  ACCESO: 3      MIEMBRO: 2

****************Posicion 139 ******************
AA_mA1@1       METODO SOBREESCRIBIBLE       TIPO: BOOLEAN  CLASE: (null)  #PAR: 0 #LOCAL: 0
       ACCESO: 3      MIEMBRO: 2

=================== AA =================

****************Posicion 64 ******************
AA_sa1 ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0 ACCESO: 1      MIEMBRO: 1

****************Posicion 139 ******************
AA_mA1@1       METODO SOBREESCRIBIBLE       POS METODO: 0 Y ACUMULADA 0  TIPO: BOOLEAN
CLASE: (null)  #PAR: 0 #LOCAL: 0      ACCESO: 3      MIEMBRO: 2

****************Posicion 144 ******************
AA_AA  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 160 ******************
AA_a1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO  DIR: 0  ACCESO: 3      MIEMBRO: 2

****************Posicion 161 ******************
AA_a2  ATRIBUTO CLASE CLASE: ESCALAR TIPO: ENTERO   DIR: 0 ACCESO: 3      MIEMBRO: 1
buscar id_no_cualificado AA a1: Encontrado en AA
buscar id_no_cualificado AA v1: Encontrado en main
buscar id_no_cualificado AA pmA1: Encontrado en mA1@1
cerrar_ambito_tsc AA.
cerrar_clase AA.
```

```
buscar declarar_miembro_instancia BB BB_b1: No encontrado: se puede declarar
insertar_tsc BB BB_b1 8 1 1 3 2

=================== BB =================

****************Posicion 148 ******************
BB_BB  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 163 ******************
BB_b1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0  CLASE: ESCALAR TIPO:
ENTERO DIR: 0 ACCESO: 3     MIEMBRO: 2
buscar declarar_miembro_instancia BB BB_aa1: No encontrado: se puede declarar
insertar_tsc BB BB_aa1 8 0 1 3 2

=================== BB =================

****************Posicion 48 ******************
BB_aa1 ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0  CLASE: ESCALAR TIPO:
(null) DIR: 0 ACCESO: 3     MIEMBRO: 2

****************Posicion 148 ******************
BB_BB  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 163 ******************
BB_b1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0  CLASE: ESCALAR TIPO:
ENTERO DIR: 0 ACCESO: 3     MIEMBRO: 2
buscar declarar_miembro_instancia BB BB_mB1@1: No encontrado: se puede declarar
abrir_ambito_tsc BB BB_mB1@1 5 1 3 2

=================== mB1@1 =================

****************Posicion 142 ******************
BB_mB1@1       METODO SOBREESCRIBIBLE       TIPO: ENTERO  CLASE: (null)  #PAR: 0 #LOCAL: 0
       ACCESO: 3     MIEMBRO: 2

=================== BB =================

****************Posicion 48 ******************
BB_aa1 ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0  CLASE: ESCALAR TIPO:
(null) DIR: 0 ACCESO: 3     MIEMBRO: 2

****************Posicion 142 ******************
BB_mB1@1       METODO SOBREESCRIBIBLE       POS METODO: 0 Y ACUMULADA 0  TIPO: ENTERO
CLASE: (null)  #PAR: 0 #LOCAL: 0     ACCESO: 3     MIEMBRO: 2

****************Posicion 148 ******************
BB_BB  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


****************Posicion 163 ******************
BB_b1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0  CLASE: ESCALAR TIPO:
ENTERO DIR: 0 ACCESO: 3     MIEMBRO: 2
buscar declarar_id_local_metodo BB mB1@1_pmB1: No encontrado: se puede declarar
insertar_tsc BB mB1@1_pmB1 2 1 1 3 2

=================== mB1@1 =================

****************Posicion 135 ******************
mB1@1_pmB1     PARAMETRO      POS_PAR: 0     POS_LOCAL: 0  CLASE: ESCALAR TIPO: ENTERO
DIR: 0 ACCESO: 3     MIEMBRO: 2

****************Posicion 142 ******************
BB_mB1@1       METODO SOBREESCRIBIBLE       TIPO: ENTERO  CLASE: (null)  #PAR: 0 #LOCAL: 0
       ACCESO: 3     MIEMBRO: 2
```

```
================== BB ================

***************Posicion 48 *****************
BB_aa1 ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
(null) DIR: 0 ACCESO: 3      MIEMBRO: 2

***************Posicion 142 *****************
BB_mB1@1       METODO SOBREESCRIBIBLE       POS METODO: 0 Y ACUMULADA 0   TIPO: ENTERO
CLASE: (null)  #PAR: 0 #LOCAL: 0      ACCESO: 3      MIEMBRO: 2

***************Posicion 148 *****************
BB_BB  CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO SOBR. 0
ACUM ATR INS Y 0 ACUM MET SOBR.


***************Posicion 163 *****************
BB_b1  ATRIBUTO INSTANCIA    POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR TIPO:
ENTERO DIR: 0 ACCESO: 3      MIEMBRO: 2
buscar id_cualificado_instancia aa1 a1 BB: Encontrado en AA
buscar id_cualificado_instancia aa1 sa1 BB: Encontrado en AA
cerrar_ambito_tsc BB.
cerrar_clase BB.
buscar declarar_main main_a: No encontrado: se puede declarar
insertar_tsa_main main_a 1 0 1 3 2

================== main =================

***************Posicion 50 *****************
main_v1 VARIABLE       POS_LOCAL: 0   POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR
TIPO: ENTERO   DIR: 0 ACCESO: 3      MIEMBRO: 2

***************Posicion 93 *****************
main_main      CLASE  ES CLASE CON 0 ATR CLASE, 0 ATR INSTANCIA, 0 MET. SOBR. 0 MET. NO
SOBR. 0 ACUM ATR INS Y 0 ACUM MET SOBR.


***************Posicion 191 *****************
main_a VARIABLE       POS_LOCAL: 0   POS ATR. INSTANCIA 0 Y ACUMULADA 0   CLASE: ESCALAR
TIPO: (null)   DIR: 0 ACCESO: 3      MIEMBRO: 2
buscar id_no_cualificado main a: Encontrado en main
buscar id_no_cualificado main v1: Encontrado en main
buscar id_no_cualificado main v1: Encontrado en main
buscar id_cualificado_instancia a mA1@1 main: Encontrado en AA
buscar id_no_cualificado main v1: Encontrado en main
buscar id_cualificado_instancia a a1 main: Encontrado en AA
buscar id_no_cualificado main x: No encontrado
buscar id_cualificado_clase AA a1 main: Encontrado en AA
buscar id_cualificado_clase AA sa1 main: Encontrado en AA
buscar id_no_cualificado main a: Encontrado en main
cerrar_tsa_main.
cerrar_tsc.
```

Desarrollo de la práctica:

1. Codificación de la librería TS

A lo largo de las últimas semanas habrás desarrollado una serie de módulos que conforman tu TS.

Asegúrate de que dispones de unas funciones de alto nivel que hacen la misma funcionalidad descrita en estas páginas.

Se te pedirá que realices un fichero .zip comprimiendo todas ellas y escribas un makefile que con la etiqueta TS genere un .o (con símbolos de depuración, para ello utiliza -g al compilar).

Si no quieres utilizar un makefile entonces asegúrate que al disponer juntos todos los ficheros de tu práctica en el mismo directorio y sin ninguna estructura adicional más (tanto los de tu TS como los del programa principal) se obtendrá un ejecutable de nombre prueba_TS cuando se ejecute el comando

gcc Wall -g -o prueba_TS *.c

2. Escritura de un programa de prueba

Debes escribir un programa en lenguaje C llamado prueba_TS.c que realice

- La lectura de un fichero de entrada proporcionado como primer argumento que contenga las indicaciones de actividad solicitada a la TS.
- La escritura de un fichero de salida proporcionado como segundo argumento que contenga la salida de tu programa al fichero de entrada.
- El resto de mensajes que quieras mostrar serán ignorados y los deberás hacer a cualquier otro sitio que no modifique el compromiso de que prueba_TS sólo tiene dos argumentos con el significado que se acaba de mencionar.

Entrega de la práctica:

Se entregará a través de Moodle un único fichero comprimido (*.zip*) que deberá cumplir los siguientes requisitos:

- Contener todos los ficheros descritos previamente
- Su nombre deberá identificar inequivocamente el grupo que lo entrega ya sea con el número de grupo o con los apellidos de los integrantes

grN_TS.zip o
Apellido1_Apellido2_...ApellidoN_TS.zip

Anexo:

A continuación se te muestra una serie de constantes que se corresponden con los conceptos vistos en clase y donde puedes encontrar los códigos numéricos utilizados en alguna de las indicaciones de operación de entrada que aparecen en el enunciado.

No es obligatorio que utilices los mismos nombres pero sí que los valores numéricos se correspondan con estos

```
#define MAX_LONG_ID 50
#define MAX_TAMANIO_VECTOR 64
```

```
#define MAX_FILAS_VECTOR 64
#define MAX_COLUMNAS_VECTOR 64
#define MAX_CAPACIDAD_CONJUNTO 64

/* CATEGORIAS */
#define VARIABLE 1
#define PARAMETRO 2
#define FUNCION 3
#define CLASE 4
/* PARA OMICRON */
#define METODO_SOBREESCRIBIBLE 5
#define METODO_NO_SOBREESCRIBIBLE 6
#define ATRIBUTO_CLASE 7
#define ATRIBUTO_INSTANCIA 8

#define NINGUNO 0 /* ES COMUN PARA ACESO Y TIPO DE MIEMBRO */

#define ACCESO_SECRET          1     // SECRET
#define ACCESO_HIDDEN          2     // HIDDEN: SOLO EN LA CLASE
#define ACCESO_EXPOSED         3     // EXPOSED O NADA

#define MIEMBRO_UNICO          1  /* PARA ATRIBUTOS DE CLASE Y MÉTODOS NO
SOBREESCRIBIBLES */
#define MIEMBRO_NO_UNICO 2  /* PARA ATRIBUTOS DE INSTANCIA Y MÉTODOS
SOBREESCRIBIBLES */

/* CLASES DE DATOS */
#define ESCALAR 1
#define PUNTERO 2
#define VECTOR 3
#define CONJUNTO 4
/* PARA CLASES DE DATOS QUE SE CORRESPONDEN CON OBJETOS */
#define OBJETO 5
/* PARA CLASES DE DATOS QUE SE CORRESPONDEN CON OBJETOS*/

/* TIPOS */
#define INT 1
#define FLOAT 2
#define BOOLEAN 3
/* PARA CLASES */
/* LAS CLASES UTILIZARAN COMO SU TIPO -1 * INDICE EN EL VECTOR DE CLASES
LINEALIZADO DEL GRAFO DE CLASES */
/* PARA CLASES */

#define ERR -1
#define OK 0

#define TOK_ERROR -1
```