4th Assignment: SYMBOL TABLE

Antes de las sesiones del 30 de noviembre, 1, 2, y 7 de diciembre de cada grupo

Goal of the assignment

The goal of the practice is the programming of the compiler's symbol table that will be developed during the course. Together with the symbol table, a test program will also be coded to check the proper functioning of the programmed symbol table.

Development of the assignment

The scheme of development of this practice will be maintained in most of the other practices of the course.
First the student will develop in C the concrete code of the practice, in this case, the table of symbols.
Secondly, the student will develop a test program that will allow to check the correct functioning of the developed code, in this case the symbol table. The specifications of this test program will be detailed in the practice statement.
Finally, optionally, depending on the subject of the practice, the student will have a set of test cases that will allow him/her to partially check the functioning of his/her code.

Coding of a library for the symbol table

The compiler's symbol table developed during the course should provide the correct management of domains. Given the characteristics of the ALFA language, it can be said that
- When the compiler is compiling code outside a function, there is only **one scope** (the global one).
- When the compiler is compiling a function, there are **two open scopes** (the global scope and the local scope of the function).

That is, at any given moment of the compilation, there can be a maximum of two open domains.

A possible solution for the management of scopes in ALFA is to have **two tables**, one for storing the identifiers of the global scope, and the other for storing the local identifiers of the functions.
In order to obtain the shortest possible compilation time, it is common to use **hash tables** for the implementation of the compilers' symbol tables. The student already knows this type of data, the algorithms for its management and its performance.

The student must write the files and C modules that he considers necessary for the definition and implementation of a hash table. Later, the compiler's symbol table will consist of the combination of two of these hash tables.

The information that is stored in the symbol table for each program identifier can be found in the documentation corresponding to the working sessions of this practice.

Coding a test program

Once the code corresponding to the hash table has been developed, the student will write in C a test program *(tablaSimbolos.c)* that allows to simulate the operation of the compiler's symbol table using two hash tables, one for the global scope, and another for the local one.

The objective is to develop a program that allows to check the correct functioning of the symbol table. This correction refers to three aspects:

- Scope management
- Insertion of elements
- Search for items

The program will read from an input file operations of opening/closing scopes and insertion/searching for elements, carry out these operations and write in an output file the result of carrying out each of these operations.

It is assumed that:

- The symbol table is made up of two hash tables, one in which the global scope identifiers (global variables and function names) are stored, which from now on will be called **global table**, and another one for the local scope identifiers (local variables of the function and parameters of the function), which will be called **local table**.
- Initially the global scope is open, that is, any insertion or search must be made in the table of that scope (the global table).
- The opening of a new scope (corresponding to a function) implies several actions:
    - First, the name of the new domain must be inserted in the global table.
    - Then the local area must be opened.
    - Finally, the name of the new domain must be inserted in its own local table.
- **Closing an area** implies "emptying" the local table.
- An insertion operation ends successfully if the identifier to be inserted does not exist in the current scope.
- A **search operation** ends successfully if the identifier is found in one of the open domains (the local table is always searched first).
- The **insertion operation**s use two parameters, the identifier of the element to be inserted and a number greater than or equal to 0, which can be stored in any of the fields, for example in the "type" field.
- The search operations that end successfully retrieve the value associated with the identifier (the number stored in the "type" field).

This test program will be invoked with two parameters:

- The first parameter is the name of the input file.
- The second parameter is the name of the output file.

The description of the input/output files is shown below.

Input file description

The format of the input file is designed to allow testing of the correct functioning of the insertion and search operations in the symbol table together with the correct management of the identifier visibility ranges.

The input file consists of a set of lines, each of which represents an action on the symbol table. These actions may be:

- **Insertion of an element**
- **Search for an item**
- **Opening of an area**
- **Closing an area**

Each line in the input file indicates an action on the symbol table as listed below. It can be seen that each line contains a string of characters followed optionally by a whole number (separated by a tab):

- **<scope ID> <integer less than -1>.** It means that a new field begins with field identifier <field identifier>. The program should open a new scope with all the actions that this implies (described above).
- **close -999**. This means that the active range is closed.
- **<identifier> <int greater than or equal to 0>.** It means that an attempt will be made to insert the identifier <identifier> (and the value <whole greater than or equal to 0> will be associated with it).
- **<identifier>.** It means that the table will be searched for the identifier <identifier>.

The following table shows an example of the operation of the test program on a possible input file. The first column shows the lines of the input file. The second column shows the actions that the test program performs on the symbol table for each line of the input file:

Input file line	Action on the symbol table
uno 1	Attempt to insert in the element's global table (uno;1) The insertion would be successful because the element is not
dos 2	Attempt to insert in the element's global table (dos;2) The insertion would be successful because the element is not
uno 1	Attempt to insert in the element's global table (uno;1) The insertion would be failure because the element is already in the table
dos 2	Attempt to insert in the element's global table (dos;2) The insertion would be failure because the element is already in the table
uno	Search in the element table (uno) The insertion would be successful because the element is there

dos	Search in the element table (dos) The insertion would be successful because the element is there
cuatro	Search in the element table (cuatro) The search would end with a failure because the element is not there
cinco	Search in the element table (cinco) The search would end with a failure because the element is not there
funcion1 -10	Openness of scope (funcion1) The opening would be successful because in the global table does not exist such function The element is inserted in the global table, the local table is initialized and the element is inserted in the local table
uno 10	Attempt to insert in the element's local table (uno;10) The insertion would be successful because the element is not there
uno 10	Attempt to insert in the element's local table (uno;10) The insertion would end with failure because the element is already there
funcion1 100	Attempt to insert in the element's local table (funcion1;100) The insertion would end with failure because the element is already there
tres 3	Attempt to insert in the element's local table (tres;3) The insertion would be successful because the element is not there
uno	Search in the element table (uno) The insertion would be successful because the element is there
funcion1	Search in the element table (funcion1) The insertion would be successful because the element is there
tres	Search in the element table (tres) The insertion would be successful because the element is there
dos	Search in the element table (dos) The insertion would be successful because the element is there
cuatro	Search in the element table (cuatro) The search would end with a failure because the element is not
cierre -999	The local area is closed The local table is "emptied". The operation ends successfully
funcion1	Search in the element table (funcion1) The insertion would be successful because the element is not

The aim of the test program is to check that the hash tables work correctly and that the areas are properly managed in terms of insertion and search operations. It is beyond the scope of the test program to check that the input file contains, for example, successive operations to open scopes, since this situation is

impossible in ALFA and therefore will never appear in the input file. Nor will scope closures appear in the input file when they have not been previously opened.

Output file description

The result of each request in the input file is recorded in the output file.

In the output file, it must be possible to distinguish between the following four situations

- Successful insertion: insertion of identifiers that are not in the current scope.
- Unsuccessful insertion: insertion of identifiers that are already in the current scope.
- Successful search: search for identifiers that are in one of the open areas.
- Unsuccessful search: search for identifiers that are not in any of the open areas.

Each line of the output file shows the result of the requested action on the corresponding line of the input file according to the following formats:

- `<identifier>` to indicate that the identifier `<identifier>` has been successfully inserted in the table since it was not previously there.
- `-1 <identifier>` to indicate that the insertion of `<identifier>` has been requested but could not be completed successfully since the identifier was already in the table as a result of a previous insertion.
- `<identifier> <int>` to indicate that the search for the `<identifier>` has been requested and has been successfully completed since it had previously been inserted with the associated value `<int>`.
- `<identifier> -1` to indicate that the search for the `<identifier>` has been requested but has not been completed successfully since it has not been inserted in the table.
- `close` to indicate that the local area has been closed.

The test program, **for each line of the input file,** generates a new line in the output file according to the rules explained above. In the following table the input and output files corresponding to the previous example are shown together.

Input file line	Output file line	Action
uno 1	uno	Successful insertion
dos 2	dos	Successful insertion
uno 1	-1 uno	Insertion failure
dos 2	-1 dos	Insertion failure
uno	uno 1	Successful search
dos	dos 2	Successful search
cuatro	cuatro -1	Search failure

cinco	cinco -1	Search failure
funcion1 -10	funcion1	Openness of scope local
uno 10	uno	Successful insertion
uno 10	-1 uno	Insertion failure
funcion1 100	-1 funcion1	Insertion failure
tres 3	tres	Successful insertion
uno	uno 10	Successful search
funcion1	funcion1 -10	Successful search
tres	tres 3	Successful search
dos	dos 2	Successful search
cuatro	cuatro -1	Search failure
cierre -999	cierre	Area closure
funcion1	funcion1 -10	Successful search

7. Examples

Try the inputs and outputs provided in this assignment (they are not in Moodle, but you can create them from the tables above). Assuming your executable file is named *tablaSimbolos*, a command like:

```
pruebaSintactico entrada.in misalida.out
diff -bB salida.out misalida_.out
```

Deliverable

The student will deliver through moodle a compressed file (.zip) that must meet the following requirements:

- The leader of the group must submit always the assignments
- It must contain all the sources (.h and .c files) necessary to solve the proposed statement.
- It must contain a Makefile compatible with the make tool that generates the executable named *tablaSimbolos*.
- The name of the zipped file will be:

Apellido1_Name1_Apellido2_Name2_Apellido3_Name2.zip

The names will be written in alphabetical order. The names cannot contain spaces, accent marks or the letter ñ.

You are warned that the assignments of this course will compose the final compiler you have to write. **For this reason the possible mistakes of each single solution (even if they pass) could generate serious mistakes in further stages of the compiler. It is your personal responsibility to fully fix all the mistakes, asking for help, if necessary, to your teacher if you feel unsure about the possible solution.**