Universidad Nacional Autónoma de México

Facultad de Ingeniería

Práctica 5

Tabla de Tipos y Tabla de Símbolos

ALUMNOS:

CÁRDENAS CÁRDENAS JORGE
MURRIETA VILLEGAS ALFONSO
REZA CHAVARRIA SERGIO GABRIEL
VALDESPINO MENDIETA JOAQUIN

PROFESOR:

ADRIAN ULISES MERCADO MARTINEZ

1. Objetivo

- Implementar en lenguaje C una tabla de tipos y una tabla de símbolos, una lista para guardar el tipo de argumentos, una tabla para la tabla de símbolos y una pila para la tabla de tipos

2. Introducción

La tabla de símbolos es una estructura de datos que almacena la información de los identificadores, como: tipo de dato para la variable, la dirección, el tipo de variable (Función, parámetro o o variable), ámbito (alcance); en caso de ser función debe almacenar también la lista de parámetros y el número de parámetros.

La tabla de tipos es una estructura que almacena la información sobre los tipos nativos y compuestos del lenguaje de programación. Los tipos definidos por el usuario son los arreglos o las estructuras. La información que guarda de cada tipo es: identificador del tipo, nombre del tipo, tamaño en bytes del tipo, tipo base del tipo y el n ´umero de elementos en caso de ser un arreglo.

3. Desarrollo

A continuación, se mostrarán cada uno de los elementos que se pidió desarrollar para esta práctica, además de sus correspondientes pruebas de escritorio:

3.1. Tabla de Símbolos

A continuación se describen brevemente cada uno de los métodos que contiene la biblioteca de la tabla de símbolos:

1) Función crearParam(int tipo): La función obtiene un entero y reserva el espacio de memoria y le asigna al atributo de tipo del parámetros el tipo.

2) Función borrarParam(param *p): Función que liberará el espacio de memoria de un parametro, el cual es obtenido por la dirección de este.

3) Función crearLP(): La función crea un espacio de memoria para la lista de parámetros, siendo que devuele el apuntador a la lista.

4) Función add(listParam *lp , int tipo): Función que agrega un nuevo parámetro. Revisará si la lista esta vacía o no. Si está vacía el nuevo elemento será agregado y se le asignará como raíz de la lista. En caso contrario la función recorrerá la lista hasta el último elemento, al está en la última posición se creará el parámetro y será asignado al next del último parámetro.

5) Funcion borrarListParam (listParam *lp): Función que recorre la lista, cada vez que recorra la lista elimina el valor del tipo y la referencia del siguiente elemento, con ayuda de una variable de apoyo.

6) Función getNumListParam (listParam *lp): Función que devuelve la cantidad de parámetros en la lista.

7) Función print_list(listParam lista): Recorrido e impresión de la lista recibida.

8) Función *crearSymbol(int dir, int tipo,int tipovar, char id[32],listParam *params): Función que devuelve el espacio de memoria de un símbolo y le asigna los valores como el tipo, el id, la dirección y la lista de parámetros.

9) Función borrarSymbol(symbol *s): Función que elimina la información del símbolo obtenido.

10) Función *crearSymTab(): Función que devuelve el espacio de memoria de una nueva tabla de símbolos.

11) Función insertar(symtab *st, symbol *sym): Ingresa el nuevo símbolo a la lista. Existen 3 casos, esto es dependiendo del estado de la tabla. Si la lista está vacía, al nuevo símbolo será asignado como la raíz. En caso que contenga elementos, la lista será recorrida, cada vez que recorra comparará el id del elemento actual con la del nuevo, en caso de que sean iguales devolverá -1 y no lo asignará a la lista, en caso de que el id sea nuevo lo asignará al final y devolverá el índice de su posición.

12) Funciones buscar(symtab *st,char *id): Revisará si existe un simbolo en la lista con el id requerido

13) Funciones getTipo(symtab *st, char *id), getTipoVar(symtab *st, char *id), getTipoVar(symtab *st, char *id), getDir(symtab *st, char *id), getListParam (symtab *st,char *id): Obtención de los atributos actuales de la tabla de símbolos con la ayuda de la comparación del id.

14) Funcion print_symtab(symtab *st): Impresión de la información de la tabla de símbolos escogida.

A continuación se muestra la prueba de escritorio de este apartado:

```
C:\Users\Sergio\OneDrive\NewCompiladores\EstructurasPractica5>gcc tablaSimbolos.c tablaTipos.c pilaTablaSimbolos.c pilaTab
laTipos.c Main.c -o Pruebas

C:\Users\Sergio\OneDrive\NewCompiladores\EstructurasPractica5>Pruebas
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
Prueba de la tabla de simbolos
Parametro 1: 10
Parametro 2: 5
Lista de parametros
Numero de parametros 3
LA LISTA ESTA VACIA
```

Imagen 1: Compilación y creación de parámetros y listas de parámetros.

```
Creacion de simbolos

Direccion: 0
Tipo no. :1
Tipo de valor: 2
ID: prueba

Los elementos de la lista son:
4, 5,
Direccion: 4
Tipo no. :4
Tipo de valor: 5
ID: seraBorrado

Los elementos de la lista son:
4, 5,
No hay simbolo existenteno agregado
Simbolo no agregado
```

Imagen 2: Creación de Símbolos.

```
Tabla de simbolos despues de la insercion
0 Simbolo:

Direccion: 9
Tipo no. :3
Tipo de valor: 10
ID: Nuevo elemento 0

Los elementos de la lista son:
4, 5,
1 Simbolo:

Direccion: 10
Tipo no. :4
Tipo de valor: 2
ID: Nuevo elemento 1

Los elementos de la lista son:
4, 5,
2 Simbolo:

Direccion: 4
Tipo no. :4
Tipo de valor: 5
ID: Nuevo elemento 2

Los elementos de la lista son:
4, 5,
Lugar de Nuevo elemento 1: 1
```

Imagen 3: Creación de una tabla de Símbolos y su inicialización.

```
Lugar de Nuevo elemento 1: 1
Tipo de valor de  Nuevo elemento 2: 5
Direccion de Nuevo elemento 0: 4
Los elementos de la lista son:
4, 5,
```

Imagen 4: Prueba a las funciones que contienen las tablas de símbolos.

3.2. Tabla de Tipos

A continuación se describen brevemente cada uno de los métodos que contiene la biblioteca de la tabla de tipos:

1) Función crearTipo(char nombre [10], tipoBase tb, int tam, int elem,int id): Función para la asignación de espacio de memoria de un tipo, asignando los parametos de un tipo como id, tipo base tamaño, nombre.

2) Función borrarType(type *t): Libera la memoria del tipo obtenido.

3) Función crearTypeTab(): Creación del espacio de memoria para una nueva tabla de tipos.

4) Función insertarTipo(typetab *tt,type *t):

5) Funciónes getTipoBase(typetab *tt, int id), getTam(typetab *tt, int id), getNumElem(typetab *tt, int id), getNombre(typetab *tt, int id): Obtención de los atributos actuales de la tabla de símbolos con la ayuda de la comparación del id.

6) Función printT(typetab * tt): Impresión de la información de la tabla de tipos para corroborar que la información se guardó.

A continuación se muestra la prueba de escritorio de este apartado:

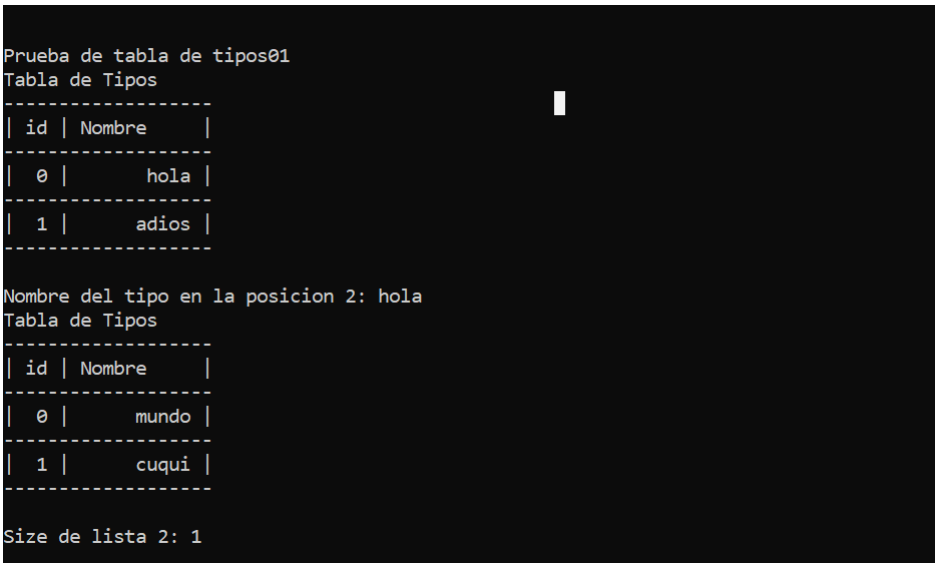


Imagen 5: Creación de 2 tablas con tipos de datos estáticos y creados por tablas de símbolos y junto con pruebas funciones de las tablas.

3.3. Pila de Tabla de Tipos

A continuación se describen brevemente cada uno de los métodos que contiene la biblioteca de la pila que contiene tabla de tipos:

1) Función crearTypeStack(): Función que devuelve la dirección de memoria reservada que genera la función para una pila de tabla de tipos.

2) Función borrarTypeStack(typestack *ts): Liberación de memoria de una pila de tabla de tipos.

3) Función insertarTypeTab(typestack *ts, typetab *sym): Inserción de una tabla de tipos a la pila, el nuevo elemento será el tope de este, o sea la raíz de la lista.

4) Función getCimaType(typestack *ts): Obtención de la dirección del tope de la pila, sin que este sea eliminado de la pila.

5) Función sacarTypeTab(typestack *ts): Obtención de la dirección del tope actual y la cima de la pila será actualizada.

6) Función isEmptyTypeStack(typestack *p): Revisará el tamaño de la pila, si el tamaño es 0 devolverá 1. Caso contrario devolverá 0.

A continuación se muestra la prueba de escritorio de este apartado:

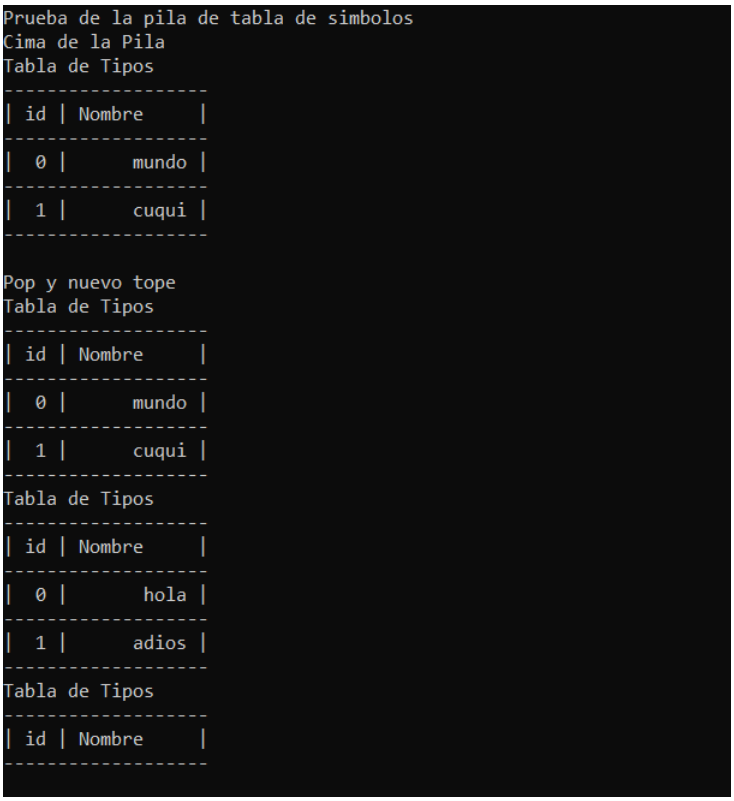


Imagen 6: Creación de la pila y el uso de las funciones de push y pop de la pil

3.4. Pila de Tabla de Símbolos

A continuación se describen brevemente cada uno de los métodos que contiene la biblioteca de la pila que contiene tabla de símbolos:

1) Función crearSymStack(): Creación de la nueva pila de símbolos.

2) Funcion borrarSymStack(symstack *st): Liberación de memoria de toda la pila obtenida en la función como parametro.

3) Función insertarSymTab(symstack *st,symtab *sym): Inserción de la tabla de tipos obtenida y la agregará en la cima de la pila de símbolos.

4) Función getCima(symstack *ss): Obtención de la dirección del tope de la pila, sin que este sea eliminado de la pila.

5) Función sacarSymTab(symstack *ss): Obtención de la dirección del tope actual y la cima de la pila será actualizada.

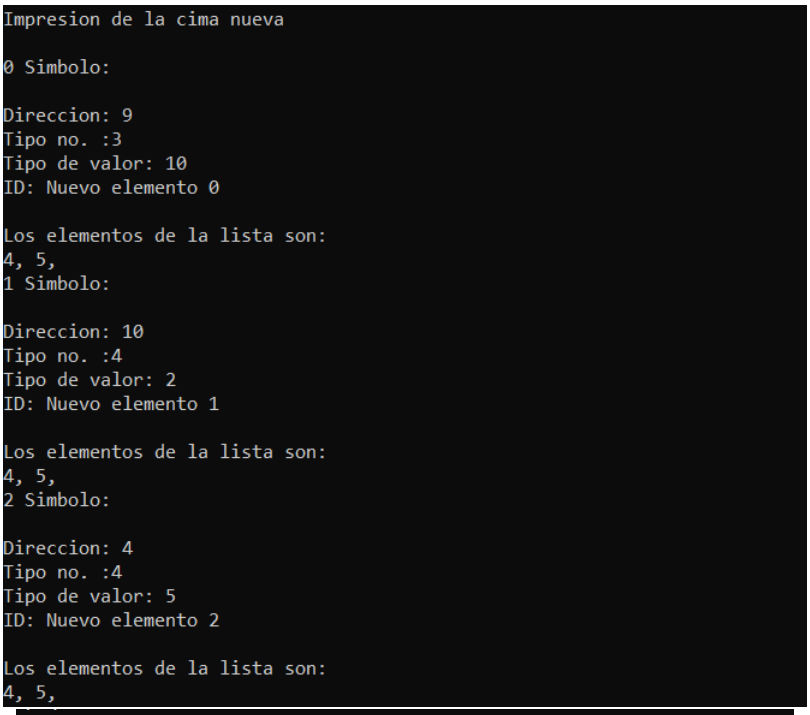
6) Función isEmpty(symstack *p): Revisará el tamaño de la pila, si el tamaño es 0 devolverá 1. Caso contrario devolverá 0.

A continuación se muestra la prueba de escritorio de este apartado:

```
Prueba de pila de tabla de simbolos
Impresion de la cima despues de ingresar las 2 tablas
0 Simbolo:

Direccion: 7
Tipo no. :3
Tipo de valor: 10
ID: ELE

Los elementos de la lista son:
4, 5,
1 Simbolo:

Direccion: 1
Tipo no. :4
Tipo de valor: 23
ID: MENTO

Los elementos de la lista son:
4, 5,
2 Simbolo:

Direccion: 2
Tipo no. :6
Tipo de valor: 5
ID: SOLO
```

Imagen 7: Creación de la pila de la tabla de símbolos e inserción de tablas de símbolos.

```
Los elementos de la lista son:
4, 5,
Pop a la pila

0 Simbolo:

Direccion: 7
Tipo no. :3
Tipo de valor: 10
ID: ELE

Los elementos de la lista son:
4, 5,
1 Simbolo:

Direccion: 1
Tipo no. :4
Tipo de valor: 23
ID: MENTO

Los elementos de la lista son:
4, 5,
2 Simbolo:

Direccion: 2
Tipo no. :6
Tipo de valor: 5
ID: SOLO
```



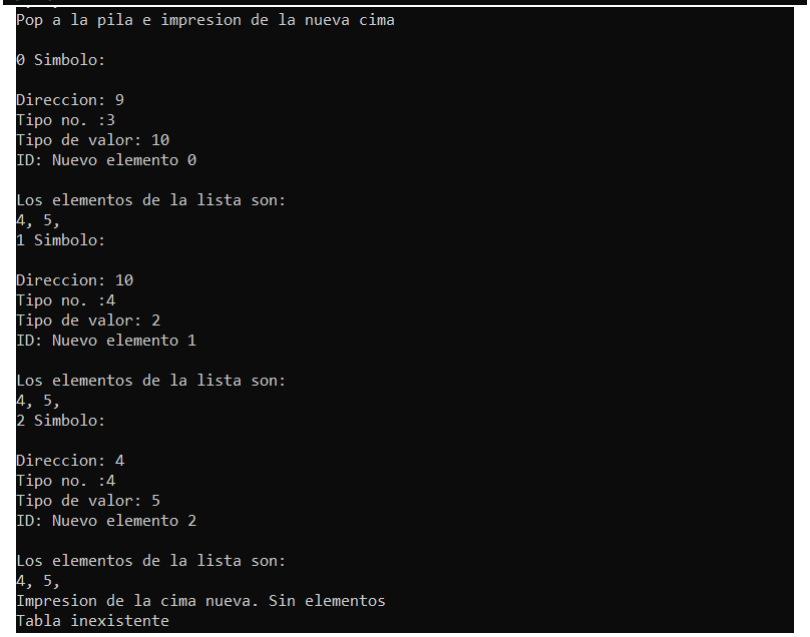


Imagen 8, 9 y 10: Uso de la función Pop y obtención de la cima

4. Conclusiones

4.1. Cárdenas Cárdenas Jorge

Con la presente practica realizamos la implementación tanto de las tablas de símbolos y tipos, como de las pilas necesarias para que el compilador funcione en base a los distintos ámbitos que una variable puede tener dentro de un programa en un lenguaje de programación.

La implementación es especifico de estas estructuras en el lenguaje de programación C, representó para el equipo una tarea un tanto compleja al vernos en la necesidad de implementar desde cero las estructuras necesarias (listas ligadas, pilas, listas de listas, etc.) a partir de apuntadores y memoria dinámica.

4.2. Murrieta Villegas Alfonso

En la presente práctica a través de nuestros conocimientos de fundamentos de programación y estructura de datos y algoritmos aplicamos estructuras de datos como son pilas (Stack) y listas (List) para el manejo e implementación de dos estructuras fundamentales en el manejo de símbolos y tipos de datos en el análisis semántico de nuestro futuro compilador.
Cabe destacar que al utilizar el lenguaje C en esta práctica realmente partimos desde ceros para poder crear estas estructuras de datos, desde el crear las estructuras como la de tipo de dato o símbolos hasta estructuras que utilizaron a estas estructuras como las pilas de tablas de símbolos y tabla de tipos, además cabe destacar la importancia de los apuntadores y el manejo de memoria dinámica para la creación de estas estructuras.

4.3. Reza Chavarria Sergio Gabriel

Con el concepto visto en clase del manejo de los símbolos y de los diferentes tipos de datos que se manejar en el análisis semántico de código a compilar el uso de las listas y pilas de estos son importante e indispensables. Al tener estas estructuras de código se obtendrá un manejo más simple para su uso posterior en el análisis. Se tiene que tener un conocimiento básico acerca de las estructuras de datos básicos como lo fueron las pilas, las listas ligadas, las listas de listas, estructuras para poder llevar a cabo la base para el análisis para un compilador.

4.4. Valdespino Mendieta Joaquín

En la presente practica se ha logrado implementar los conocimientos de estructura de datos y del lenguaje C, mediante el uso de memoria dinámica, apuntadores, para las pilas y listas ligadas, esto con el fin de obtener el código y estructura necesaria para trabajar con tablas de símbolos y tipos, además de la pila de tablas de tipos y pila de tabla de símbolos para la resolución de los ámbitos de las variables. esto como parte del desarrollo de nuestro compilador, ya que son esenciales para la parte de análisis sintáctico y semántico.