Universidad Nacional Autónoma de México
Facultad de Ingeniería
Compiladores
Práctica Tabla de símbolos y tabla de tipos
Profesor: Adrián Ulises Mercado Martínez

Tabla de símbolos y tabla de tipos

Objetivo:

Implementar en lenguaje C una tabla de símbolos, una tabla de tipos, una lista para guardar el tipo de los argumentos, una pila para tablas de símbolos y una pila para tablas de tipos.

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 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 número de elementos en caso de ser un arreglo.

Ejercicios

1. Implementar la tabla de símbolos. Puede basarse en la siguiente estructura

```
typedef struct _param param;
struct _param{
  int tipo;
  param *next;
};

typedef struct _listParam listParam;

struct _listParam{
  param *root;
  int num;
};

/* Retorna un apuntador a una variable Param */
param *crearParam(int tipo);
/* Borra param, libera la memoria */
void borraParam(param *p);
/* Retorna un apuntador a una variable listParam */
listParam *crearLP();
/* Agrega al final de la lista el parametro  e incrementa num*/
void add(listParam lp, int tipo);
/* Borra toda la lista, libera la memoria */
void borrarListParam(listParam* lp);
/* Cuenta el numero de parametros en la lista */
int getNumListParam(listaParam *lp);

typedef struct _symbol symbol;

struct _symbol{
  char id[32];
  int tipo;
  int dir;
  int tipoVar;
  listParam *params;
  symbol *next;
};
```

```
/* Retorna un apuntador a una variable symbol */
symbol *crearSymbol();
/* Borra symbol, libera la memoria */
void borrarSymbol(symbol *s);

typedef struct _symtab symtab;

struct _symtab{
  symbol *root;
  int num;
  symtab *next;
};

/* Retorna un apuntador a una variable symtab,
 * inicia contador en 0
 */
symtab *crearSymTab();
/* Borra toda la lista, libera la memoria */
void borrarSymTab(symtab* st);
/* inserta al final de la lista en caso de insertar incrementa num
 * retorna la posicion donde inserto en caso contrario retorna -1
 */
int insertar(symtab *st, symbol *sym);
/* Busca en la tabla de simbolos mediante el id
 * En caso de encontrar el id retorna la posicion
 * En caso contrario retorna -1
 */
int buscar(symtab *st, char *id);
/* Retorna el tipo de dato de un id
 * En caso de no encontrarlo retorna -1
 */
int getTipo(symtab *st, char *id);
/* Retorna el tipo de Varibale de un id
* En caso de no encontrarlo retorna -1
*/
int getTipoVar(symtab *st, char *id);
/* Retorna la direccion de un id
* En caso de no encontrarlo retorna -1
*/
int getDir(symtab *st, char *id);
/* Retorna la lista de parametros de un id
* En caso de no encontrarlo retorna NULL
*/
listParam *getListParam(symtab *st, char *id);
/* Retorna el numero de parametros de un id
* En caso de no encontrarlo retorna -1
*/
int getNumParam(symtab *st, char *id);
```

2. Implementar la tabla de tipos. Puede basarse en la siguiente estructura

```
#include <stdbool.h>
typedef struct _type type;
typedef struct _tipoBase tipoBase;
typedef union _tipo tipo;

union _tipo{
  int type; //Tipo simple
  symtab *estructura; // Tipo estructura
};

struct _tipoBase{
  bool est; // Si est es verdadero es estructura sino es tipo simple
  tipo t;
};

struct _type{
  int id;
  char nombre[10]; // se puede sustituir por un entero tambien
  tipoBase tb;
  int tamBytes;
  int numElem;
  type *next;
};
```

```
typedef struct _typetab typetab;

struct _typetab{
  type *root;
  int num;
};

/* Retorna un apuntador a una variable type */
type *crearTipo();
/* Borra type, libera la memoria */
void borrarType(type *t);
/* inserta al final de la lista en caso de insertar incrementa num
 * retorna la posicion donde inserto en caso contrario retorna -1
 */
int insertarTipo(typetab *tt, type *t);
/* Retorna el tipo base de un tipo
 * En caso de no encontrarlo retorna NULL
 */
TipoBase getTipoBase(typetab *tt, int id);
/* Retorna el numero de bytes de un tipo
 * En caso de no encontrarlo retorna -1
 */
int getTam(typetab *tt, int id);
/* Retorna el numero de elementos de un tipo
 * En caso de no encontrarlo retorna -1
 */
int getNumElem(typetab *tt, int id);
/* Retorna el nombre de un tipo
 * En caso de no encontrarlo retorna NULL
 */
char* getNombre(typetab *tt, int id);
```

3. Implementar la pila para las tablas de tipos. Puede basarse en la siguiente estructura

```
typedef struct _typetab typetab;

struct _typetab{
  type *root;
  int num;
  type *next;
};


typedef struct _typestack typestack;

struct _typestack{
  typetab *root;
  int num;
};

typestack *crearTypeStack();
void borrarTypeStack(typestack *ts);
void insertarTypeTab(typetab *sym);
typetab* getCimaType(typestack *ts);
typetab* sacarTypeTab(typetack *ts);
```

4. Implementar la pila para las tablas de símbolos. Puede basarse en la siguiente estructura

```
typedef struct _symtab symtab;

struct _symtab{
  symbol *root;
  int num;
  symtab *next;
};


typedef struct _symstack symstack;

struct _symstack{
  symtab *root;
```

```
  int num;
};

symstack *crearSymStack();
void borrarSymStack();
void insertarSymTab(symtab *sym);
symtab* getCima(symstack *ss);
symtab* sacarSymTab(symstack *ss);
```

5. Escriba sus conclusiones de forma individual