Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Estructuras de Datos y Algoritmos I

Trabajo Práctico

Alumna:

Sullivan, Katherine

Universidad Nacional de Rosario

2021

1 Dificultades encontradas

1.1 Cómo ir guardando los nombres de las expresiones

En la parte interactiva del programa se necesita tener un acceso rápido a una cantidad posiblemente extensa de nombres, para poder lograr esto se implementa una tabla de hash que sirve como diccionario para el acceso a los nombres de las expresiones. Para el manejo de colisiones de esta tabla de hash se hace uso del doble hashing (asegurando mantener la coprimalidad entre el valor en la segunda función de hash y la capacidad de la tabla). A continuación se muestra su estructura y dentro del archivo tablahash.c se puede ver su implementación.

```
typedef struct {
    CasillaHash* tabla;
    unsigned numElems;
    unsigned capacidad;
    FuncionHash hash;
    FuncionHash hash2;
} TablaHash;
```

con CasillaHash siendo la siguiente estructura

```
typedef struct {
  char* clave;
  Arbol dato;
  int estado;
} CasillaHash;
```

Nota: Si bien entiendo que para este momento de la cursada no vieron tablas de hash me pareció la mejor estructura para utilizar. Esto mismo se podría implementar con una lista enlazada o árbol pero los costos de inserción, eliminación y búsqueda se elevarían notablemente.

1.2 Manejo de errores en la creación del árbol de expresión

Resulto problemático el manejo de los distintos errores que podría tener la creación del árbol puesto que este podía ersultar mal estructurado o incompleto y perder memoria, si no se respetaba bien la notación postfija o la aridad de los operadores. Para resolver esto se decide la no creación del árbol y liberación de memoria cuando se notase que:

- por la aridad de un operador encontrado se necesitase extraer de la pila que contiene los nodos y no fuese posible, o que
- luego de haber completado la creación del árbol quedasen en la pila elementos (el árbol no está debidamente unido).

1.3 Impresión con paréntesis

Para resolver la impresión con la menor cantidad de paréntesis posible se decidió agregar un parámetro más a la estructura de operador (nodo de la tabla de operadores), quedando la estructura como se muestra a continuación:

```
typedef struct _NodoTablaOps {
  char* simbolo;
  int aridad;
  FuncionEvaluacion eval;
  int precedencia;
  struct _NodoTablaOps* sig;
} NodoTablaOps;
```

Luego para imprimir los paréntesis lo que la función asignada para la tarea realiza es, una vez encontrado un operador, comparar su precedencia con la del operador anterior, y si tiene menor precedencia imprime paréntesis.

2 Compilación y uso del programa

Para la compilación del programa la entrega cuenta con un archivo Makefile. Una vez ejecutado el comando make, se puede empezar a ejecutar el programa con el comando ./main.

Para el uso del programa se cuenta con 4 comandos principales:

- ALIAS = cargar EXPR: esta estructura de comando nos permite asignarle un nombre ALIAS a una expresión EXPR. La expresión debe cumplir con la notación postfija y con el formato de dejar un espacio cada vez que se escribe un operador o un operando.
- imprimir ALIAS: imprime la expresión con nombre ALIAS en notación infija con los paréntesis necesarios.
- evaluar ALIAS: devuelve el resultado de la expresión con nombre ALIAS.
- salir: cierra el programa.

Tener en cuenta que los espacios explicitados se deben respetar.

3 Bibliografía

3.1 Obras consultadas

- Brassard, G. - Bratley, P. (1997) *Fundamentos de la Algoritmia.*
- Kumar, V. (2019) *On the prerequisite of Coprimes in Double Hashing.*
- Tenebaum, A. - Augenstein, M. - Langsam, Y. (1993) *Estructuras de Datos con C.*

3.2 Enlaces de interés

- https://www.geeksforgeeks.org/expression-tree/
- https://www.geeksforgeeks.org/evaluation-of-expression-tree/
- https://www.shorturl.at/asuyI