Tutoría 4 - Estructuras de datos y algoritmos

Cristóbal Ganter y Felipe Vera

24 de septiembre de 2012

¡Aviso! ¡Recuerde dejar la máquina virtual en orden una vez que termine su trabajo! ¡Pregunte al ayudante apenas tenga una duda!

Si tiene problemas de *segmentation fault* o *stack overflow* en su programa, una buena opción es *debuggear* su programa con el comando `ddd nombre-de-programa`. Estos errores son difíciles de encontrar, por lo que llenar su programa de `printf`s no lo ayudará mucho.

Contenido de esta semana

- punteros, punteros dobles y operador &
- Asignación dinámica de memoria: stack, heap, `malloc` y `free`.
- Estructuras de datos dinámicas: árboles binarios y operaciones con ellos: búsqueda e inserción.
- Empleo de funciones recursivas para los contenidos anteriores.

Actividad

1. **Uso de los árboles binarios**

   Se usarán árboles binarios para modelar un *parser aritmético*, es decir, una función que acepta un *string* con una operación aritmética y regresa el resultado de tal operación.

   a) Cree la estructura básica de un árbol binario. Defina un `struct _parser` con los miembros `char operacion`, `double operando`, `struct _parser *leftchild` y `struct _parser *rightchild`. Ocupe posteriormente `typedef` para referirse a esta estructura como un tipo `parser`.

   b) Escriba una función para crear un nodo, llamada `parser crear_nodo`, que acepte como argumentos nuevos valores para `operacion` y `operando`.

2. **Crear el árbol a partir de una expresión sencilla: inserción**

   La función que usted creará ahora se limitará a operaciones de suma y resta, sin paréntesis y sin números negativos.

   a) Cree una función `parser *crear_arbol`, que acepte como argumento la cadena de caracteres. Ayúdese de la referencia de la librería de C para dividir la cadena en operadores y números empleando `strtok`; y convertir los números a tipo `double` empleando `atof`.

   b) Tal función necesitará inserción simple en un árbol binario. Cree una función `void insertar_numero (parser **top, double numero)` que inserte un número como nodo hijo izquierdo. De existir el nodo hijo izquierdo, intenta al derecho.

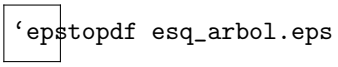
   c) También se necesitará una inserción reemplazando padre por hijo izquierdo. Cree una función `void insertar_operacion (parser **top, double numero, char op)` que cambie al nodo apuntado por `*top` (un número) por la operación indicada, y lo baje al nodo hijo izquierdo. Como nodo hijo derecho, coloca al nuevo número ofrecido como argumento a esta función.

   d) Implemente con estas dos nuevas funciones, haciendo crecer el árbol por la derecha. Pruebe la función de la manera que usted quiera: puede ser ingresando datos por consola o en el mismo código.

3. **Función para evaluar el árbol**

   El parser funcionará aceptando en el miembro `operacion` a los operandos `+`, `-`, `*`, `/`, `^`, paréntesis y `n`. El último indicará que el miembro es una constante numérica, y en ese caso será cuando `operando` entre en acción.

   a) Cree una función recursiva `double evaluar` que acepte como argumento el árbol *parser* y devuelva el resultado de la operación. Guíese por sus apuntes y siga un esquema como el siguiente. El siguiente árbol muestra, por ejemplo, el parser que devolvería un árbol que fue creado con la operación $10 * (0 - 12) + 60 + 35$



4. **Prioridad de operaciones**

   Las potencias se hacen primero, luego las multiplicaciones y divisiones y por último las sumas y restas. Por ahora no ocuparemos los paréntesis.

   a) Asigne un número de prioridad a cada operación: sumas y restas corresponden a 1, multiplicaciones y divisiones corresponden a 2, y potencias a 3. Con ello, modifique las funciones `insertar_operacion` y `crear_arbol` para que se adecúen a la prioridad de las operaciones. Se sugiere que `insertar_operacion` sea recursiva.

5. **Inclusión de paréntesis**

   Los paréntesis sirven para cambiar la prioridad de las operaciones.

   a) Por cada signo de “abrir paréntesis”, se suma 3 a la prioridad de la operación. Por ejemplo, una suma dentro de un paréntesis tendrá prioridad 4, la multiplicación 5; y si están dentro de un paréntesis anidado, prioridad 7 y 8 respectivamente. Haga los cambios respectivos a las funciones necesarias para que respeten los paréntesis.

   b) Por último, pruebe el código ingresando una expresión por consola como argumento.