Tutoría 3 - Estructuras de datos y algoritmos

Cristóbal Ganter y Felipe Vera

10 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: listas doblemente enlazadas, inserción en la cola, inserción en la posición especificada, búsqueda.
- Empleo de funciones recursivas para los contenidos anteriores.

Actividad

1. Listas doblemente enlazadas.
   a) Se desea modelar la fila frente a la caja de un banco. Cree una estructura llamada `persona` que represente a una persona en la fila. La estructura debe permitir construir una lista doblemente enlazada, con los atributos `struct persona *prev, *next`, su `edad` y la **cantidad de** `dinero` **que traen**.
   b) En su función `main` cree un puntero a la primera persona de la fila, esto es la persona que está más cerca de la caja. ¿Hacia donde apuntará este puntero si aún no hemos creado personas?
   c) Escriba una función que le permita crear personas. Las personas deben ser creadas con edad aleatoria entre 18 y 200 años y cantidad de dinero aleatoria entre 0 y 2^{32} unidades monetarias. ¿De qué tipo tiene que ser la variable usada para almacenar la cantidad de dinero?
2. Inserción al final de la lista
   a) Escriba una función `void ins_persona (persona **final, persona *nuevo_elemento)` que le permita insertar personas al final de la fila.
   b) En su función `main` cree un un bucle `while` que se ejecute una vez cada 10 segundos. Si tiene problemas utilizando la función `sleep` avise inmediatamente a su ayudante!
   c) Al comenzar el bucle `while` debe eliminarse a la primera persona de la fila. Esto simulará al usuario que acaba de ser atendido y sale de la fila. La memoria que ocupa este usuario debe ser liberada.
   d) Después de quitar al primer usuario deben agregarse aleatoriamente entre 0 y 3 nuevos usuarios al final de la fila.
3. Cantidad de elementos en una lista.

a) Para cada iteración del bucle `while` imprima por pantalla todos los usuarios que se encuentran en la fila indicando su edad y cantidad de dinero. Además imprima la cantidad total de personas esperando en la fila.

4. Búsqueda en una lista

   a) Para cada iteración del bucle while busque a todas las personas mayores de 70 años y permita que pasen al comienzo de la fila. No es necesario que las personas que reasigne al principio de la fila sigan un orden determinado. Esta búsqueda debe ser realizada desde la persona que está más cerca de la caja a la que está más lejos.

5. Inserción en una lista

   a) Suponga que su fila tampoco está libre de gente que quiera "colarse". Para simular este comportamiento, cree una función `void insertar(persona ** fila, persona * interpuesta, int posicion)`. Tenga precaución con los casos en que fila o interpuesta sean nulos, y la posición (contada desde el principio de la fila) exceda los límites de esta fila.

   b) Incluya esta función en el bucle `while` que hizo en la parte 2. La probabilidad de que una persona se "cuele" en la fila debe ser del 25 %, aleatoriamente en cualquier posición de la fila. Sea cuidadoso de no generar *segmentation faults*.

6. Función recursiva

   a) Una de las personas, al ser atendida, tiene problemas de dinero y el resto de las personas en la fila son inusualmente bondadosas. Cree una función recursiva con prototipo `int bondad(persona *siguiente)`. Esta función recibe como argumento una persona y retorna dinero. Al ser llamada se ejecutan tres acciones:

   - La persona que ha sido pasada como argumento ejecuta la función bondad sobre la persona que está inmediatamente detrás de ella.
   - Una vez que retorne la función `bondad` aplicada a la persona de atrás, el retorno de la función debe ser sumado al dinero de la persona que se ha pasado por argumento.
   - La persona que se ha pasado como argumento le entrega un 10 % de su dinero a quien llamó la función. Esto es, el 10 % del dinero de la persona es usado como retorno de la función.

   b) La función `bondad` debe ser llamada por primera vez con la segunda persona (desde la caja) como argumento y el dinero de salida debe ser asignado a la primera persona.