INF-253 Lenguajes de Programación
Tarea 2: C

Profesor: Roberto Díaz
Ayudante de Cátedra: Anastasiia Fedorova
Ayudantes de Tareas: Sebastián Campos, Héctor Larrañaga, Axel Reyes

2 de octubre de 2020

1. void* Lore

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris ac justo vel leo rutrum vulputate. Cras in sapien accumsan, malesuada ligula sit amet, aliquam augue. Ut pharetra felis imperdiet, tempor mauris sit amet, tristique quam. In hac habitasse platea dictumst. Pellentesque et sapien nunc. Sed lorem enim, convallis vitae vehicula quis, tristique a tortor. Vivamus est nunc, condimentum non erat sit amet, laoreet convallis leo. Vivamus vitae auctor mauris, faucibus accumsan metus.

Integer sed quam et massa iaculis pretium quis ut nibh. Nam metus quam, laoreet et risus nec, porttitor ullamcorper tortor. Duis id tincidunt neque. Curabitur dui leo, semper ac convallis quis, scelerisque iaculis lectus. Nullam a lectus mi. Morbi ultricies ex quis sem bibendum dictum. Quisque sit amet tincidunt mauris, quis accumsan libero. Pellentesque condimentum convallis pharetra. Mauris vitae lorem mauris. Mauris tincidunt velit porttitor arcu bibendum rutrum. Nunc vitae iaculis ipsum, at interdum augue. Quisque erat magna, hendrerit non nibh vel, auctor tincidunt odio.

Aliquam eu laoreet lectus, a convallis elit. Proin nibh ipsum, pellentesque eu sagittis in, venenatis quis erat. Donec ac felis massa. Sed non mollis nisl. Proin tempor elit elit, in efficitur neque scelerisque non. Pellentesque dictum, odio quis porta ultrices, enim orci maximus turpis, quis aliquet lorem neque in justo. Mauris ultricies mauris tortor, in malesuada lorem suscipit nec. Curabitur condimentum gravida velit et congue.

Nunc ullamcorper, lacus eu placerat tristique, orci mauris scelerisque diam, ac volutpat lorem erat sit amet eros. Vestibulum laoreet aliquet posuere. Integer auctor quam sed est imperdiet porttitor. Ut tincidunt imperdiet magna eget feugiat. Suspendisse ut vehicula leo. Proin malesuada tellus sed leo imperdiet elementum. Fusce non turpis at arcu condimentum dictum at eu tellus. Nam fringilla, sapien quis convallis sodales, felis turpis imperdiet lacus, ut ultricies arcu purus ac massa. Praesent aliquam consectetur est, sed vulputate magna molestie vulputate. Quisque velit nunc, auctor in nisi a, scelerisque posuere sapien. Nullam dignissim, augue sit amet luctus venenatis, lacus nisl tristique elit, sit amet varius nibh mauris sed orci. Fusce non nisi arcu.

2. Heap Genérico

Para esta ocasión, se le pedirá realizar una implementación de un Heap genérico. En este heap los nodos van a ser punteros a void, por lo que el heap básicamente podrá albergar cualquier tipo

de dato que se le entregue. Por otro lado, el heap tendrá asociada una función de comparación y una función para mostrar por pantalla elementos, las que se deberán entregar a la creación del heap en forma de punteros a función. El heap deberá tener un mínimo de funciones que deben ser parte de su TDA, además, se les pedirá implementar las soluciones a dos problemas que se propondrán más adelante en la tarea donde deberán utilizar el heap.

3. TDA del Heap

Para el Heap deberán implementar las siguientes funciones:

```
# Constructor del heap
tHeap* newHeap(int (*comparador)(void*, void*),
               void (*printElem)(void*)
               );

# Destructor del Heap
void deleteHeap(tHeap* h);

# Vaciar el heap
void clearHeap(tHeap* h);

# Ver la raz del heap
void* topHeap(tHeap* h);

# Ver el tamao del heap
int sizeHeap(tHeap* h);

# Sacar el primer elemento del heap
void popHeap(tHeap* h);

# Agregar un elemento al heap
void pushHeap(tHeap* h, void* elem);

# Revisar si hubo algn error en la asignacin de memoria del Heap
int is_goodHeap(tHeap* h);

# Revisar si el heap est  vac o
int emptyHeap(tHeap* h);

# Mostrar por pantalla los elementos del heap
void printHeap(tHeap* h);
```

Una implementación completa del Heap para el caso particular de los enteros puede ser encontrada en: https://github.com/HectorxH/c-heap. La función de comparación `comp` debe retornar 1 si el primer elemento pasado como parámetro va antes que el segundo, y 0 en cualquier otro caso. La función `printElem` recibirá un elemento e imprimirá su valor sin estar rodeado de carácteres en blanco (espacios, tabulaciones, saltos de líneas, etc).

4. Tipos de Heap

Como mencionamos con anterioridad, el heap debería poder albergar cualquier tipo de dato. Se les pedirá implementar una función comparación y una función para imprimir los siguientes tipos de datos:

- **String (`char*`)**: la función de comparación debe ordenar en orden alfabético strings compuestos por caracteres alfabéticos. La raíz del heap debe contener la palabra que empiece con la letra que esté antes en el abecedario (sólo se considera la inicial de la palabra). Palabras con las mismas iniciales tienen el mismo peso (no existe diferencia entre mayúsculas y minúsculas).
- **Entero (`int`)**: la función de comparación debe ordenar de menor a mayor. La raíz del heap debe contener el número más pequeño.
- **Flotante (`float`)**: la función de comparación debe ordenar de mayor a menor. La raíz del heap debe contener el número más grande.
- **Arreglo (`int*`)**: La función de comparación debe ordenar arreglos de enteros en base al valor de la sumatoria de sus elementos, ordenando de menor a mayor. La raíz del heap debe contener el arreglo cuya suma sea menor.

Las funciones deben ser llamadas `cmpTipo` y `printTipo`. Por tanto se deben crear 8 funciones:

- **Comparación**: `cmpString`, `cmpEntero`, `cmpFlotante` y `cmpArreglo`
- **Impresión**: `printString`, `printEntero`, `printFlotante` y `printArreglo`

5. Problemas a resolver

1. **Heap Sort**: deberán crear una función que reciba un arreglo de punteros a **void**, un entero con el tamaño del arreglo, un puntero a la función de comparación y un puntero a la función de impresión. La función debe ser capaz de crear un heap con los elementos del arreglo e imprimirlos por pantalla de forma ordenada. La firma de la función debe ser la siguiente:

```
void heapSort(void** arr,
              int N,
              int (*comparador)(void*, void*),
              void (*printElem)(void*)
              );
```

2. **Máximo de bases**: se le entregará un arreglo de structs, con números expresados en distintas bases númericas (base 2, 4, 8 y 10). Cada número irá acompañado de un caracter indicando la base correspondiente. Además se le entregará el largo del arreglo. Deberá confeccionar un heap de structs con los datos entregados (utilizando la función de comparación que deberá crear) e imprimir por pantalla los N máximos del heap (escritos en su base respectiva). La firma de la función y la estructura a utilizar debe ser la siguiente:

```
typedef struct{
    int base;
    char* numero;
}tBase;

void maxBases(tBase* arr,int N, int CantidadDeMaximos);\\
```

Nota: N corresponde al largo del arreglo y CantidadDeMaximos a la cantidad de máximos que se deben obtener del Heap.

Ejemplos:
Si quisiéramos representar el número 2 en binario, el atributo base de tBase debería ser 2, mientras que el arreglo llamado número tendría el valor de "10".
Si quisiéramos representar el número 12 en base octal, el atributo base de tBase debería ser 8, mientras que el arreglo llamado número tendría el valor de "14".
Si quisiéramos representar el número 7 en base cuatro, el atributo base de tBase debería ser 4, mientras que el arreglo llamado número tendría el valor de "13".

6. Funcionamiento

Se creará una función `main` con la cual se probará la implementación de su heap, por lo que se hace énfasis en que mantengan las firmas explícitamente entregadas en el enunciado. **Por cada firma mal escrita habrá descuentos**.

7. Archivos a Entregar

Resumiendo lo anterior, los archivos mínimos a entregar son:

- `README.txt`
- `heap.h`
- `heap.c`
- `main.c`
- `MAKEFILE`

El archivo `heap.h` debe tener las firmas del TDA heap y el **struct** relacionado. El archivo `heap.c` debe tener implementado todas las funciones definidas en el archivo **heap.h**. Finalmente, el archivo `main.c` debe implementar las funciones que solucionan los problemas planteados.

8. Sobre Entrega

- El código debe venir **identado y ordenado**.
- Las funciones no definidas en el enunciado deberán ser comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y su retorno según corresponda.
- Debe estar presente el archivo `MAKEFILE` para que se efectúe la revisión, este debe compilar **TODOS** los archivos. En caso de no entregar Makefile o que este no funcione **la tarea no se revisará**.
- Se debe trabajar de forma individual.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea2LP_RolIntegrante.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la compilación y utilización de su programa.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **18 de Octubre a las 23:55 hora aula**.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Si el programa no compila, **la tarea no se revisará**.
- Se utilizará Valgrind para detectar fugas de memoria.

9. Calificación

- **Puntaje base:**
  - Implementación TDA Heap (30 puntos)
  - Función de comparación e impresión por tipo (5 c/u)
  - Heap Sort (25 puntos)
  - Máximo de Bases (25 puntos)
- **Descuentos:**
  - Código no ordenado (-10 puntos)
  - Código no compila (-100 puntos)
  - `MAKEFILE` no incluído (-100 puntos)
  - Warning (c/u -5 puntos)
  - Falta de comentarios (Máx. -30 puntos)
  - Por cada día de atraso se descontarán 20 puntos (La primera hora de atraso serán solo 10 puntos).
  - Porcentaje de fuga de memoria ((1-5) % -5 puntos (6- 50 %) -15 puntos (51-100 %) -25 puntos)
  - Firma distinta (-15 puntos c/u).

En caso de existir nota negativa esta será reemplazada por un 0.