Tarea 2
Listas y Árboles
Curso 2023

Esta tarea tiene como principales objetivos:

- Trabajar sobre el manejo dinámico de memoria.
- Trabajar con listas simplemente y doblemente enlazadas.
- Trabajar con árboles binarios de búsqueda.

La fecha límite de entrega es el **viernes 21 de abril a las 16:00 horas**. El mecanismo específico de entrega se explica en la Sección 6. Por otro lado, para plantear **dudas específicas de cada paso** de la tarea, se deja un link a un **foro de dudas** al final de cada parte.
A continuación se presenta una **guía** que deberá **seguir paso a paso** para resolver la tarea. Tenga en cuenta que la especificación de cada función se encuentra en el **.h** respectivo, y para cada función se especifica cuál debe ser el orden del tiempo de ejecución en el **peor caso**.

1. Partiendo de la Tarea 1

1. Para comenzar con la Tarea 2 se deben **descargar** los materiales de la sección correspondiente del eva. Observe que en los archivos *fecha.cpp* y *evento.cpp* hay una sección demarcada con el título **PEGAR CÓDIGO TAREA 1**. **Reemplace** estas funciones con el código de la Tarea 1, **sin pisar** el de la función nueva **copiar**. **Compile** el código de la tarea ejecutando *make*, y verifique que los módulos *evento* y *fecha* funcionan correctamente **ejecutando** el test fechaEvento1-combinado. Foro de dudas.

2. **Implemente** las funciones copiarTFecha y copiarTEvento dentro de los módulos *fecha* y *evento* respectivamente. Verifique el funcionamiento de estas funciones **ejecutando** el test fechaEvento2-copia. Foro de dudas.

2. Lista simple: módulo agendaLS

En esta sección se implementará el módulo *agendaLS.cpp*. A diferencia del módulo anterior *agenda.cpp*, *agendaLS* estará implementado mediante una *lista simplemente enlazada*. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de eventos en la agenda. A la hora de implementar listas hay funciones que se pueden hacer de forma iterativa o recursiva. Recomendamos que las implementaciones sean **iterativas** ya que en el módulo de árboles se practicará implementar recursión.

1. Antes de comenzar con la implementación del módulo de listas, se deben **estudiar** las estructuras lineales dinámicas en memoria, o bien **mirando** el teórico grabado en openfing de años anteriores, o esperando a que esté subido el de este año. Foro de dudas.

2. **Implemente** la representación de la lista simplemente enlazada rep_agendaLS. La representación debe almacenar un evento y un puntero al siguiente nodo, por ejemplo, de la siguiente manera:

```
TEvento evento;
rep_agendaLS * sig;
```

   Observe que en *agendaLS.h* se encuentra declarado el tipo TAgendaLS de la siguiente manera:

```
typedef struct rep_agendaLS* TAgendaLS;
```

   Es decir, *TAgendaLS* es un *alias* de *rep_agendaLS ** (son equivalentes). Por lo tanto, también podemos definir *rep_agendaLS* de la siguiente manera:

```
TEvento evento;
TAgendaLS sig;
```

Debemos tener en cuenta que esto es posible ya que el tipo de TAgendaLS fue definido antes (en el .h) que rep_agenda. Foro de dudas.

3. **Implemente** las funciones crearTAgendaLS, agregarEnAgendaLS, imprimirTAgendaLS y liberarTAgendaLS. Recomendamos que la representación de la lista vacía sea simplemente un *puntero a NULL*. Recuerde que la agenda mantiene los eventos de forma ordenada de **menor a mayor** por fecha. Si dos eventos tienen la misma fecha, debe aparecer primero en la agenda el evento ingresado de forma más reciente. Por otro lado, el formato en el que se debe imprimir la Agenda es simplemente utilizando de forma secuencial la función *ImprimirTEvento*. **Ejecute** el caso de prueba agenda1-crear-agregar-imprimir-liberar. Foro de dudas.

4. **Implemente** las funciones esVaciaAgendaLS y copiarAgendaLS. Recuerde que la copia de la agenda debe copiar cada uno de los eventos almacenados. **Ejecute** el caso de prueba agenda2-crear-agregar-esVacia-copiar-imprimir-liberar. Foro de dudas.

5. **Implemente** la función estaEnAgendaLS y **ejecute** el caso de prueba agenda3-esta. Foro de dudas.

6. **Implemente** las funciones obtenerDeAgendaLS y posponerEnAgendaLS. El evento pospuesto debe reubicarse en la agenda según el criterio de orden establecido. En caso de que la nueva fecha del evento pospuesto coincida con la de otro evento en la agenda, el evento con la fecha modificada debe quedar inmediatamente antes en la agenda que todos los de fecha igual o posterior. **Ejecute** los casos de prueba agenda4-obtener-posponer y agenda5-obtener-posponer. Foro de dudas.

7. **Implemente** las funciones imprimirEventosFechaLS y hayEventosFechaLS. **Ejecute** el caso de prueba agenda6-hayEventosFecha-imprimirEventosFecha. Foro de dudas.

8. **Implemente** la función removerDeAgenda y **ejecute** el caso de prueba agenda7-remover. Foro de dudas.

9. **Ejecute** el caso de prueba agenda8-combinado. Foro de dudas.

3. Módulo persona

En esta sección se implementará el módulo *persona.cpp*. Cada elemento del tipo persona almacenará un *id*, una *edad*, un *nombre* y una *agenda* representada mediante una lista de eventos.

1. **Implemente** la representación de persona *rep_persona*. **Implemente** las funciones crearTPersona, liberarTPersona e imprimirTPersona. Tenga en cuenta que el formato de impresión se especifica en *persona.h*. Ejecute el caso de prueba persona1-crear-imprimir-liberar para verificar el funcionamiento de las operaciones. Foro de dudas.

2. **Implemente** las funciones idTPersona, edadTPersona, nombreTPersona, agendaTPersona. **Ejecute** el test persona2-id-edad-nombre-agenda para verificar las funciones. Foro de dudas.

3. **Implemente** las funciones agregarEvento, posponerEvento y removerEvento. **Ejecute** el test persona2-id-edad-nombre-agenda para verificar las funciones. Foro de dudas.

4. **Implemente** la funciones estaEnAgenda, obtenerDeAgenda y copiarPersona. Tenga en cuenta que se debe realizar una **copia** de la agenda. **Ejecute** el test persona4-esta-obtener-copia para verificar el funcionamiento de las funciones. Foro de dudas.

4. Lista doblemente enlazada: módulo personasLDE

En esta sección se implementará el módulo *personasLDE.cpp*. La estructura de tipo TPersonasLDE almacenará elementos del tipo TPersona y estará implementada como una **lista doblemente encadenada (LDE)**. Además, se contará con acceso directo (puntero) al inicio y al final de la lista. Esta estructura lineal es dinámica y permitirá almacenar una cantidad no acotada de personas. La lista **NO estará ordenada** por ningún criterio en particular.

1. **Implemente** la estructura *rep_personasLDE* que permita almacenar una lista doblemente enlazada. Para poder cumplir con los órdenes de tiempo de ejecución de las operaciones recomendamos que la representación sea mediante un **cabezal** con un puntero al nodo *inicial* y otro al nodo *final*. En este sentido, se debe definir además una **representación auxiliar** para los nodos de la lista doblemente enlazada, que tengan un elemento *TPersona*, un puntero a un nodo *siguiente* y un puntero a un nodo *anterior*. Foro de dudas.

   Implemente las funciones crearTPersonasLDE, insertarTPersonasLDE y liberarTPersonasLDE. Puede resultar útil implementar una **función auxiliar** que permita liberar un nodo individual. Verifique el funcionamiento de las funciones ejecutando el test personasLDE1-crear-insertar-liberar. Foro de dudas.

2. **Implemente** las funciones imprimirTPersonasLDE y cantidadTPersonasLDE. **Ejecute** el test personasLDE2-crear-liberar-imprimir-cantidad para verificar el funcionamiento de las funciones. Foro de dudas.

3. **Implemente** las funciones eliminarInicioTPersonasLDE y eliminarFinalTPersonasLDE. **Ejecute** el test personasLDE3-eliminarInicio-eliminarFinal para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones estaEnTPersonasLDE y obtenerDeTPersonasLDE. **Ejecute** el test personasLDE4-esta-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Implemente** la funcion concatenarTPersonasLDE. Preste especial atención a que esta función debe ser $\Theta(1)$ en el peor caso. Ejecute los tests personasLDE5-concatenar y personasLDE6-concatenar-tiempo para verificar el funcionamiento de la función. En test de tiempo corrobora que las funciones del módulo hayan sido implementadas en orden del tiempo de ejecución requerido. De no ser así, el test no terminará de ejecutarse rápidamente y dara *timeout* (ERROR: 124). Foro de dudas.

6. **Ejecute** el test personasLDE7-combinado. Foro de dudas.

5. Árbol binario de búsqueda: módulo personasABB

En esta sección se implementará el módulo *personasABB.cpp*. La estructura de tipo TPersonasABB almacenará elementos del tipo TPersona y estará implementada como un **árbol binario de búsqueda (ABB)**, **ordenado por el índice de la persona**. La estructura **no aceptará índices repetidos** (se puede asumir que nunca se agregarán repetidos).

1. Antes de comenzar con la implementación del módulo, se deben **estudiar** los árboles binarios de búsqueda, o bien **mirándo** el teórico grabado en openfing de años anteriores, o esperando a que esté subido el de este año. Foro de dudas.

2. **Implemente** la representación del árbol binario de búsqueda *rep_personasAbb*. La representación debe tener un elemento del tipo *TPersona* y un puntero a un nodo *izquierdo* y a otro nodo *derecho*. Foro de dudas.

3. **Implemente** las funciones crearTPersonasABB, esVacioTPersonasABB, insertarTPersonasABB y liberarTPersonasABB. Recomendamos que el árbol vacío se represente mediante un *puntero a NULL*. Por otro lado, recomendamos crear una **función auxiliar** para liberar un nodo individual. **Ejecute** el test personasABB1-crear-esVacio-insertar-liberar para verificar el funcionamiento de las funciones. Foro de dudas.

4. **Implemente** las funciones imprimirTPersonasABB y cantidadTPersonasABB. **Ejecute** el test personasABB2-crear-liberar-insertar-imprimir-cantidad para verificar el funcionamiento de las funciones. Foro de dudas.

5. **Implemente** las funciones maxIdPersona y removerTPersonasABB. **Ejecute** el test personasABB3-remover para verificar el funcionamiento de las funciones. Foro de dudas.

6. **Implemente** las funciones estaTPersonasABB y obtenerDeTPersonasABB. **Ejecute** el test personasABB4-esta-obtener para verificar el funcionamiento de las funciones. Foro de dudas.

7. **Implemente** las funciones alturaTPersonasABB y esPerfectoTPersonasABB. Un árbol es *perfecto* cuando cada nodo interno tiene dos hijos y todos los nodos hoja están en el mismo nivel. Tenga especial cuidado en que para cumplir con el orden de tiempo de ejecución requerido $\Theta(n)$ (para calcular la altura y si el árbol es perfecto) se debe **recorrer cada nodo una única vez**. Es decir, por ejemplo, **no se puede** llamar a la función **cantidadPersonasABB** desde cada nodo ya que de esa forma se recorren los sub-árboles **múltiples veces**. **Ejecute** el test personasABB5-altura-esPerfecto para verificar el funcionamiento de las funciones. Foro de dudas.

8. **Implemente** la función mayoresTPersonasABB, que retorna un **nuevo** árbol (**no comparte memoria** con el árbol parámetro) con las personas que son mayores a cierta edad. Tenga especial cuidado en que para cumplir con el orden de tiempo de ejecución requerido $\Theta(n)$ se debe **recorrer cada nodo una única vez**. Es decir, por ejemplo, **no se puede** crear un nuevo árbol y llamar a la función **insertarTPersonasABB** por cada nodo, porque la función es $\Theta(n)$, y si se la llama desde cada desde cada nodo el orden del tiempo de ejecución en el peor caso es $\Theta(n^2)$. **Ejecute** el test personasABB6-maxId-mayoresEdad para verificar el funcionamiento de las funciones. Foro de dudas.

9. **Implemente** la función aTPersonasLDE, que dado un árbol de personas retorna una lista de tipo *TPersonasLDE* con las mismas personas del árbol *ordenadas* por id. La lista retornada **no compoarte memoria** con el árbol parámetro. Tenga especial cuidado en que para cumplir con el orden de tiempo de ejecución $\Theta(n)$, **no se pueden recorrer las listas**. **Ejecute** el test personasABB7-aPersonasLDE para verificar el funcionamiento de las funciones. Foro de dudas.

10. **Ejecute** el test personasABB8-combinado. Foro de dudas.

6. Test final y entrega de la Tarea

Para finalizar con la prueba del programa utilice la regla *testing* del Makefile y verifique que no hay errores en los tests públicos. Esta regla se debe utilizar **únicamente luego de realizados todos los pasos anteriores (instructivo especial para PCUNIX en paso 3)**.

1. **Ejecute**:

```
$ make testing
```

   Si la salida no tiene errores, al final se imprime lo siguiente:

```
-- RESULTADO DE CADA CASO --
11111111111111111111111111111
```

   Donde un 1 simboliza que no hay error y un 0 simboliza un error en un caso de prueba, en este orden:

```
fechaEvento1-combinado
fechaEvento2-copia
agenda1-crear-agregar-imprimir-liberar
agenda2-crear-agregar-esVacia-copiar-imprimir-liberar
agenda3-esta
agenda4-obtener-posponer
agenda5-obtener-posponer
agenda6-hayEventosFecha-imprimirEventosFecha
agenda7-remover
agenda8-combinado
persona1-crear-imprimir-liberar
persona2-id-edad-nombre-agenda
persona3-agregar-posponer-remover
persona4-esta-obtener-copia
personasLDE1-crear-insertar-liberar
```

```
personasLDE2-crear-liberar-imprimir-cantidad
personasLDE3-eliminarInicio-eliminarFinal
personasLDE4-esta-obtener
personasLDE5-concatenar
personasLDE6-concatenar-tiempo
personasABB1-crear-esVacio-insertar-liberar
personasABB2-crear-liberar-insertar-imprimir-cantidad
personasABB3-remover
personasABB4-esta-obtener
personasABB5-altura-esPerfecto
personasABB6-maxId-mayoresEdad
personasABB7-aPersonasLDE
personasABB8-combinado
```

Foro de dudas.

2. **Prueba de nuevos tests**. Si se siguieron todos los pasos anteriores el programa creado debería ser capaz de ejecutar todos los casos de uso presentados en los tests públicos. Para asegurar que el programa es capaz de ejecutar correctamente ante nuevos casos de uso es importante realizar tests propios, además de los públicos. Para esto **cree un nuevo archivo en la carpeta test**, con el nombre *test_propio.in*, y **escriba una serie de comandos** que permitan probar casos de uso que no fueron contemplados en los casos públicos. **Ejecute el test** mediante el comando:

   ```
   $ ./principal < test/test_propio.in
   ```

   y verifique que la salida en la terminal es consistente con los comandos ingresados. La creación y utilización de casos de prueba propios, es una forma de robustecer el programa para la prueba de los casos de test privados. Foro de dudas.

3. **Prueba en pcunix**. Es importante probar su resolución de la tarea con los materiales más recientes y en una pcunix, que es el ambiente en el que se realizarán las correcciones. Para esto siga el procedimiento explicado en Sugerencias al entregar.

   IMPORTANTE: Debido a un problema en los *pcunix*, al correrlo en esas máquinas se debe iniciar valgrind **ANTES** de correr *make testing* como se indica a continuación:

   Ejecutar los comandos:

   ```
   $ make
   $ valgrind ./principal
   ```

   Aquí se debe **ESPERAR** hasta que aparezca:

   ```
   $ valgrind ./principal
   ==102508== Memcheck, a memory error detector
   ==102508== Copyright (C) 2002-2022, and GNU GPL'd, by Julian $ Seward et al.
   ==102508== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
   ==102508== Command: ./principal
   ==102508==
   $ 1>
   ```

   Luego se debe ingresar el comando **Fin** y recién luego ejecutar:

   ```
   $ make testing
   ```

   Foro de dudas.

4. **Armado del entregable**. El archivo entregable final debe generarse mediante el comando:

   ```
   $ make entrega
   ```

   Con esto se empaquetan los módulos implementados y se los comprime generando el archivo `EntregaTarea2.tar.gz`.

   El archivo a entregar **DEBE** ser generado mediante este procedimiento. Si se lo genera mediante alguna otra herramienta (por ejemplo, usando un entorno gráfico) **la tarea no será corregida**, independientemente de la calidad del contenido. Tampoco será corregida si el nombre del archivo se modifica en el proceso de entrega. Foro de dudas.

5. **Subir la entrega al receptor**. Se debe entregar el archivo **EntregaTarea2.tar.gz**, que contiene los módulos a implementar **fecha.cpp**, **evento.cpp**,**agendaLS.cpp**,**personasLDE.cpp** y **personasABB.cpp**. Una vez generado el entregable según el paso anterior, es necesario subirlo al receptor ubicado en la sección Laboratorio del EVA del curso. **Recordar que no se debe modificar el nombre del archivo generado mediante** `make entrega`. Para verificar que el archivo entregado es el correcto se debe acceder al receptor de entregas y hacer click sobre lo que se entregó para que automáticamente se descargue la entrega.
   IMPORTANTE: Se puede entregar **todas las veces que quieran** hasta la fecha final de entrega. La última entrega **reemplaza a la anterior** y es la que será tenida en cuenta. Foro de dudas.