ESTRUCTURAS DE DATOS PL1

Nombre: García Ibáñez Pedro DNI: 54890356M
Nombre: Rodríguez Nieto Rodrigo DNI: 72008756B

ÍNDICE

ESPECIFICACION DE LA INTERFAZ DE LOS TADS IMPLEMENTADOS

En esta primera parte del programa hemos implementado 3 estructuras basadas en tipos abstractos de datos (TADs); la pila, la cola y la lista.

TADS CREADOS

Pila

espec PILA[ELEMENTO]

usa BOOLEANOS

parámetro formal

géneros elemento

fparámetro

géneros pila

fespec

Cola

espec COLA[ELEMENTO]

usa BOOLEANOS

parametro formal

géneros elemento

fparametro

generos cola

fespec

Lista

espec LISTA[ELEMENTO]

usa BOOLEANOS

parámetro formal

géneros elemento

géneros lista

fespec

DEFINICION DE LAS OPERACIONES

Pila

Operaciones:

pvacia: → pila	{Generadora}
insertar: elemento pila → pila	{Generadora}
parcial extraer: pila → pila	{Modificadora}
parcial cima: pila → elemento	{Observadora}
longitud: pila → natural	{Observadora}
vacía?: pila → bool	{Observadora}

Ecuaciones:

var p: pila; x: elemento

ecuaciones de definitud

Def (desapilar (apilar (x, p)))

Def (cima (apilar (x, p)))

ecuaciones

desapilar (apilar (x, p)) = p

cima (apilar (x, p)) = x

vacía? (pvacía) = T

vacía? (apilar (x, p)) = F

longitud (pvacía) = 0

longitud (apilar (x, p)) = longitud (p) +1

Cola

Operaciones:

cvacía: → cola	{Generadora}
encolar: elemento cola → cola	{Generadora}
parcial desencolar: cola → cola	{Modificadora}
parcial primero: cola → elemento	{Observadora}
vacía: cola → bool	{Observadora}
longitud: cola → natural	{Observadora}

Ecuaciones:

var c: cola; x: elemento

ecuaciones de definitud

Def (eliminar (añadir (x, c)))

Def (primero (añadir (x, c)))

ecuaciones

eliminar (añadir (x, c)) = c

vacía(c) = F ⇒ eliminar (añadir (x, c)) = añadir (x, eliminar(c))

primero (añadir (x, cvacía)) = x

vacía (c) = F ⇒ primero(añadir (x, c)) = primero (c)

vacía (cvacía) = T

vacía (añadir (x, c)) = F

longitud (cvacia) = 0

longitud (encolar (x, c)) = longitud (c) +1

Lista

Operaciones

[]: →lista (Generar lista vacia)	{Generadora}
_:_: elemento lista → lista	{Generadora}
parcial elimPrimero: natural lista → lista	{Modificadora}
parcial primero: lista → elemento	{Observadora}
parcial ultimo: lista → elemento	{Observadora}
longitud: lista → natural	{Observadora}
parcial contiene: natural lista →bool	{Observadora}
vacia: lista → bool	{Observadora}

var x: elemento; l: lista; n natural

ecuaciones de definitud

Def (primero (x:l))

Def (ultimo (l:x))

Def (elimPrimero (x:l))

Def (contiene (n, l))

ecuaciones

elimPrimero (x, l) = l

vacia (l) = F → elult (x:l) = x:elult (l)

prim (x:l) = x

vacia (l) = T → ultimo (x:l) = primero (x:l)

vacia (l) = F → ultimo (x:l) = ultimo (l)

vacia ([]) = T

vacia (x:l) = F

longitud ([]) = 0

longitud (x:c) = longitud (c) +1

SOLUCIONES ADOPTADAS

Problema de la implementación dinámica

Al inicio al implementar la pila, el atributo "valor", que es lo que guarda la pila, se describió como un objeto, Proceso p, esto causó el problema de que no se había implementado dinámicamente, así que se tuvo que cambiar toda la estructura a puntero a objeto proceso "Proceso* p, lo cual llevo tiempo. Afortunadamente vimos este error pronto. En el desarrollo de la práctica y para el resto de EEDD se hizo bien desde el principio.

Problema Inicial con el uso de GitHub y GitHub Desktop

Tuvimos algunos problemas al usar la herramienta por varios motivos, primero se nos olvidó añadir el .gitignore en la primera interacción, y al añadirlo más tarde se mandaban archivos como el .mk aunque estuvieran añadidos, segundo teníamos repetido varias veces el nombre de PL1 en distintas carpetas, lo que generaba cierta confusión al programa que si se saltaba una carpeta por llevar el mismo nombre no encontraba los archivos. Además, uno de los miembros utilizaba una versión más antigua del codelite por tenerlo del año pasado mientras que el otro tenía el material reciente, por lo que el traspaso de archivos incluía elementos de diferentes versiones que generaba aún más problemas al compilar. Para arreglarlo, el profesor Carlos nos ayudó, y la solución consistió en mandar primero la eliminación de los archivos que no queríamos que se reenviasen, después se revisó la versión y se retocó el programa de compilación para que no diese problemas de versiones, y por último se creó un nuevo workspace con un nombre distinto para reorganizar de nuevo los archivos y que dejase de generar problemas por la estructura de carpetas.

Problema con la sobre definición de la clase Proceso

La clase proceso al ser común a todas las EEDD que se usan producía errores ya que se declaraba en clases en las que ya estaban declaradas por un import a otra clase que ya tenía un

import a esa clase, por lo que con tantas clases con un import a Proceso era difícil ver donde estaba el "doble import proceso".

Duda a la hora de implementar los nombres de procesos

Se dudó sobre cómo hacer el atributo nombre de los procesos, se pensó e intentó hacer con array de char, pero tras intentar implementarlo se hizo con simple string.

Problema con la función reiniciar

La función necesitaba vaciar todas las estructuras que existen en el programa y además de las variables globales que se usan para generar procesos. Para solucionarlo se modificó la función insertar en las listas, con la ayuda de los profesores de la asignatura, así como vaciar el set una vez se reinician las pilas.

MÉTODOS DESTACADOS

Procesos

Los procesos tienen los siguientes métodos:

Métodos que generan sus datos:

generarNombre(): elige un nombre entre un array de 10 nombres.

generarBool(): devuelve true o false, para el tipo.

generarPrioridad(): según el tipo hace un cálculo u otro, y lo añade al set de prioridades, para que no se repitan.

Métodos para mostrar su información:

Mostrar_proceso/_cola/_lista(): son 3 métodos diferentes que varían según el tipo de estructura que les llame para adecuarse al formato que pide en el enunciado.

Otros:

Gets y sets para la prioridad, tipo, estado y nombre.

resetProcesos(): reinicia las variables globales de la clase Proceso.

Pila

Posee las operaciones básicas de las pilas (Insertar, extraer, cima y getLongitud) las cuales han sido modificadas con respecto a los apuntes para que en lugar de contener valores, contenga punteros al objeto proceso.

Además la pila incluye los métodos:

Vaciar(): libera el contenido de la pila. Actúa como un destructor.

Mostrar(): se apoya en el método comentado de Proceso para mostrar los contenidos de cada proceso que se encuentre en la pila.

Colas

Posee las operaciones básicas de las colas (desencolar, verPrimero, getLongitud, esVacia) las cuales han recibido las mismas modificaciones que las pilas, para trabajar con los punteros a Procesos.

encolarOrdenado(): que extrae de las pilas e inserta según prioridad y tipo en su cola correspondiente.

Vaciar(): libera el contenido de la Cola. Actúa como destructor.

Listas

Posee las operaciones básicas de las listas (insertar (al inicio), eliminar, getLongitud, getPrimero, getUltimo, contiene) e incluye métodos para satisfacer las características de la práctica, estos son:

Eliminar(int PID): elimina según un PID introducido previamente.

Mayor prioridad: muestra el proceso en la lista con mayor prioridad.

Menor prioridad: muestra el proceso con menor prioridad de la lista.

busquedaNombre(string nombre): muestra los procesos creados por un nombre introducido previamente.

cambiarPrioridad(int PID, int Prioridad): busca si hay un proceso con el PID introducido y en caso de que exista se le cambia la prioridad seleccionada.

Mostrar(): Muestra los datos de los procesos que se encuentren en la lista.

Vaciar(): elimina el contenido de la lista. Actúa como destructor.

Gestor

Los métodos de la clase Gestor se basan en llamadas a otros métodos de las estructuras de datos.

Los hemos clasificado en métodos de interfaz, o aquellos que se utilizan para mostrar los datos por pantalla que no son opciones, métodos de pila para los que hacen una llamada a la pila, métodos de colas para aquellos que llaman a las colas, métodos de listas para los que llaman a las listas, y el Reiniciar que llama a todas las estructuras para reiniciar el programa sin necesidad de cerrar y abrirlo.

COMPORTAMIENTO DEL PROGRAMA

El programa muestra el menú de opciones. A continuación, se describe lo que ocurre al seleccionar cada una de las opciones.

A → Si la longitud de la pila es menor que 48 genera 12 procesos llamando al constructor de la clase Proceso y los mete en la pila de la clase gestor. Si está fuera de rango lanzara out_of_range.

B → Esta opción llama a un método de la clase Pila, llamado mostar_pila, este recorre la pila con un while, mostrando los datos (con un método de la clase Proceso) y avanzando el puntero hasta que este llegue a NULL (se termine la pila).

C → Esta opción recorre la pila (de la misma manera que la opción b) pero esta vez eliminando los datos, hasta que la pila este vacía.

D → Esta opción recorre la pila hasta que este vacía haciendo la operación pop e insertando en su cola correspondiente (evaluado con un if) en su cola, teniendo en cuenta tanto la prioridad como el "aforo" de la cola, insertando en caso de misma prioridad, para esto se apoya en punteros a primero y último, en caso de no ser primero ni ultimo entra en un bucle while, para recorrer la pila y encontrar su lugar según prioridad.

E → Esta opción llama a una función de la clase cola que se encarga de imprimir los procesos, la función de la clase cola llama a su vez a otra de la clase proceso, que se encarga de imprimir por pantalla en el formato pedido.

F → Esta opción hace lo mismo, pero para las colas de tiempo real.

G → Esta opción llama al procedimiento vaciar de la clase cola, este desencola la cola mientras esta contenga el nodo primero.

H → Esta opción vacía las colas, las dos de cada tipo en su lista correspondiente, por lo tanto se desencola (lo que devuelve el puntero a proceso) y lo enlista en la lista que le corresponde, por lo tanto las colas se enlistan de forma ordenada, pero una detrás de otra, por lo tanto las listas no están ordenadas.

I → Esta opción llama a un procedimiento de la clase lista que imprime las columnas de la tabla, y en la función mostar_procesos_lista se muestran los datos del proceso, en el formato específico pedido para las listas, en este caso para la lista normal.

J → Mismo procedimiento que en la opción anterior pero accediendo a la lista en tiempo real.

K → Esta opción accede a una función de la lista que le corresponda para mostrar en formato cola el proceso con mayor o menor prioridad, según el caso que definamos en código. Se recorre la lista con un while(aux), siendo aux=ultimo y avanzado con aux=aux->siguiente

l → Esta opción pedirá por consola un tipo string y lo guardará en una variable, imprimirá las columnas y llamará al método de cada lista llamado busquedaNombres(), que muestra el proceso en formato lista(con columnas) de los procesos que tengan el mismo atributo nombre que la variable nombre introducida.

M → Esta opción pedirá un PID por consola y lo guardará en una variable de tipo int, después con una estructura try intenta buscarlo en las dos listas, en caso de que no lo encuentre da error, en caso de que lo encuentre, lo guarda en una variable local eliminado, que será insertado en la pila inicial.

N → Esta opción pedirá un PID al igual que la anterior y una prioridad, en caso de que encuentre el proceso le cambiará el atributo prioridad con la operación setPrioridad, que se encuentra en la clase Proceso.

O → Esta opción elimina todos los procesos de cada EEDD, así como posibles contadores o punteros, reiniciando así el programa.

BIBLIOGRAFIA

Apuntes de clase, documentos proporcionados por los profesores y asistencia de los mismos.

Colección Set->https://aprende.olimpiada-informatica.org/cpp-set

https://learn.microsoft.com/es-es/cpp/standard-library/set-class?view=msvc-170