

Práctica 1: Elementos básicos de los lenguajes de programación

Link de portafolio en GitHub: [Asignatura Paradigmas](#)

Link de portafolio en GitHub Pages (página estática): [Asignatura Paradigmas](#)

Introducción

En esta práctica se vio el tema de la estructura de datos cola, con el propósito de identificar los elementos fundamentales de los lenguajes de programación, en este caso con el lenguaje C. Se identificarán los nombres, marcos de activación, bloques de alcance, administración de memoria, expresiones, comandos, control de secuencia como los de selección, iteración y recursión, subprogramas, y los tipos de datos. Con esta estructura de dato se logró implementar para poder implementar un sistema de gestión de trabajos, en donde, se almacenan los trabajos ingresados por los usuarios con sus identificadores y contenido, una vez en la cola se podrá manipular esta información como ver qué trabajo es más urgente, ver el que sigue, poder completar un trabajo, ver su estado, etc. Dado que los trabajos tienen un orden de completación (IFO), el uso de la cola es perfecto para este caso.

Diseño

PrintJob_t o, en este caso, printQueue, es la función que imprime todos los elementos de la cola. El único campo que se encuentra en esta función es la cola, ya que se requiere de ella para recorrer todos sus elementos y mostrarlos en pantalla.

1. Cola estatica Esta se implementó mediante el uso de un struct que será nuestra estructura de la cola, aquí se usaron los atributos frente y final, esto para poder llevar el rastro de los elementos en la fila.
2. Cola dinamica En esta se hizo uso de las listas enlazadas simples para llevar a cabo el uso de la memoria dinámica, apoyándonos en los nodos que apuntan a otros y manteniendo la estructura de frente y final para conocer el primer nodo y el último, así como el rastro de los datos.

Implementacion

Las funciones utilizadas en cada uno de los programas.

Las funciones en común de los dos programas:

- **int msg();** Muestra en pantalla (consola) las opciones, el menú de las funcionalidades disponibles.
- **void menu();** Procesa por medio de un switch la opción elegida por el usuario para ejecutar el código adecuado.
- **void agregarTrabajo(Trabajo *trabajo);** Pide al usuario que ingrese la información del trabajo a agregar, guardándola en una variable de ese tipo.

- **void initQueue(Queue *queue);** Inicializa la cola con valores por defecto.
- **void isEmpty(Queue *queue);** Revisa el contador, si es 0, se dice que está vacía la cola.
- **void isFull(Queue *queue);** Revisa el contador, si es diferente a 0, se dice que no está vacía la cola.
- **void peek(Queue *queue);** Revisa la frente de la cola para conocer el primer trabajo. Esta no debe modificar la cola debido a que solo es para consultarla.

Funciones de cola estatica

- **void inQueue(Queue *queue, Trabajo *trabajo);** Recibe el trabajo a agregar y agrega al final de la cola. Incrementa la cantidad de elementos en la cola.
- **void deQueue(Queue *queue);** Ubicando el trabajo de enfrente, lo remueve de la cola, lo deja de rastrear, ya que realmente sigue ahí. Disminuye el contador.
- **void printQueue(Queue *queue);** Imprime toda la cola desde la frente.

Funciones de cola dinamica

- **void inQueue(Queue *queue, Trabajo trabajo);** Crea un nodo con dirección de memoria, se almacena en el el trabajo y dependiendo del estado de la cola, se inserta en la cola, asignando su puntero siguiente al que le sigue.
- **void deQueue(Queue *queue);** Remueve el primer nodo, el nodo del frente, de la cola, usando la función free(), así tomando de vuelta el espacio de memoria usado para ese nodo.
- **void printQueue(Queue *queue);** Imprime toda la cola, empezando por enfrente. Recorre por todos los nodos usando el puntero siguiente para avanzar.
- **void limpiarQueue(Queue *queue);** Recorre toda la cola y elimina uno por uno los nodos de la memoria con la función free().
- **void simularImpresion(Queue *queue);** Recorre toda la cola y mediante la función SLEEP() de la librería Windows, pudiendo implementar un delay, se simula el procesamiento de los trabajos, específicamente de las páginas.

Para estos dos casos de uso de una librería personalizada para la validación de datos, es decir, para asegurar el input correcto del usuario a la aplicación, así evitando errores inesperados. Para las colas se implementó el diseño de la cola circular, esto para poder evitar pasos costosos. En el caso de la creación de memoria dinámica se valida su creación correcta, esto para el manejo adecuado de la aplicación.

Demostración de conceptos

Alcance y duración

Dentro del programa se tienen 3 variables en el menú:

1. queue: Este es de tipo Queue, una estructura y es la cola del programa, esta variable se pasa por referencia a varias funciones en las que se requieren de la cola, como en insertar, eliminar, consultar,

etc.

2. trabajo: Esta variable es de tipo Trabajo (estructura), en esta se almacena el contenido del trabajo, como el id, usuario, domicilio, cantidad de páginas, etc. Esta se pasa por valor a la función de insertar en la cola.

3. trabajold: El tipo de dato de esta variable es de tipo entero, esta que se encuentra en el menu como variable local se pasa por referencia a la función de crear trabajo, esto para poder mantener el conteo de los id ya creados.

Memoria

En el stack del programa se encuentran las funciones, las variables locales, los punteros pasados como argumentos, etc.

En el heap del programa se encuentran los nodos, o bien, los trabajos creados con información del usuario, estos que fueron creados usando la función malloc().

Subprogramas

Algunas funciones usan como parámetros punteros, esto se debe a que se necesita modificar el valor actual de la variable, para esto se necesita acceder a su ubicación, esto es posible pasando por referencia estos valores, es decir, usando punteros en los parámetros de las funciones. En otras situaciones solo se usa const, esto se debe a que solo se necesita el valor sin modificaciones, por lo que no se necesita acceder a su espacio de memoria.

Tipos de datos

Para este programa fue necesario el uso de structs, esto ya que se ocupa una estructura para mantener guardada tanto la cola como sus datos, es decir, informacion.

Simulación

A la hora de procesar el trabajo, se muestra en consola la impresión de las páginas por trabajo. Esta simulación fue lograda por medio de la librería Windows y su función SLEEP(), a la cual se le especifica esperar 1000ms por trabajo, con esto se mostrará un porcentaje de que tan completo está el trabajo en procesar. Se toma en cuenta la cantidad de copias que se pidió.

Evidencia de simulacion de impresión:  EVIDENCIA

Análisis comparativo

Las estructuras de datos son esenciales para el almacenamiento y gestión de la información cuando estamos trabajando con colas, como en el caso de nuestro programa, se tiene la opción de dos estilos/modos en los que podemos implementar esta estructura para cumplir con el programa. Estas dos formas son con la implementación de la cola estática y con la cola dinámica, ambas ofreciendo sus beneficios y ventajas sobre la otra. En seguida se analizarán estos dos métodos de implementación y llevar a cabo un análisis comparativo para explorar estas mismas ventajas y desventajas.

Estructuras de Datos Estáticas

Las estructuras estáticas, como los arreglos tradicionales, reservan un bloque de memoria desde el momento de la compilación o el inicio de la ejecución.

Gestión de memoria: La gestión de memoria es más sencilla para trabajar al usar memoria estática, debido a que la memoria se libera por sí sola al no estar usando el elemento o al terminar la ejecución del programa. En los arreglos por ejemplo se define el tamaño y si se intenta sobrepasar este tamaño definido habrá un error de desbordamiento.

Acceso y rendimiento: El acceso a elementos en memoria es rápido, debido a que la dirección de memoria se calcula mediante un desplazamiento directo.

Limitaciones: La inserción de elementos en posiciones medias en la cola es más costosa, debido a tener que realizar el desplazamiento de elementos.

Estructuras de Datos Dinámicas

En el programa se hizo uso de los nodos para almacenar la información, de lo cual se utilizó la memoria dinámica.

Gestión de memoria: La memoria debe ser solicitada mediante la función malloc(), esta para crear la memoria necesaria de nuestra información que se quiere almacenar, después se debe liberar esta memoria usando la función free(). Si omitimos la liberación de memoria, tendremos un memory leak. También se puede asegurar si malloc regresa null verificando si se creó exitosamente el nodo.

Inserción de prioridad: Debido a que en este caso estamos utilizando los nodos, de los cuales cada uno tiene un puntero que apunta al siguiente nodo, se puede ir recorriendo de nodo en nodo para buscar la posición requerida para insertar el nuevo nodo, así evitando desplazamientos costosos, solo siendo requerido cambiar los punteros.

Costo extra en memoria: Debido a que con la estructura de memoria dinámica se necesita tener almacenada la dirección de memoria del siguiente nodo, se requiere mayor memoria.

Evidencias

Alcance/duració:



Memoria:



Contratos:



Sesión 1: log/captura mostrando lleno/vacío.



Sesión 2: log/captura mostrando FIFO y salida limpia.



Sesión 3: log/captura/video mostrando progreso y estados.



Conclusion

Para concluir, durante la practica y el desarrollo del programa obtuvimos conocimiento sobre la implementación de la estructura de datos cola de diferentes métodos, esto dando a conocer las diferentes ventajas y desventajas de cada uno. Al implementar las diferentes versiones del programa se pudo observar las diferencias entre el uso del stack y del heap, así como los elementos que se almacenan/usan cada uno de estos dos. También se conoció más a fondo el uso adecuado del almacenamiento de información con la cola estática y la cola dinámica, teniendo en cuenta el uso y liberación de memoria con las funciones malloc() y free(). Esto nos permitió llevar a cabo un análisis comparativo entre estos dos métodos de la programación.

Referencias

Pozo, S. (s. f.). Estructuras de datos: Colas. © 2000 Salvador Pozo. <https://conclase.net/c/edd/cap3>