Universidad Mariano Gálvez de Guatemala

Facultad de Ingeniería en Sistemas

Ingeniería en Sistema de Información y Ciencias de la Computación

Curso: Algoritmos

Catedrático: Ing. Miguel catalán

Ciclo: Segundo

Manual Técnico

Nombre: Sharon Patricia Guacamaya Ley

Carné: 7590-25-24933

Guatemala, octubre del 2,025.

1. Descripción Técnica del Sistema

El sistema Gestor de Notas Académicas, es una aplicación desarrollada en Python que permite al estudiante gestionar las calificaciones de sus cursos. Opera completamente en consola y aplica conceptos fundamentales de programación estructurada: uso de listas, pilas, colas, funciones, validaciones, algoritmos de búsqueda y ordenamiento.

2. Estructura general del código

1.Definición de listas iniciales de cursos y notas.

2. Implementación de estructuras adicionales: pila (historial de cambios) y cola (solicitudes de revisión).

3. Creación de funciones para cada opción del menú (registro, visualización, búsqueda, ordenamiento, etc.).

4. Implementación de un menú interactivo en bucle que gestiona la navegación del usuario.

3. Explicación del uso de listas, pilas, colas, etc.

Listas: Se utilizan para almacenar los nombres de los cursos y sus notas asociadas.
Pila: (historial_cambios): Registra las modificaciones o eliminaciones, aplicando el principio LIFO.
Colas (cola revisiones): Simula las solicitudes de revisión de cursos, aplicando el principio FIFO.

4. Justificación de los algoritmos de ordenamiento implementados

Búsqueda lineal: Permite localizar un curso recorriendo la lista de manera secuencial.
Búsqueda binaria: Requiere que la lista esté ordenada por nombre y permite una búsqueda más eficiente.
Ordenamiento burbuja: Se utiliza tanto para ordenar por nombre como por nota, recorriendo e intercambiando elementos.

5.Documentación breve de cada función o módulo

Registrar_curso: Agrega un curso nuevo validando nombre y nota.

Mostrar_cursos: Muestra todos los cursos registrados.

Promedio: Calcula el promedio general de las notas.

Contar_aprobados_reprobados: Determina cuántos cursos están aprobados o reprobados.

Busqueda_curso_lineal: Permite buscar un curso por nombre.

Actualizar_nota: Actualiza la nota de un curso existente y lo guarda en el historial.

Eliminar_curso: Elimina un curso y su nota correspondiente.

Ordenar_por_nota y ordenar_por_nombre: Ordenan la lista según el criterio elegido.

Buscar_curso_binario: Realiza una búsqueda binaria en la lista.

Ordenada.simular_cola: Simula la gestión de solicitudes de revisión.

Mostrar_historial: Muestra el registro de los últimos cambios aplicados.

6. Pseudocódigo Principal

```
INICIO


 // ESTRUCTURAS INICIALES

 lista cursos ← ["Algoritmos", "Álgebra Lineal", "Precálculo", "Contabilidad",
"Matemática Discreta"]

 lista notas ← [62, 80, 65, 90, 97]

 cola_revisiones ← lista vacía   // para simular una cola FIFO

 historial_cambios ← lista vacía // para registrar cambios (pila LIFO)


 PROCEDIMIENTO mostrar_menu()

  IMPRIMIR opciones del menú (1..13)

 FIN_PROCEDIMIENTO


 // 1. Registrar nuevo curso

 PROCEDIMIENTO registrar_curso()

  pedir nombre

  SI nombre ya está en cursos ENTONCES

   mostrar "Ese curso ya está registrado."

   RETORNAR

  FIN_SI

  pedir nota

  SI nota es número y 0 ≤ nota ≤ 100 ENTONCES

   agregar nombre a cursos

   agregar nota a notas

   mostrar "Curso registrado correctamente."
```

```
  SINO
    mostrar "Nota inválida."
  FIN_SI
FIN_PROCEDIMIENTO

// 2. Mostrar todos los cursos y notas
PROCEDIMIENTO mostrar_cursos()
  SI cursos está vacío ENTONCES
    mostrar "No hay cursos registrados"
    RETORNAR
  FIN_SI
  PARA i desde 0 hasta longitud(cursos)-1 HACER
    imprimir índice+1, cursos[i], "→", notas[i]
  FIN_PARA
FIN_PROCEDIMIENTO

// 3. Calcular promedio general
PROCEDIMIENTO promedio()
  SI notas está vacío ENTONCES
    mostrar "No hay cursos registrados"
    RETORNAR
  FIN_SI
  prom ← suma(notas) / longitud(notas)
  mostrar prom formateado
FIN_PROCEDIMIENTO
```

```
// 4. Contar aprobados y reprobados
PROCEDIMIENTO contar_aprobados_reprobados()
  aprobados ← 0
  reprobados ← 0
  PARA cada nota EN notas HACER
    SI nota ≥ 61 ENTONCES aprobados ← aprobados + 1
    SINO reprobados ← reprobados + 1
    FIN_SI
  FIN_PARA
  mostrar aprobados, reprobados
FIN_PROCEDIMIENTO

// 5. Búsqueda lineal por nombre
PROCEDIMIENTO busqueda_curso_lineal()
  pedir nombre_buscar
  PARA i desde 0 hasta longitud(cursos)-1 HACER
    SI minusculas(cursos[i]) = minusculas(nombre_buscar) ENTONCES
      mostrar curso y nota encontrados
      RETORNAR
    FIN_SI
  FIN_PARA
  mostrar "Curso no encontrado"
FIN_PROCEDIMIENTO

// 6. Actualizar nota (registra en historial)
PROCEDIMIENTO actualizar_nota()
```

```
 pedir nombre
 SI nombre está en cursos ENTONCES
  i ← índice de nombre en cursos
  mostrar nota actual
  pedir nueva_nota
  SI nueva_nota es número y 0 ≤ nueva_nota ≤ 100 ENTONCES
   agregar a historial_cambios el registro "curso | nota anterior | nueva nota"
   notas[i] ← nueva_nota
   mostrar "Nota actualizada"
  SINO
   mostrar "Nota inválida"
  FIN_SI
 SINO
  mostrar "Curso no encontrado"
 FIN_SI
FIN_PROCEDIMIENTO

// 7. Eliminar curso por posición
PROCEDIMIENTO eliminar_curso()
 mostrar_cursos()
 pedir pos (número del curso a eliminar)
 pos ← pos - 1 // convertir a índice
 SI 0 ≤ pos < longitud(cursos) ENTONCES
  eliminar cursos[pos] y notas[pos]
  mostrar curso eliminado y su nota
 SINO
```

```
   mostrar "Número inválido"
  FIN_SI
FIN_PROCEDIMIENTO

// 8. Ordenar por nota (ej. burbuja, de menor a mayor)
PROCEDIMIENTO ordenar_por_nota()
  aplicar algoritmo de ordenamiento (manteniendo paralelo cursos ↔ notas)
  mostrar_cursos()
FIN_PROCEDIMIENTO

// 9. Ordenar por nombre (ej. burbuja, alfabético)
PROCEDIMIENTO ordenar_por_nombre()
  aplicar algoritmo de ordenamiento por cursos (comparar en minúsculas)
  mostrar_cursos()
FIN_PROCEDIMIENTO

// 10. Búsqueda binaria por nombre (requiere lista ordenada)
PROCEDIMIENTO buscar_curso_binario()
  ordenar_por_nombre()
  pedir nombre_buscar
  izq ← 0
  der ← longitud(cursos) - 1
  MIENTRAS izq ≤ der HACER
    medio ← (izq + der) // 2
    SI minusculas(cursos[medio]) = minusculas(nombre_buscar) ENTONCES
      mostrar curso y nota encontrados
```

```
    RETORNAR
   SINO SI minusculas(nombre_buscar) < minusculas(cursos[medio]) ENTONCES
    der ← medio - 1
   SINO
    izq ← medio + 1
   FIN_SI
  FIN_MIENTRAS
  mostrar "Curso no encontrado"
FIN_PROCEDIMIENTO

// 11. Simular cola de solicitudes de revisión
PROCEDIMIENTO simular_cola()
  crear cola local vacía
  MIENTRAS VERDADERO HACER
   mostrar submenú de la cola (agregar, atender, mostrar, salir)
   leer opción
   SI opción = "1" ENTONCES
    pedir nombre_solicitud
    agregar a cola
   SINO SI opción = "2" ENTONCES
    SI cola no está vacía ENTONCES
     atender ← eliminar primer elemento de la cola
     mostrar atendida
    SINO
     mostrar "No hay solicitudes"
    FIN_SI
```

```
   SINO SI opción = "3" ENTONCES
     mostrar elementos de la cola (si hay)
   SINO SI opción = "4" ENTONCES
     salir del bucle
   SINO
     mostrar "Opción inválida"
   FIN_SI
  FIN_MIENTRAS
FIN_PROCEDIMIENTO

// 12. Mostrar historial de cambios (pila LIFO)
PROCEDIMIENTO mostrar_historial()
  SI historial_cambios no está vacío ENTONCES
    PARA cada cambio EN historial_cambios en orden inverso HACER
      mostrar cambio
    FIN_PARA
  SINO
    mostrar "No hay cambios registrados"
  FIN_SI
FIN_PROCEDIMIENTO

// BUCLE PRINCIPAL
MIENTRAS VERDADERO HACER
  mostrar_menu()
  leer opcion_usuario
  SEGÚN opcion_usuario HACER
```

```
    1: registrar_curso()
    2: mostrar_cursos()
    3: promedio()
    4: contar_aprobados_reprobados()
    5: busqueda_curso_lineal()
    6: actualizar_nota()
    7: eliminar_curso()
    8: ordenar_por_nota()
    9: ordenar_por_nombre()
    10: buscar_curso_binario()
    11: simular_cola()
    12: mostrar_historial()
    13: mostrar "Gracias..." ; SALIR BUCLE
    OTRO: mostrar "Opción inválida"
  FIN_SEGÚN

  pedir "¿Desea realizar otra operación? (s/n)"
  SI respuesta ≠ "s" ENTONCES
    mostrar "Programa finalizado."
    SALIR BUCLE
  FIN_SI
 FIN_MIENTRAS

FIN
```