Universidad San Carlos de Guatemala

Facultad de Ingeniería

Estructura de Datos, Sección B

Ing. Álvaro Obrayan Hernández García

Auxiliar: Carlos Javier Castro González

Primer Semestre 2024

Tema:

Manuel Técnico – Pixel Print Studio – Fase 3

Nombre: Carlos Manuel Lima y Lima

Registro Académico: 202201524

CUI: 3009368850101

Guatemala, 29 de abril del 2024.

LECTURA DE ARCHIVO JSON CON FPM FORTRAN

La subrutina carga_masiva en Fortran es un procedimiento que se encarga de la lectura y procesamiento de un archivo JSON. Inicialmente, solicita al usuario el nombre del archivo a procesar. Luego, inicializa un objeto JSON y carga el archivo especificado. Posteriormente, recorre cada elemento del archivo JSON, extrayendo y almacenando los datos relevantes.

- **Lectura del nombre del archivo:** La línea read(*,*) documento_grafo lee el nombre del archivo JSON desde la entrada estándar (generalmente el teclado) y lo almacena en la variable documento_grafo.
- **Inicialización y carga del archivo JSON:** Las líneas call json%initialize() y call json%load(filename=documento_grafo) inicializan el objeto JSON y cargan el archivo JSON especificado por el usuario.
- **Obtención de información del archivo JSON:** Las líneas call json%info('',n_children=size_grafo) y call json%get_core(jsonc) obtienen información sobre el archivo JSON y la almacenan en las variables size_grafo y jsonc.
- **Bucle sobre los elementos del archivo JSON:** Las líneas do contador_grafo = 1, size_grafo y do contador_ruta = 1, size_ruta son bucles que iteran sobre los elementos del archivo JSON. Dentro de estos bucles, se extraen los datos de cada elemento y se almacenan en las variables correspondientes.
- **Extracción de datos de cada elemento:** Las líneas que contienen call jsonc%get_child(...) y call jsonc%get(...) se utilizan para extraer los datos de cada elemento del archivo JSON. Los datos extraídos se almacenan en las variables correspondientes.
- **Destrucción del objeto JSON:** Finalmente, la línea call json%destroy() destruye el objeto JSON, liberando así los recursos que había reservado.

Este código es un ejemplo eficiente de cómo se puede manejar y procesar archivos JSON en Fortran. Hace uso de las capacidades de lectura y escritura de archivos de Fortran, así como de su capacidad para interactuar con objetos JSON a través de una biblioteca específica.

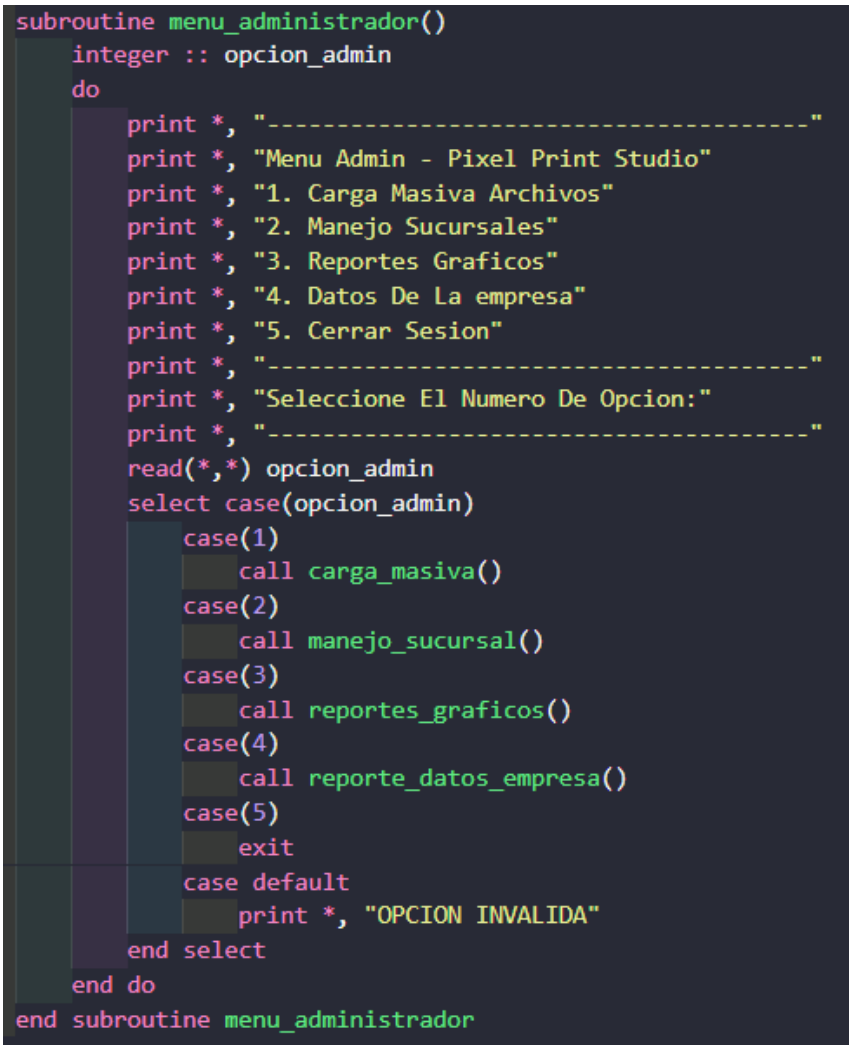
```
subroutine carga_masiva_ruta()
    print *, "----------------------------------------"
    print *, "CARGA MASIVA RUTAS"
    print *, "----------------------------------------"
    print *, "Ingrese el nombre del documento de ruta:"
    print *, "----------------------------------------"
    read(*,*) documento_grafo
    print *, "----------------------------------------"
    call json%initialize()
    call json%load(filename=documento_grafo)
    call json%info('',n_children=size_grafo)
    call json%get_core(jsonc)
    call json%get('', lista_puntero_r, grafo_encontrado)
    do contador_grafo = 1, size_grafo
        call jsonc%get_child(lista_puntero_r, contador_grafo, puntero_r, grafo_encontrado)
        call jsonc%get_child(puntero_r, 'grafo', atributo_puntero_r, grafo_encontrado)
        call jsonc%info(atributo_puntero_r, n_children=size_ruta)
        do contador_ruta = 1, size_ruta
            call jsonc%get_child(atributo_puntero_r, contador_ruta, puntero_aux, grafo_encontrado)
            call jsonc%get_child(puntero_aux, 's1', atributo_puntero_aux, grafo_encontrado)
            call jsonc%get(atributo_puntero_aux, s1)
            call jsonc%get_child(puntero_aux, 's2', atributo_puntero_aux, grafo_encontrado)
            call jsonc%get(atributo_puntero_aux, s2)
            call jsonc%get_child(puntero_aux, 'distancia', atributo_puntero_aux, grafo_encontrado)
            call jsonc%get(atributo_puntero_aux, distancia)
            call jsonc%get_child(puntero_aux, 'imp_mantenimiento', atributo_puntero_aux, grafo_encontrado)
            call jsonc%get(atributo_puntero_aux, imp_mantenimiento)
            print *, "--------------"
            print *, "s1: ", s1
            print *, "s1: ", s2
            print *, "distancia: ", distancia
            print *, "imp_mantenimiento: ", imp_mantenimiento
        end do
    end do
    call json%destroy()
end subroutine carga_masiva_ruta
```

MANEJO DE BUCLES PARA REPRESENTAR UN MENÚ DE SELECCIÓN

El código proporcionado a continuación es una subrutina en Fortran llamada menu_administrador que presenta un menú de opciones al usuario y realiza diferentes acciones basadas en la opción seleccionada por el usuario.

- **Definición de la Subrutina:** La línea subroutine menu_administrador() define la subrutina menu_administrador.
- **Declaración de Variables:** La línea integer :: opcion_admin declara una variable entera llamada opcion_admin que se utiliza para almacenar la opción seleccionada por el usuario.
- **Bucle Principal:** El bucle do sin un número de iteraciones especificado se ejecuta indefinidamente hasta que se encuentre una sentencia exit.
- **Impresión del Menú:** Las líneas con print *, "..." se utilizan para imprimir el menú de opciones en la consola.
- **Lectura de la Opción del Usuario:** La línea read(*,*) opcion_admin lee la opción del usuario desde la entrada estándar (generalmente el teclado) y la almacena en la variable opcion_admin.
- **Selección de la Acción:** La estructura select case(opcion_admin) se utiliza para seleccionar una acción basada en la opción del usuario. Cada case(n) corresponde a una opción del menú, y la sentencia call dentro de cada case llama a una subrutina diferente. Si el usuario introduce una opción que no está en el menú, se ejecuta la sentencia bajo case default y se imprime “OPCION INVALIDA”.
- **Fin de la Subrutina:** La línea end subroutine menu_administrador marca el final de la subrutina.

Este código es un ejemplo clásico de cómo implementar un menú de selección en Fortran. Utiliza un bucle para presentar continuamente el menú al usuario y una estructura select case para realizar acciones basadas en la entrada del usuario.



```
subroutine menu_administrador()
    integer :: opcion_admin
    do
        print *, "---------------------------------------"
        print *, "Menu Admin - Pixel Print Studio"
        print *, "1. Carga Masiva Archivos"
        print *, "2. Manejo Sucursales"
        print *, "3. Reportes Graficos"
        print *, "4. Datos De La empresa"
        print *, "5. Cerrar Sesion"
        print *, "---------------------------------------"
        print *, "Seleccione El Numero De Opcion:"
        print *, "---------------------------------------"
        read(*,*) opcion_admin
        select case(opcion_admin)
            case(1)
                call carga_masiva()
            case(2)
                call manejo_sucursal()
            case(3)
                call reportes_graficos()
            case(4)
                call reporte_datos_empresa()
            case(5)
                exit
            case default
                print *, "OPCION INVALIDA"
        end select
    end do
end subroutine menu_administrador
```

MÓDULO DE ENCRIPTACIÓN

El módulo modulo_encriptacion en Fortran es una colección de funciones y subrutinas diseñadas para realizar operaciones de encriptación SHA-256. Las funciones y subrutinas incluyen sha256, dirty_sha256, sha256b, swap32, swap64, swap64a, ch, maj, cs0, cs1, ms0, ms1 y consume_chunk.

Funciones de Encriptación: Las funciones sha256, dirty_sha256 y sha256b son responsables de la encriptación SHA-256. Estas funciones toman una cadena de entrada y devuelven su hash SHA-256.

Funciones de Swap: Las funciones swap32, swap64 y swap64a se utilizan para cambiar el orden de los bytes en un número entero de 32 o 64 bits. Esto es útil para convertir entre diferentes formatos de endianidad.

Funciones de Operaciones Lógicas: Las funciones ch, maj, cs0, cs1, ms0 y ms1 realizan varias operaciones lógicas utilizadas en el algoritmo SHA-256. Estas operaciones incluyen operaciones de cambio de bits, operaciones AND y OR, y otras operaciones de bits.

Subrutina consume_chunk: Esta subrutina se encarga de procesar un bloque de datos de la cadena de entrada. Lee los datos, realiza las operaciones necesarias y actualiza el estado del algoritmo SHA-256.

Este módulo es un ejemplo de cómo se puede implementar el algoritmo de encriptación SHA-256 en Fortran. Hace uso de varias funciones y subrutinas para dividir el algoritmo en partes manejables y mantener el código organizado.

```
module modulo_encriptacion
   use iso_c_binding
   implicit none
   private
   public sha256
   public dirty_sha256
   public sha256b
   public ms0
   public ms1
   public cs0
   public cs1
   public maj
   public ch
   public swap32
   public swap64
   public swap64a
   public consume_chunk

   contains

   function sha256(str)
      implicit none
      character(len=64) :: sha256
      character(len=*), intent(in) :: str
      sha256 = sha256b(str, 1)
      end function sha256
```

Resultado de una encriptación:

```
Password: A9BC5F21B71268000016208995 9D2D5367B1F722FD79576D8EC2D81AF73FB144
```

TABLA HASH

Resumen: El módulo modulo_tabla_hash en Fortran es una implementación de una tabla hash para almacenar y manipular datos de técnicos. Las operaciones clave incluyen la inserción de un técnico en la tabla hash, la resolución de colisiones en la tabla hash, la obtención de la posición de un técnico en la tabla hash, la impresión de los detalles de un técnico, la lista de todos los técnicos en la tabla hash y la generación de una representación gráfica de la tabla hash.

Definición del Módulo: La línea module modulo_tabla_hash define el módulo modulo_tabla_hash. Un módulo en Fortran es una colección de declaraciones de datos y procedimientos que pueden ser utilizados por otros programas o módulos.

Tipo de Datos Técnico: El tipo de datos tecnico se define para almacenar la información de un técnico, que incluye DPI, nombre, apellido, dirección, teléfono y género.

Tipo de Datos TablaHash: El tipo de datos TablaHash se define para representar una tabla hash. Contiene un arreglo de técnicos y un contador de elementos. También contiene varios procedimientos para manipular la tabla hash.

Funciones y Subrutinas

- **insertar:** Esta subrutina inserta un nuevo técnico en la tabla hash. Toma como entrada los detalles del técnico (DPI, nombre, apellido, dirección, teléfono y género) y los inserta en la tabla hash en la posición correspondiente. Si la tabla hash se llena más allá de un cierto porcentaje, se realiza un rehashing para aumentar el tamaño de la tabla.
- **imprimir:** Esta subrutina imprime los detalles de un técnico específico. Toma como entrada el DPI de un técnico y busca el técnico correspondiente en la tabla hash. Si el técnico se encuentra, imprime sus detalles.
- **listar_tecnico:** Esta subrutina lista todos los técnicos en la tabla hash. Recorre todos los elementos de la tabla hash e imprime los detalles de cada técnico.
- **grafica_tabla:** Esta subrutina genera una representación gráfica de la tabla hash. Utiliza la biblioteca Graphviz para crear un gráfico que muestra cómo están organizados los técnicos en la tabla hash.
- **resolver_colision:** Esta subrutina se encarga de resolver las colisiones en la tabla hash. Cuando dos técnicos tienen la misma posición en la tabla hash, esta subrutina encuentra una nueva posición para el segundo técnico.
- **obtener_posicion:** Esta función calcula la posición de un técnico en la tabla hash. Toma como entrada el DPI de un técnico y devuelve la posición correspondiente en la tabla hash.
- **rehashing:** Esta función se encarga de realizar un rehashing de la tabla hash. Cuando la tabla hash se llena más allá de un cierto porcentaje, esta función crea una nueva tabla hash de mayor tamaño y reinserta todos los técnicos en la nueva tabla.
- **generar_grafica**: Esta subrutina genera un archivo de gráfico a partir de un código DOT. Toma como entrada el nombre del gráfico y el código DOT, y genera un archivo de gráfico correspondiente.

Este módulo es un ejemplo de cómo se puede implementar una tabla hash en Fortran. Hace uso de varias funciones y subrutinas para dividir el algoritmo en partes manejables y mantener el código organizado.

Módulos importantes de la tabla hash:

```
module modulo_tabla_hash
    implicit none
    private
    integer :: tamano_tabla = 7
    integer, parameter :: porcentaje_maximo = 70
    type tecnico
        integer(8) :: dpi, telefono
        character(:), allocatable::nombre, apellido, genero, direccion
    end type tecnico
    type, public :: TablaHash
        integer :: elemento = 0
        type(tecnico), allocatable :: arreglo(:)
        contains
        procedure :: insertar, imprimir, listar_tecnico, grafica_tabla
        procedure, private :: resolver_colision
    end type TablaHash
```

```
function rehashing(arreglo_anterior) result(nueva_tabla)
    type(tecnico), intent(in) :: arreglo_anterior(:)
    integer :: i
    type(TablaHash) :: nueva_tabla
    tamano_tabla = tamano_tabla*2
    allocate(nueva_tabla%arreglo(0:tamano_tabla-1))
    nueva_tabla%arreglo(:)%dpi = -1
    do i = 1, size(arreglo_anterior)
        if(arreglo_anterior(i)%dpi /= -1) then
        call nueva_tabla%insertar(arreglo_anterior(i)%dpi,arreglo_anterior(i)%nombre,&
        arreglo_anterior(i)%apellido, &
        arreglo_anterior(i)%direccion,&
        arreglo_anterior(i)%telefono, arreglo_anterior(i)%genero)
        end if
    end do
end function rehashing

subroutine resolver_colision(self, posicion)
    class(TablaHash), intent(inout) :: self
    integer(8), intent(inout) :: posicion
    do while(self%arreglo(posicion)%dpi /= -1)
        posicion = posicion + 1
        posicion = mod(posicion, tamano_tabla)
    end do
end subroutine resolver_colision

function obtener_posicion(dpi) result(posicion)
    integer(8), intent(in) :: dpi
    integer(8) :: posicion
    posicion = mod(dpi,tamano_tabla)
end function obtener_posicion
```

ÁRBOL AVL

El módulo modulo_arbol_avl en Fortran es una implementación de un árbol AVL para almacenar y manipular datos de sucursales. Las operaciones clave incluyen la inserción de un nodo en el árbol AVL (insertar_nodo), la búsqueda de un nodo en el árbol (obtener_nodo y valor_existe), y la visualización del árbol (graficar_arbol).

Definición del Módulo: La línea module modulo_arbol_avl define el módulo modulo_arbol_avl. Un módulo en Fortran es una colección de declaraciones de datos y procedimientos que pueden ser utilizados por otros programas o módulos.

Tipo de Datos nodo_avl y arbol_avl: El tipo de datos nodo_avl se define para almacenar la información de un nodo en el árbol AVL, que incluye valor, departamento, dirección, contraseña, altura y punteros a los nodos hijos. El tipo de datos arbol_avl se define para representar un árbol AVL, que contiene un puntero a la raíz del árbol.

Funciones y Subrutinas:

- **insertar_nodo:** Esta subrutina inserta un nuevo nodo en el árbol AVL. Toma como entrada los detalles del nodo (valor, departamento, dirección, contraseña) y los inserta en el árbol AVL en la posición correspondiente. Si el árbol AVL se llena más allá de un cierto porcentaje, se realiza un rehashing para aumentar el tamaño del árbol.
- **obtener_nodo:** Esta función busca un nodo en el árbol AVL basándose en el valor y la contraseña. Si el nodo se encuentra, devuelve el nodo; de lo contrario, devuelve null.
- **valor_existe:** Esta función verifica si un nodo con un valor y contraseña específicos existe en el árbol AVL. Devuelve verdadero si el nodo existe y falso en caso contrario.
- **graficar_arbol:** Esta subrutina genera una representación gráfica del árbol AVL. Utiliza la biblioteca Graphviz para crear un gráfico que muestra cómo están organizados los nodos en el árbol AVL.
- **buscar_recursivo:** Esta función busca un nodo en el árbol AVL de manera recursiva. Toma como entrada el valor y la contraseña y devuelve el nodo si se encuentra.
- **rotacionIzquierda y rotacionDerecha:** Estas funciones realizan rotaciones izquierda y derecha en el árbol AVL, respectivamente. Las rotaciones son operaciones fundamentales en los árboles AVL que se utilizan para mantener el árbol equilibrado.
- **obtenerMayorDeMenores:** Esta subrutina busca el nodo con el mayor valor en el subárbol izquierdo de un nodo dado. Se utiliza durante la eliminación de un nodo en un árbol AVL.
- **maximo:** Esta función toma dos números como entrada y devuelve el mayor de los dos.
- **obtenerBalance:** Esta función calcula el factor de equilibrio de un nodo en el árbol AVL. El factor de equilibrio es la diferencia entre las alturas del subárbol derecho y el subárbol izquierdo de un nodo.
- **obtenerAltura:** Esta función devuelve la altura de un nodo en el árbol AVL.
- **RoamTree:** Esta subrutina recorre el árbol AVL y genera el código DOT para cada nodo. Este código DOT se utiliza luego para generar la representación gráfica del árbol AVL.
- **obtener_direccion_memoria_avl:** Esta función toma un nodo como entrada y devuelve su dirección de memoria.
- **generar_grafica:** Esta subrutina genera un archivo de gráfico a partir de un código DOT. Toma como entrada el nombre del gráfico y el código DOT, y genera un archivo de gráfico correspondiente. Luego abre el archivo de gráfico en el visor de PDF predeterminado.

Este módulo es un ejemplo de cómo se puede implementar un árbol AVL en Fortran. Hace uso de varias funciones y subrutinas para dividir el algoritmo en partes manejables y mantener el código organizado.

Módulos importantes del árbol AVL:

Arbol_Avl.f90

```
module modulo_arbol_avl
    use modulo_encriptacion
    use modulo_tabla_hash
    implicit none
    private
    public :: nodo_avl
    public :: arbol_avl
    type :: nodo_avl
        integer :: valor
        character(:), allocatable :: departamento
        character(:), allocatable :: direccion
        character(:), allocatable :: contrasena
        integer :: altura = 1
        type(nodo_avl), pointer :: derecha => null()
        type(nodo_avl), pointer :: izquierda => null()
        type(TablaHash) :: tabla
    end type
    type arbol_avl
        type(nodo_avl), pointer :: raiz => null()
    contains
        procedure :: insertar_nodo
        procedure :: obtener_nodo
        procedure :: valor_existe
        procedure :: graficar_arbol
    end type arbol_avl
```

```
subroutine insertar_nodo(self, valor, departamento, direccion, contrasena)
    class(arbol_avl), intent(inout) :: self
    integer, intent(in) :: valor
    character(len=*), intent(in) :: departamento, direccion, contrasena
    call insertar_recursivo(self%raiz, valor, departamento, direccion, contrasena)
end subroutine insertar_nodo

recursive subroutine insertar_recursivo(raiz, valor, departamento, direccion, contrasena)
    type(nodo_avl), pointer, intent(inout) :: raiz
    integer, intent(in) :: valor
    character(len=*), intent(in) :: departamento, direccion, contrasena
    if(.not. associated(raiz)) then
        allocate(raiz)
        raiz = nodo_avl(valor=valor, departamento=departamento, direccion=direccion, contrasena=contrasena)
    else if(valor < raiz%valor) then
        call insertar_recursivo(raiz%izquierda, valor, departamento, direccion, contrasena)
    else if(valor > raiz%valor) then
        call insertar_recursivo(raiz%derecha, valor, departamento, direccion, contrasena)
    end if
    raiz%altura = maximo(obtenerAltura(raiz%izquierda), obtenerAltura(raiz%derecha)) + 1
    if(obtenerBalance(raiz) > 1) then
        if(obtenerBalance(raiz%derecha) < 0) then
            raiz%derecha => rotacionDerecha(raiz%derecha)
            raiz => rotacionIzquierda(raiz)
        else
            raiz => rotacionIzquierda(raiz)
        end if
    end if
    if(obtenerBalance(raiz) < -1) then
        if(obtenerBalance(raiz%izquierda) > 0) then
            raiz%izquierda => rotacionIzquierda(raiz%izquierda)
            raiz => rotacionDerecha(raiz)
        else
            raiz => rotacionDerecha(raiz)
        end if
    end if
end subroutine insertar_recursivo
```