**RESUMEN REGISTROS Y ARCHIVOS**

Fuente: Material oficial de la cátedra de AyED Mód 3 Struct y File.doc

*SI BIEN SE RESUMEN AQUIÍ ALGUNOS CONCEPTOS/COMANDOS/ALGORITMOS PRINCIPALES, DEBE LEERSE Y MANEJARSE EL MÓD 3 STRUCT Y FILE COMPLETO)*

**ARCHIVOS DE TEXTO**

Secuencia de líneas compuestas por cero o más caracteres, con un fin de línea y una marca de final de archivo.

**Algunas funciones/operaciones**

|  |  |
| --- | --- |
| Función | Descripción |
| FILE \*f; | Define f como puntero a FILE |
| f = fopen (“archivo”, “w”); | Asocia f a un flujo |
| fclose(f); | Vacía el flujo y cierra el archivo asociado |
| fflush(f); | Produce el vaciado de los flujos |
| remove(“archivo”); | El archivo ya no queda accesible |
| rename(“viejo”, “nuevo”); | Renombra con nuevo el viejo nombre |
| fprintf( f, “%d”, valor); | Escritura con formato en un flujo |
| fscanf( f, “%d”, &valor); | Lectura con formato desde un flujo |
| c = fgetc(f); | Lee un carácter desde el flujo |
| ungetc (c, f); | Retorna el carácter al flujo y retrocede |
| fputc(c, f); | Escribe un carácter en el flujo |
| fgets(s, n, f); | Lee hasta n-1 carácter del flujo en s |
| fputs(s, f); | Escribe la cadena s en el flujo |
| feof(f) | Retorna no cero si el indicador de fin está activo |
| ferror(f); | Retorna no cero si el indicador de error está activo |
| clearerr(f); | Desactiva los indicadores de error |

|  |  |
| --- | --- |
| **Equivalencias que pueden utilizarse entre C y la representación algorítmica para flujos de texto**  **Siendo int c; char s[n]; float r; FILE \* f;** | |
| Representacion: | Equivalente a: |
| LeerCaracter(f,c) | c = fgetc(f) |
| LeerCadena(f,s) | fgets(s,n,f) |
| LeerConFormato(f,c,r,s) | fscanf(f,”%c%f%s”,& c,&r,s) |
| GrabarCaracter(f,c) | fputc(c,f) |
| GrabarCadena(f,s) | fputs(f,s) |
| GrabarConFormato(f,c,r,s) | fprintf(f,”%c %f %s \n”, c,r,s) |

**Operaciones simples**

|  |  |
| --- | --- |
| Como función no void | Como función void |
| FILE \* AbrirArchTextoLectura( char nombre[]) {  // Asocia el flujo f con el archivo nombre, lo abre en modo lectura  return fopen(nombre, “r”);  } | void AbrirArchTextoLectura( FILE \* f, char nombre[]) {  // Asocia el flujo f con el archivo nombre, lo abre en  modo lectura  f= fopen(nombre, “r”);  } |
| Luego al invocar en el main  F= AbrirArchTextoLectura(“nombre físico”); | AbrirArchTextoLectura(f, “nombre físico”); |

|  |  |
| --- | --- |
| Como función no void | Como función void |
| FILE \* AbrirArchTextoEscritura( char nombre[]) {  // Asocia el flujo f con el archivo nombre, lo abre en modo lectura  return fopen(nombre, “w”);  } | void AbrirArchTextoEscritura( FILE \* f, char nombre[]) {  // Asocia el flujo f con el archivo nombre, lo abre en  modo lectura  f= fopen(nombre, “w”);  } |
| Luego al invocar en el main  F= AbrirArchTextoEscritura (“nombre físico”); | AbrirArchTextoEscritura(f, “nombre físico”); |

**Ejemplos**

*Dado un archivo de texto leerlo carácter a carácter y mostrar su contenido por pantalla*

|  |  |
| --- | --- |
| Solución en C | Solución algorítmica |
| main(){  FILE \*f1, f2; int c;  f1 = fopen(“entrada”, “r”);  f2 = fopen(“salida”,”w”);  c = fgetc(f1);  while (!feof(f1)) {  fputc(c,f2);  c=fgetc(f1); }  fclose(f1);  fclose(f2);  return 0; } | AbrirArchTextoLectura(f1, “entrada”)  AbrirArchTextoEscritura(f2,”salida”)  LeerCaracter(f1,c)  Mientras(!feof(f1))  GrabarCaracter(f2,c)  LeerCaracter(f1,c)  FinMientras  Cerrar(f1)  Cerrar(f2) |

*Dado un archivo de texto con líneas de no más de 10 caracteres, leerlo por línea y mostrar su contenido por pantalla*

|  |  |
| --- | --- |
| Solución en C | Solución algorítmica |
| main(){  FILE \*f1 ; char s[10+1];  f1 = fopen(“entrada”, “r”);  fgets(s,10+1,f1);  while (!feof(f1)) {  fprintf(“%s/n”,s);  fgets(s,10+1,f1); }  fclose(f1);  return 0; } | AbrirArchTextoLectura(f1, “entrada”)  LeerCadena(f1,s)  Mientras(!feof(f1))  Imprimir(s)  LeerCadena(f1,s)  FinMientras  Cerrar(f1) |

*Dado un archivo de texto con valores encolumnados como indica el ejemplo mostrar su contenido por pantalla. 10 123.45 Juan*

|  |  |
| --- | --- |
| Solución en C | Solución algorítmica |
| main(){  FILE \*f1; int a; float f; char s[10+1];  f1 = fopen(“entrada”, “r”);  fscanf(f1,”%d %f %s”, &a,&f,s);  while (!feof(f1)) {  printf(“%10 d%7.2f%s/n”,a,f,s);  fscanf(f1,”%d %f %s”, &a,&f,s);  }  fclose(f1);  return 0; } | AbrirArchTextoLectura(f1, “entrada”)  LeerConFormato(f1,a,f,s)  Mientras(!feof(f1))  Imprimir(a,f,s)  LeerConFormato(f1,a,f,s)  FinMientras  Cerrar(f1) |

**ARCHIVOS BINARIOS**

**ESTRUCTURA TIPO REGISTRO**

|  |  |
| --- | --- |
| Formato | Ejemplo |
| Struct NombreTipo {  tipo identificador;  tipo identificador;  etc. } | typedef char str30[31]; //p def tipo cadena 30  Struct trProd {  Unsigned CodP;  str30 Desc;  float PU; } |
| Luego hay que definir una variable registro de ese tipo | trProd Rprod |
| Para utilizar los campos el formato se coloca Registro.campo | Rprod.Codp  Rprod.Desc  Rprod.PU |

**Algunas funciones/operaciones**

|  |  |
| --- | --- |
| Función | Descripción |
| FILE \*f; | Define f como puntero a FILE |
| f = fopen (“archivo”, “wb”); | Asocia f a un flujo |
| fclose(f); | Vacía el flujo y cierra el archivo asociado |
| remove(“archivo”); | El archivo ya no queda accesible |
| rename(“viejo”, “nuevo”); | Renombra con nuevo el viejo nombre |
| sizeof(tipo) | Retorna el tamaño de un tipo o identificador |
| SEEK\_CUR | Constante asociada a fseek (lugar actual) |
| SEEK\_END | Constante asociada a fseek (desde el final) |
| SEEK\_SET | Constante asociada a fseek (desde el inicio) |
| fread(&r, tam,cant, f) | Lee cant bloques de tamaño tam del flujo f  tam es n\*recsize(registro) |
| fwrite(&r,tam,cant,f) | Graba cant bloques de tamaño tam del flujo f  tam es n\*recsize(registro) |
| fgetpos(f, pos) | Almacena el valor actual del indicador de posición |
| fsetpos(f,pos) | Define el indicador de posición del archivo en pos |
| ftell(f) | El valor actual del indicador de posición del archivo |
| fseek(f, cant, desde) | Define indicador de posición a partir de una posición. |
|  |  |
|  |  |
|  |  |

Modos de apertura

|  |  |
| --- | --- |
| Modo | Descripción |
| R | Reset Abre archivo de texto para lectura |
| Rt | Idem anterior,explicitando t:texto |
| W | Write Abre archivo de texto para escritura, si el archivo existe se descarta el contenido sin advertencia |
| Wt | Idem anterior,explicitando t:texto |
| Rb | Reset abre archivo binario para lectura |
| Wb | Write Abre archivo binario para escritura, si el archivo existe se descarta el contenido sin advertencia |
| + | Agrega la otra modalidad a la de apertura |

Ejemplos de declaración, apertura, lectura y grabación

|  |  |
| --- | --- |
| Formato | Ejemplo |
| FILE \*var de archivo; //identificador para el archivo  var de archivo = fopen ("nombre externo", "modo de apertura");  if (f != NULL) {  }  fwrite (&registro, sizeof(registro), 1, var de archivo);  fread (&registro, sizeof(registro), 1, var de archivo);  feof(var de archivo) si no pudo leer devuelve distinto de cero (o sea true)  fclose(var de archivo) | File \*f;  f = fopen ("productos.dat","wb");  if (f != NULL) { acc }  else  acciones de no existe el archivo;  fwrite (&Rprod, sizeof(Rprod), 1, f);  fread (&Rprod, sizeof(Rprod), 1, f);  while (!feof(f)) {  acc  };  fclose(f); |

**Moviendo el puntero del archivo**

Ftell indica el nro de byte al que está apuntando el indicador de pos del archivo

Ejemplo: Pos = ftell(f)

Fseek reubicar el indicador de pos del archivo

Ejemplos:

fseek(f, 0, SEEK\_SET) va al comienzo del archivo

fseek(f, 0, SEEK\_END) va al final del archivo

fseek(f, (n-1)x sizeof(reg), SEEK\_SET) va al registro nro n (dde el comienzo se mueve n-1 bloques más)

fseek(f, -1x sizeof(reg), SEEK\_CUR) va al registro anterior al actual

**Funciones predefinidas a partir de las existentes**

Utilizando las siguientes funciones (declaradas en el archivo stdio.h y que se utilizan agregando #include <stdio.h> a nuestro programa)

 fopen - Abre un archivo.

 fwrite - Graba datos en el archivo.

 fread - Lee datos desde el archivo.

 feof - Indica si quedan o no más datos para ser leidos desde el archivo.

 fseek - Permite reubicar el indicador de posición del archivo.

ftell - Indica el nro de byte al que está apuntando el indicador de posición del archivo.

 fclose - Cierra el archivo.

desarrollaremos las siguientes que nos permitirán operar con archivos de registros:

 **seek** – Mueve el indicador de posición de un archivo al inicio de un determinado reg.

 **fileSize** – Indica cuantos registros tiene un archivo.

 **filePos** – Retorna el nro de registro que está siendo apuntado por el indicador de posición.

 **read** – Lee un registro del archivo.

 **write** – Graba un registro en el archivo.

**void** seek(FILE\* arch, **int** recSize, **int** n) {

fseek(arch, n\*recSize,SEEK\_SET); // SEEK\_SET hace referencia al comienzo del archivo

}

**long** fileSize(FILE\* f, **int** recSize) {

// tomo la posicion actual

**long** curr=ftell(f);

// muevo el puntero al final del archivo

fseek(f,0,SEEK\_END); // SEEK\_END hace referencia al final del archivo

// tomo la posicion actual (ubicado al final)

**long** ultimo=ftell(f);

// vuelvo a donde estaba al principio fseek(f,curr,SEEK\_SET);

**return** ultimo/recSize;

}

**long** filePos(FILE\* arch, **int** recSize) {

**return** ftell(arch)/recSize;

}

**void read**(FILE\* arch, Treg reg, **int** recSize) {

fread(&reg, recsize, 1, arch)

}

**void write**(FILE\* arch, Treg reg, **int** recSize) {

fwrite(&reg, recsize, 1, arch)

}

**ALGORITMOS FRECUENTES**

**BusquedaDirectaArchivo**(Dato\_Resultado B: TipoArchivo; Posic: Entero, Dato\_Resultado R: TipoRegistro): una accion

*Utilizar este algoritmo si los datos en el archivo están ordenados por un campo clave y la posición es conocida o es una Posición Única y Predecible*

PRE: B: Archivo binario en el que se debe buscar con clave sin repetir

Posic: Posición donde se encuentra la clave buscada que puede ser PUP POS: R: el registro completo de la clave buscada

ALGORITMO

Seek (B, Posic);{*ubica el puntero en la posición conocida o en la PUP*} LeerRegistro (B,R);{*lee el registro en la posición definida*}

**BusquedaBinariaArchivo**(Dato\_Resultado B: TipoArchivo; Dato Clave:Tinfo; Dato\_resultado

Posic: Entero): una accion

*Utilizar este algoritmo si los datos en el archivo están ordenados por un campo clave y se busca por ese campo. Debe tenerse en cuenta que si la clave es posicional se deberá utilizar búsqueda directa ya que la diferencia en eficiencia está dada entre 1, para la búsqueda directa y log2N para la binaria*

PRE: B: Archivo binario en el que se debe buscar con clave sin repetir

Clave : Valor Buscado

POS: Posic: Posición donde se encuentra la clave, ó (-1) si no esta

LEXICO

j : Entero; u,m : Entero;

ALGORITMO

Posic = -1

Pri=0

U = filesize(B); //la función predefinida

MIENTRAS (Pri < = U y Posic = -1) HACER M = (Pri + U ) div 2

Seek(B, M); // la función predefinida

LeerRegistro(B,R) SI R.clave = Clave ENTONCES

Posic = M;

SI\_NO

SI Clave > R.Clave

ENTONCES

Pri = M+1

SI\_NO

U = M - 1

FIN\_SI

FIN\_SI

FIN\_MIENTRAS;

FIN. // Búsqueda binaria en archivo

**ApareoBinarioV1**(Dato\_Resultado A, B, C: TipoArchivo): una acción

*Requiere al menos dos estructuras, con al menos un campo en común y se requiere procesarlos simultáneamente, intercalándolos por la clave común.*

*Se suponen los archivos ordenados crecientes, en caso de que el orden sea decreciente solo habrá que modificar los operadores de relación.*

PRE: A, B: Archivos binarios existente, ordenado

C: Archivo Binario a crear

Ra,Rb: TipoRegistro {Variables locales para las lecturas de los registros} POS: Intercala dos archivos manteniendo el orden de las claves

*La primera versión presenta tres ciclos de repetición. Un primer ciclo mientras haya datos en ambos archivos y a continuación, cuando uno de ellos se termina, hacer un ciclo de repetición*

*hasta agotar el que no se agotó. Como no se sabe a priori cual de los dos se agotara, se hacen los ciclos para ambos, desde luego que uno de ellos no se ejecutara. Mientras hay datos en ambos se requiere saber cual tiene la clave menor para procesarlo primero. Al agotarse uno de los archivos el otro archivo se lo procesa directamente sin necesidad de hacer ninguna comparación.*

LEXICO

R a. Rb: TipoRegistro; {para la lectura}

F a, Fb: Boolean {valor centinela que indica si se pudo leer el registro}

ALGORITMO

AbrirArchBinarioLectura(A); AbrirArchBinarioLectura(B); AbrirArchBinarioEscritura(C); LeerRegistro (A,Ra) LeerRegistro (A,Ra)

{*Inicializacion de variables generales*}

MIENTRAS ((! feof (A)) Y (!feof(B))) HACER { *mientras haya datos en ambos archivos*}

SI (Ra.clave < Rb.clave) {*si la clave del registro a es menor lo procesa y avanza*} ENTONCES

GrabarRegistro(C, Ra) {*Procesa el registro de A y avanza*} LeerRegistro (A,Ra)

SI\_NO

GrabarRegistro(C, Rb) {*Procesa el registro de B y avanza*}

LeerRegistro (B,Rb)

FIN\_SI

FIN\_MIENTRAS

{*agotar los que no se agotaron*}

MIENTRAS (!feof (A) ) HACER { *agota A si es el que no termino*} GrabarRegistro(C, Ra) {*Procesa el registro de A y avanza*} LeerRegistro (A,Ra)

FIN\_MIENTRAS

MIENTRAS (!feof(B) ) HACER { *agota B si es el que no termino*} GrabarRegistro(C, Rb) {*Procesa el registro de A y avanza*} LeerRegistro (A,Rb)

FIN\_MIENTRAS

Cerrar(A); Cerrar(B); Cerrar(C); FIN. // *Recorrido apareo*

**CorteControlBinario**(Dato\_Resultado B: TipoArchivo): una acción

*Recorre un archivo completo con corte de Control.*

PRE: B: Archivo binario existente, ordenado, con una clave que se repite

R: TipoRegistro {Variable local para las lecturas de los registros}

POS: Muestra el contenido completo del archivo por pantalla agrupado por clave común.

*Se hace una lectura anticipada para ver si hay datos. Se ingresa en un ciclo de repetición mientras haya datos, eso lo controla la variable F. Aquí se toma en una variable la clave de control. Se ingresa a un ciclo interno mientras haya datos y la clave leída sea igual al valor que se está controlando. Cuando se lee un registro cuyo valor no es igual al que se venía leyendo se produce un corte en el control que se estaba ejerciendo. Al salir del ciclo interno se verifica si sigue habiendo datos, si aún quedan, se toma la nueva clave como elemento de control y se sigue recorriendo, si no hubiera más se termina el proceso completo*

LEXICO

R : TipoRegistro; {para la lectura}

Anterior : Dato del tipo de la clave por la que se debe agrupar

ALGORITMO AbrirArchBinarioLectura(B); LeerRegistro (B,R)

{*Inicializacion variables generales si corresponde*}

MIENTRAS (! feof (F)) HACER {*Itera mientras haya datos, es decir F = Falso*} Anterior  r.clave {*conserva en anterior la clave por la cual agrupar*}

{*inicialización de variables de cada grupo*} MIENTRAS((! feof(F)) Y (Anterior = R.Clave) HACER

{*recorre en ciclo interno mientras haya datos y la clave sea la misma*}

{EJECUTAR LAS ACCIONES DE CADA REGISTRO}

LeerRegistro (B,R) FIN\_MIENTRAS

{*Acciones generales, si corresponden, de cada grupo*} FIN\_MIENTRAS

{*Acciones generales, si corresponde del total de los datos*} Cerrar(B);

FIN. // *Recorrido con corte de control*

**Otras funciones definidas a tener en cuenta (entre págs. 15 a 26)**

**ConvertirBinario\_Texto**(Dato\_Resultado B: TipoArchivo; Dato\_resultado T: Texto): una acción

*Usar este algoritmo para convertir un archivo binario a uno de texto, teniendo en cuenta la transportabilidad de los archivos de texto estas acciones pueden ser necesarias*

POS: Crea el archivo de texto con el formato requerido y encolumnado correctamente.

**ConvertirTexto\_Binario**(Dato\_Resultado B: TipoArchivo; Dato\_resultado T: Texto): una acción

POS: Crea el archivo de binario a partir de uno de texto con formato conocido.

**AgregarRegistrosABinarioNuevo**(Dato\_Resultado B: TipoArchivo): una acción

POS: Crea el archivo de binario y le agrega reagistros, r.numero>0 es la condición del ciclo.

**AgregarRegistrosABinarioExistente**(Dato\_Resultado B: TipoArchivo): una acción

POS: Crea el archivo de binario y le agrega reagistros, r.numero>0 es la condición del ciclo.

**RecorrerBinarioV1**(Dato\_Resultado B: TipoArchivo): una acción

*Recorre un archivo completo con un ciclo de repetición exacto, dado que filesize del archivo indica la cantidad de registros, si se lee con un ciclo exacto una cantidad de veces igual a filesize y por cada iteracion se lee un registro se leerían todos los registros del archivo*

POS: Muestra el contenido completo del archivo por pantalla.

**RecorrerBinarioV2**(Dato\_Resultado B: TipoArchivo): una acción

*Recorre un archivo completo con un ciclo de repetición mientras haya datos, es decir mientras fin*

*de archivo sea falso, con una lectura anticipada.*

POS: Muestra el contenido completo del archivo por pantalla.

**RecorrerBinarioV3**(Dato\_Resultado B: TipoArchivo): una acción

*Recorre un archivo completo con una lectura como condición del ciclo mientras.*

POS: Muestra el contenido completo del archivo por pantalla.

**ApareoBinarioV2**(Dato\_Resultado A, B, C: TipoArchivo): una acción

*Requiere al menos dos estructuras, con al menos un campo en común y se requiere procesarlos simultaneamente, intercalándolos por la clave común.*

POS: Intercala dos archivos manteniendo el orden de las claves

**ApareoBinarioPorDosClaves**(Dato\_Resultado A, B, C: TipoArchivo): una acción

*Requiere al menos dos estructuras, con al menos un campo en común y se requiere procesarlos simultaneamnte, intercalándolos por la clave común.*

*Se suponen los archivos ordenados crecientes, en este caso por dos campos de ordenamiento*

POS: Intercala dos archivos manteniendo el orden de las claves

**BusquedaBinariaArchivoV2**(Dato\_Resultado B: TipoArchivo; Dato Clave:Tinfo; Dato\_resultado

Posic: Entero): una accion

*Utilizar este algoritmo si los datos en el archivo están ordenados por un campo clave, la clave se repite y se desea encontrar la primera ocurrencia de la misma en el archivo*

POS: Posic: Posición donde se encuentra la clave, ó (-1) si no esta

**GenerarBinarioOrdenadoPUP**(Dato\_Resultado A,B: TipoArchivoDatosPersonales): una accion *Utilizar este algoritmo sólo si la clave es numérica, con valor inicial y final conocido y están todos los valores intermedios. En este caso es posible generar un archivo con una posición que es única y se la puede conocer a priori. La posición que ocupará esa clave en el archivo ordenad es:*

*PUP = valor de la clave – Valor de la clave inicial. Es decir si decimos que los valores de la clave en los registros están entre 1 y 999, por ejemplo, la PUP = Registro.clave – 1, si los valores estuvieran comprendidos entre 30001 y 31000 la PUP = Registro.clave – 30000.*

POS: Genera un archivo ordenado con PUP, con el mismo registro que el sin orden

**GenerarIndicePUP**

(Dato\_Resultado A: TipoArchivoDatosPersonales; B:TipoArchivoEnteros): una accion

*Utilizar este algoritmo sólo si la clave es numérica, con valor inicial y final conocido y están todos los valores intermedios .No se tiene espacio para duplicar el archivo .*

POS: Genera un archivo ordenado con PUP, con la posición del registro en el sin orden