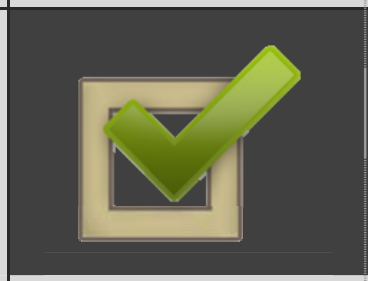
ALGORITMOS Y ESTRUCTURA DE DATOS



Operaciones sobre archivos

Antes de comenzar

Este documento resume las principales operaciones que son generalmente utilizadas para la manipulación de archivos. Cubre el uso de todas las funciones primitivas que provee el lenguaje C y explica también como desarrollar *templates* que faciliten el uso de dichas funciones.

Autor: Ing. Pablo Augusto Sznajdleder.

Revisores: Ing. Analía Mora, Martín Montenegro.

Introducción

Las funciones que analizaremos en este documento son:

- `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 número de byte al que está apuntando el indicador de posición del archivo.
- `fclose` - Cierra el archivo.

Todas estas funciones están declaradas en el archivo `stdio.h` por lo que para utilizarlas debemos agregar la siguiente línea a nuestros programas:

```
#include <stdio.h>
```

Además, basándonos en las anteriores desarrollaremos las siguientes funciones que nos permitirán operar con archivos de registros:

- `seek` – Mueve el indicador de posición de un archivo al inicio de un determinado registro.
- `fileSize` – Indica cuantos registros tiene un archivo.
- `filePos` – Retorna el número de registro que está siendo apuntado por el indicador de posición.
- `read` – Lee un registro del archivo.
- `write` – Graba un registro en el archivo.

Comenzando a operar con archivos

Grabar un archivo de caracteres

```
#include <iostream>
#include <stdio.h>

using namespace std;

int main()
{
   // abro el archivo; si no existe => lo creo vacio
   FILE* arch = fopen("DEMO.DAT","w+b");

   char c = 'A';
   fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'A' contenido en c

   c = 'B';
   fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'B' contenido en c

   c = 'C';
   fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'C' contenido en c

   // cierro el archivo
   fclose(arch);

   return 0;
}
```

La función `fwrite` recibe los siguientes parámetros:

- Un puntero al *buffer* (variable) que contiene el datos que se van a grabar en el archivo.
- El tamaño (en bytes) del tipo de dato de dicho *buffer*; lo obtenemos con la función: `sizeof`.
- La cantidad de unidades del tamaño indicado más arriba que queremos escribir; en nuestro caso: 1.
- Finalmente, el archivo en donde grabará el contenido almacenado en el *buffer*.

La función `fopen` recibe los siguientes parámetros:

- Una cadena indicando el nombre físico del archivo que se quiere abrir.
- Una cadena indicando la modalidad de apertura; `"w+b"` indica que queremos crear el archivo si es que aún no existe o bien, si existe, que queremos dejarlo vacío.

La función `fclose` cierra el archivo que recibe cómo parámetro.

Leer un archivo de caracteres

```
#include <iostream>
#include <stdio.h>

using namespace std;

int main()
{
   // abro el archivo para lectura
   FILE* arch = fopen("DEMO.DAT","r+b");
```

```
   char c;
   // leo el primer caracter grabado en el archivo
   fread(&c,sizeof(char),1,arch);
   // mientras no llegue el fin del archivo...
   while( !feof(arch) )
   {
      // muestro el caracter que lei
      cout << c << endl;
      // leo el siguiente caracter
      fread(&c,sizeof(char),1,arch);
   }
   fclose(arch);
   return 0;
}
```

La función `fread` recibe exactamente los mismos parámetros que `fwrite`. Respecto de la función `fopen`, en este caso la modalidad de apertura que utilizamos es: `"r+b"` para indicarle que el archivo ya existe y que no queremos vaciar su contenido; solo queremos leerlo.

Respecto de la función `feof` retorna `true` o `false` según si se llegó al final del archivo o no.

Archivos de registros

Las mismas funciones que analizamos en el apartado anterior nos permitirán operar con archivos de estructuras o archivos de registros. La única consideración que debemos tener en cuenta es que la estructura que vamos a grabar en el archivo no debe tener campos de tipos `string`. En su lugar debemos utilizar *arrays* de caracteres, al más puro estilo C.

Veamos un ejemplo:

```
struct Persona
{
   int dni;
   char nombre[25];
   double altura;
};
```

Grabar un archivo de registros

El siguiente programa lee por consola los datos de diferentes personas y los graba en un archivo de estructuras `Persona`.

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
   FILE* f = fopen("PERSONAS.DAT","w+b");
   int dni;
   string nom;
   double altura;
```

```
   // el usuario ingresa los datos de una persona
   cout << "Ingrese dni, nombre, altura: ";
   cin >> dni;
   cin >> nom;
   cin >> altura;
   while( dni>0 )
   {
      // armo una estructura para grabar en el archivo
      Persona p;
      p.dni = dni;
      strcpy(p.nombre,nom.c_str()); // la cadena hay que copiarla con strcpy
      p.altura = altura;
      fwrite(&p,sizeof(Persona),1,f); // grabo la estructura en el archivo
      cout << "Ingrese dni, nombre, altura: ";
      cin >> dni;
      cin >> nom;
      cin >> altura;
   }
   fclose(f);
   return 0;
}
```

En este programa utilizamos el método `c_str` que proveen los objetos `string` de C++ para obtener una cadena de tipo `char*` (de C) tal que podamos pasarla cómo parámetro a la función `strcpy` y así copiar el nombre que ingresó el usuario en la variable `nom` al campo `nombre` de la estructura `p`.

Leer un archivo de registros

A continuación veremos un programa que muestra por consola todos los registros del archivo `PERSONAS.DAT`.

```
#include <iostream>
#include <stdio.h>
#include <string.h>

using namespace std;

int main()
{
   FILE* f = fopen("PERSONAS.DAT","r+b");
   Persona p;
   // leo el primer registro
   fread(&p,sizeof(Persona),1,f);
   while( !feof(f) )
   {
      // muestro cada campo de la estructura
      cout << p.dni << ", " << p.nombre <<", " << p.altura << endl;

      // leo el siguiente registro
      fread(&p,sizeof(Persona),1,f);
   }
   fclose(f);
   return 0;
}
```

Acceso directo a los registros de un archivo

La función `fseek` que provee C/C++ permite mover el indicador de posición del archivo hacia un determinado byte. El problema surge cuando queremos que dicho indicador se desplace hacia el primer byte del registro ubicado en una determinada posición. En este caso la responsabilidad de calcular el número de byte que corresponde a dicha posición será nuestra. Lo podemos calcular de la siguiente manera:

```
void seek(FILE* arch, int recSize, int n)
{
   // SEEK_SET indica que la posicion n es absoluta respecto al inicio
   fseek(arch, n*recSize,SEEK_SET);
}
```

El parámetro `recSize` será el `sizeof` del tipo de dato de los registros que contiene el archivo. En el siguiente ejemplo accedemos directamente al tercer registro del archivo `PERSONAS.DAT` y mostramos su contenido.

```
#include <iostream>
#include <stdio.h>
#include <string.h>

using namespace std;

int main()
{
   FILE* f = fopen("PERSONAS.DAT","r+b");
   Persona p;

   // muevo el indicador de posicion hacia el 3er registro (contando desde 0)
   seek(f,sizeof(Persona), 2);

   // leo el registro apuntado por el indicador de posicion
   fread(&p,sizeof(Persona),1,f);

   // muestro cada campo de la estructura leida
   cout << p.dni << ", " << p.nombre <<", " << p.altura << endl;

   return 0;
}
```

Cantidad de registros de un archivo

En C/C++ no existe una función comparable a `fileSize` de Pascal que nos permita conocer cuántos registros tiene un archivo. Sin embargo podemos programarla nosotros mismos utilizando las funciones `fseek` y `ftell`.

```
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;
}
```

Probamos ahora las funciones `seek` y `fileSize`.

```
#include <iostream>
#include <stdio.h>
#include <string.h>

using namespace std;

int main()
{
   FILE* f = fopen("PERSONAS.DAT","r+b");
   Persona p;

   // cantidad de registros del archivo
   long cant = fileSize(f,sizeof(Persona));

   for(int i=cant-1; i>=0; i--)
   {
      // acceso directo al i-esimo registro del archivo
      seek(f,sizeof(Persona),i);

      // leo el registro apuntado por el indicador de posicion
      fread(&p,sizeof(Persona),1,f);

      // muestro el registro leido
      cout << p.dni <<", "<<p.nombre<<", "<< p.altura << endl;
   }

   fclose(f);

   return 0;
}
```

Identificar el registro que está siendo apuntando por el identificador de posición del archivo

Dado un archivo y el tamaño de sus registros podemos escribir una función que indique cual será el próximo registro será afectado luego de realizar la próxima lectura o escritura. A esta función la llamaremos: `filePos`.

```
long filePos(FILE* arch, int recSize)
{
   return ftell(arch)/recSize;
}
```

Templates

Como podemos ver, las funciones `fread` y `fwrite`, y las funciones `seek` y `fileSize` que desarrollamos más arriba realizan su tarea en función del `sizeof` del tipo de dato del valor que vamos a leer o a escribir en el archivo. Por esto, podemos parametrizar dicho tipo de dato mediante un *template* lo que nos permitirá simplificar dramáticamente el uso de todas estas funciones.

Template: `read`

```
template <typename T> T read(FILE* f)
{
   T buff;
   fread(&buff,sizeof(T),1,f);
   return buff;
}
```

Template: write

```
template <typename T> void write(FILE* f, T v)
{
   fwrite(&v,sizeof(T),1,f);
   return;
}
```

Template: seek

```
template <typename T> void seek(FILE* arch, int n)
{
   // SEEK_SET indica que la posicion n es absoluta respecto del inicio
   fseek(arch, n*sizeof(T),SEEK_SET);
}
```

Template: fileSize

```
template <typename T> long fileSize(FILE* f)
{
   // 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/sizeof(T);
}
```

Template: filePos

```
template <typename T> long filePos(FILE* arch)
{
   return ftell(arch)/sizeof(T);
}
```

Template: binarySearch

El algoritmo de la búsqueda binaria puede aplicarse perfectamente para emprender búsquedas sobre los registros de un archivo siempre y cuando estos se encuentren ordenados.

Recordemos que en cada iteración este algoritmo permite descartar el 50% de los datos; por esto, en el peor de los casos, buscar un valor dentro de un archivo puede insumir $\log_2(n)$ accesos a disco siendo n la cantidad de registros del archivo.

```
template <typename T, typename K>
int binarySearch(FILE* f, K v, int (*criterio)(T,K))
{
   // indice que apunta al primer registro
   int i = 0;

   // indice que apunta al ultimo registro
   int j = fileSize<T>(f)-1;

   // calculo el indice promedio y posiciono el indicador de posicion
   int k = (i+j)/2;
   seek<T>(f,k);

   // leo el registro que se ubica en el medio, entre i y j
   T r = leerArchivo<T>(f);

   while( i<=j && criterio(r,v)!=0 )
   {
      // si lo que encontre es mayor que lo que busco...
      if( criterio(r,v)>0 )
      {
         j = k-1;
      }
      else
      {
         // si lo que encontre es menor que lo que busco...
         if( criterio(r,v)<0 )
         {
            i=k+1;
         }
      }
      // vuelvo a calcular el indice promedio entre i y j
      k = (i+j)/2;
      // posiciono y leo el registro indicado por k
      seek<T>(f,k);
      // leo el registro que se ubica en la posicion k
      r = leerArchivo<T>(f);
   }
   // si no se cruzaron los indices => encontre lo que busco en la posicion k
   return i<=j?k:-1;
}
```

Ejemplos

Leer un archivo de registros usando el *template* `read`.

```
   f = fopen("PERSONAS.DAT","r+b");
   // leo el primer registro
   Persona p = read<Persona>(f);
   while( !feof(f) )
   {
      // muestro
      cout << p.dni<<", "<<p.nombre<<", "<<p.altura << endl;
      p = read<Persona>(f);
   }
   fclose(f);
```

Escribir registros en un archivo usando el *template* `write`.

```
f = fopen("PERSONAS.DAT","w+b");
// armo el registro
Persona p;
p.dni = 10;
strcpy(p.nombre,"Juan");
p.altura = 1.70;
// escribo el registro
write<Persona>(f,p);
fclose(f);
```

Acceso directo a los registros de un archivo usando los templates `fileSize`, `seek` **y** `read`.

```
f = fopen("PERSONAS.DAT","r+b");
// cantidad de registros del archivo
long cant = fileSize<Persona>(f);
for(int i=cant-1; i>=0; i--)
{
   // acceso directo al i-esimo registro del archivo
   seek<Persona>(f,i);
   Persona p = read<Persona>(f);
   cout << p.dni<<", "<<r.nombre<<", "<< r.altura << endl;
}
fclose(f);
```