Tema 2: Ficheros

Tecnología de la Programación

Índice

Índice

Introducción

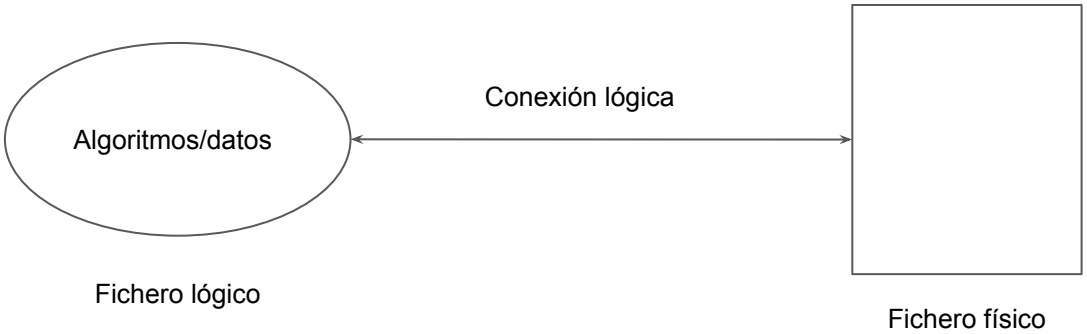
¿Por qué estudiar los ficheros?

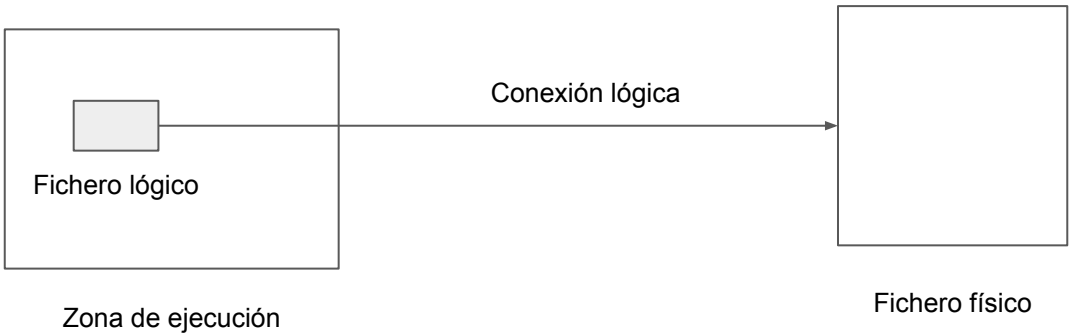
- Limitaciones de los Tipos de Datos (TDs) vistos hasta ahora:
  - Falta de información permanente
  - Limitación en cuanto al espacio disponible para los datos:
    - Tamaño de memoria es limitado
    - El tamaño (declarado) de la estructura de datos (fijo) también es limitado
- Principales características de los ficheros:
  - Información permanente
  - Número de datos indefinido (indeterminado) y limitado únicamente por el tamaño del dispositivo de almacenamiento
  - Desventaja: el tiempo de acceso a la información aumenta

Introducción

- Idea intuitiva
  - Los ficheros son “algo” que nos permite almacenar datos de forma permanente una vez acabada la ejecución del programa.
- Dos tipos:
  - Ficheros físicos (externos al programa): valores almacenados en algún dispositivo externo.
  - Ficheros lógicos (internos al programa): representación en el programa del fichero físico.

Introducción





No trabajamos directamente con el fichero físico, sino con el nombre lógico del fichero

Introdución

Para trabajar con el fichero físico hay que asociar el fichero físico al lógico estableciendo una conexión lógica

Símil: parecido a las variables y las posiciones de memoria donde se almacenan los valores de estas:

No trabajamos directamente con la memoria, sino con el nombre de la variable

No trabajamos directamente con el fichero físico, sino con el nombre lógico del fichero

Índice

Ficheros físicos

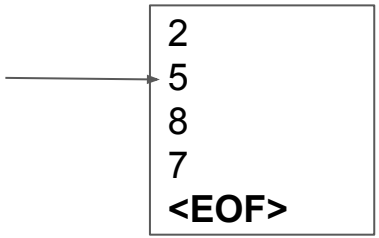
Características:

1. El tamaño de un fichero es indeterminado y limitado únicamente por la capacidad del dispositivo de almacenamiento
2. Todos los datos de nuestro ficheros serán del mismo tipo:
    - Lenguajes de programación admiten ficheros con datos de distinto tipo
    - Pero, algoritmos están muy pegados a la estructura del fichero y son susceptibles a múltiples errores

Ficheros físicos

Elementos

1. Apuntador



2. <EOF>

Ficheros físicos

Clasificación de los ficheros físicos:

1. Según el modo de acceso a los datos:
    - Ficheros secuenciales
    - Ficheros de acceso directo
2. Según el modo de almacenamiento de los datos:
    - Ficheros de texto (o con formato)
    - Ficheros binarios (o sin formato)

Índice

Procesamiento de ficheros

Pasos necesarios:

1. **Declarar** una variable de tipo fichero
2. **Apertura** de fichero - establecer conexión lógica
    - Comprobar que conexión se ha establecido
3. Operaciones
4. **Cierre** del fichero - cerrar conexión lógica

Índice

Ficheros en C++

Biblioteca en C++ que proporciona “funciones” y operadores para el manejo de ficheros

```
#include <fstream>
```

Ficheros en C++: Declaración

Declaración de variables de tipo fichero

a. Fichero de entrada (o de lectura): `ifstream <fichero_logico>;`
b. Fichero de salida (o de escritura): `ofstream <fichero_logico>;`
c. Fichero genérico: `fstream <fichero_logico>;`

Ficheros en C++: Apertura de ficheros

Apertura de ficheros:

```
<fichero_logico>.open(<nombre_fichero_fisico>,modo);
```

- `modo`: puede ser uno de los siguientes (o una combinación de):
  - `ios::in` - modo entrada (lectura)
  - `ios::out` - modo salida (escritura), es el modo por defecto si no se pone nada
  - `ios::app` - modo añadir al final
  - `ios::binary` - fichero binario (si no se pone, por defecto es de texto)
- Nota: se pueden combinar varias opciones usando |

Ficheros de C++: Apertura de ficheros

Hay más variantes a parte de las aquí vistas, pero no las veremos

Usando estas variantes:

- Abrir un fichero en modo escritura:
  - Si existe, lo vacía
  - Si no existe, lo crea
- Abrir en modo lectura un fichero:
  - Si existe, pone el apuntador al principio
  - Si no existe, produce un error de lectura

Ficheros de C++: Apertura de ficheros

Importante comprobar que la apertura se ha realizado correctamente:

```
if(<fichero_logico>){
   // operar con el fichero
}
else{
   cout << “Error al abrir el fichero” << endl;
}
```

Ficheros en C++: Cierre de ficheros

Cierre de ficheros:

```
<fichero_logico>.close();
```

- Semántica
  a. Rompe la conexión lógica
  b. Cierra el fichero para poder volver a utilizarlo

Ficheros en C++: Fin de fichero

Detección del fin de fichero:

```
<fichero_logico>.eof();
```

Esquema de recorrido de un fichero

```
ifstream f;
f.open(<nombre_fichero_fisico>);
if(f){
    // Leer un dato del fichero
    ...
    while(!f.eof()){
       // Operar con el dato
       // Leer siguiente dato del fichero
    }
}
```

Ficheros en C++: Operaciones sobre ficheros

1. Eliminación:

```
remove(<nombre_fichero_fisico>);
```

2. Renombrado:

```
rename(<nombre_fichero_anterior>,<nombre_fichero_nuevo>);
```

Ficheros en C++: Operaciones sobre ficheros de texto

1. Operador de lectura:

   `>>`

   `<fichero_logico> >> dato;`

   Nota: `>>` omite los espacios en blanco

2. Operador de escritura:

   `<<`

   `<fichero_logico> << dato;`

   Incluir explícitamente separadores entre los distintos datos para luego poder recuperarlos

Nota: `cin y cout` son ficheros predefinidos asociados con la entrada y la salida estándar del sistema operativo

Ficheros en C++: Operaciones sobre ficheros de texto

¿De qué tipo es el dato que se lee? ¿de qué tipo declaramos `dato`?

- Opción A:
  - `char dato;`
  - De este modo se puede mostrar por pantalla el contenido del fichero, pero:
    - Al almacenar los datos no lo hemos hecho carácter a carácter. Por ejemplo:
      - `f << dato;`
      - `dato` de tipo int

Ficheros en C++: Operaciones sobre ficheros de texto

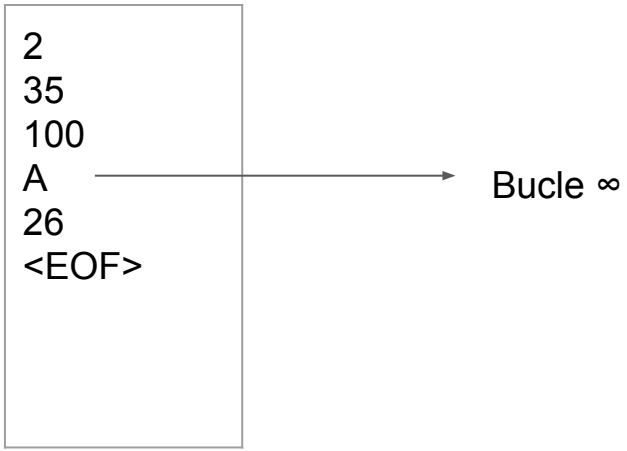
- Opción B:
  - Dar a `dato` el tipo original
  - ¿Qué hace `cin >> dato;`? Va al búffer original y toma caracteres que pueden formar parte del dato. Ejemplo: buffer 34.4a26
    - `dato int` → toma 34
    - `dato float` → toma 34.4
    - `dato` string → toma 34.4a26
  - De esta forma la información se puede recuperar tal y como se almacenó

Ficheros en C++: Operaciones sobre ficheros de texto

En la especificación del problema debería aparecer el tipo de los datos del fichero. De lo contrario trabajar con `char`

¿Qué pasa al ejecutar el siguiente programa con este fichero?

```
int dato;
…
if(f){
   f >> dato;
   while(!f.eof()){
      cout << dato <<endl;
      f >> dato;
   }
}
```



Ficheros en C++: Operaciones sobre ficheros binarios

1. Operador de lectura:

   `<fichero_logico>.read((char *) &dato, sizeof(dato));`

   `dato` es la variable donde se almacenará el dato leído

   `sizeof(_)` devuelve el número de bytes empleados para almacenar el dato

2. Operador de escritura:

   `<fichero_logico>.write((char *) &dato, sizeof(dato));`

   `dato` es la variable que se almacenará en el fichero

Ficheros en C++: lectura/escritura de estructuras

Suponed la siguiente estructura

```
struct tEmpleado{
    char dni[9];
    char nombre[20];
    int horasTrabajadas;
    int precioHora;
};
```

Ficheros en C++: lectura/escritura de estructuras

Si trabajamos con ficheros de texto, se almacena del siguiente modo:

```
12345678 Bea 12 24
23456789 Jose 23 12
...
```

Se lee del siguiente modo:

```
tEmpleado empl; ifstream canalE; canalE.open("Empleados.txt",ios::in);
if(canalE){
  canalE>>empl.dni;
  while(!canalE.eof()){
    canalE>>empl.nombre; canalE>>empl.horasTrabajadas; canalE>>empl.precioHora;
    cout<<"DNI: "<<empl.dni<< " Nombre: "<<empl.nombre<<" Horas trabajadas:"<<empl.horasTrabajadas<<
" Precio hora: "<<empl.precioHora<<endl;
    canalE>>empl.dni;
    }
 canalE.close();
}
```

Ficheros en C++: lectura/escritura de estructuras

Si trabajamos con ficheros binarios, se trabaja del siguiente modo:

```
tEmpleado empl;
ifstream canalE;
canalE.open("Empleados.dat",ios::in|ios::binary);
if(canalE){
  canalE.read((char *)&empl,sizeof(empl));
  while(!canalE.eof()){
     cout<<"DNI: "<<empl.dni<< " Nombre: "<<empl.nombre<<"Horas trabajadas:"<<empl.horasTrabajadas<<
     " Precio hora: "<<empl.precioHora<<endl;
    canalE.read((char *)&empl,sizeof(empl));
    }
 canalE.close();
}
```

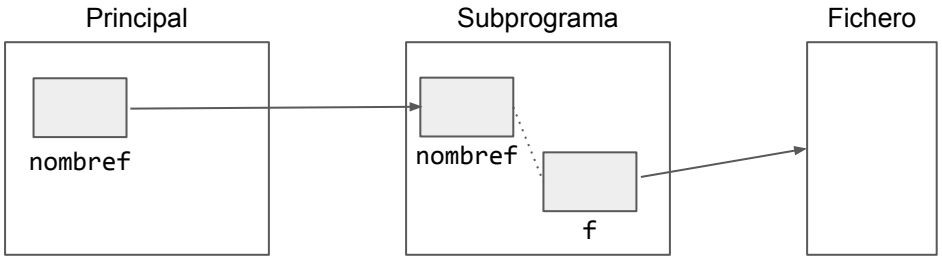
Ficheros en C++: ficheros como parámetros

Ficheros pueden pasarse como parámetros entre subalgoritmos

¿Dónde se establece la conexión lógica?

Opción A: Apertura en el subprograma:

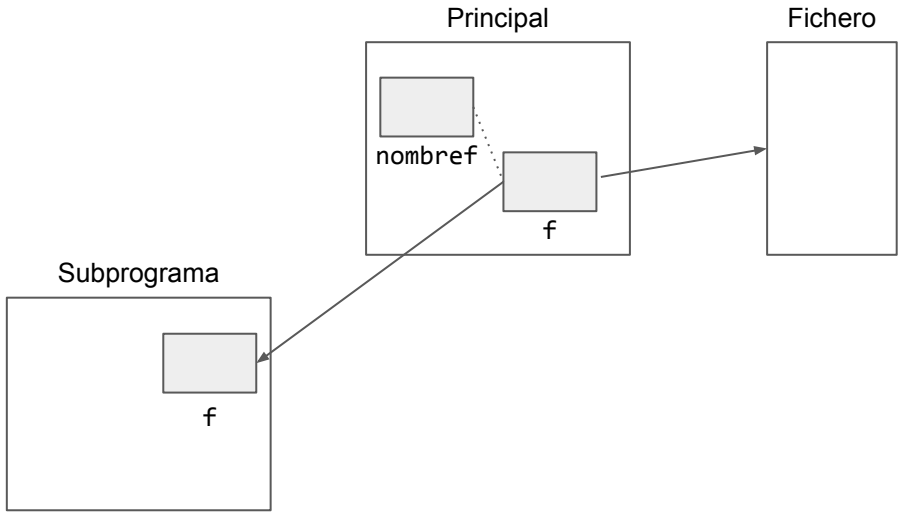
- Conexión en subprograma
- Subprograma necesita conocer el nombre del fichero físico ⇒ parámetro de entrada de tipo cadena
- La conexión se cierra en el subprograma



Ficheros en C++: ficheros como parámetros

Opción B: Apertura en el principal:

- Conexión en principal
- Subprograma recibe fichero abierto ⇒ No necesita nombre del fichero
- Subprograma recibe parámetro de tipo fichero de entrada/salida



Ficheros en C++: ficheros como parámetros

¿Qué opción usar?

Usaremos la opción B (pasamos fichero abierto) cuando:

- El subalgoritmo necesite recibir el fichero con el apuntador en una posición determinada (que puede no ser al principio del fichero)
- El subalgoritmo modifica el apuntador y es necesario que la posición de éste pase al principal

De lo contrario usaremos la opción A

Ficheros en C++: esquemas de operaciones

Operaciones de consulta:

1. Recorrido
2. Búsqueda

Operaciones de modificación:

3. Inserción
4. Eliminación
5. Modificación

Esquemas de operaciones: recorrido

Requiere siempre el recorrido completo del fichero

Ciclo: leer - comprobar - procesar

```
ifstream f; char nombref[30];
…
f.open(nombref,ios::in|ios::binary);
if(f){
   f.read((char*) & dato,sizeof(dato));
   while(!f.eof()){
      // tratamiento
      f.read((char*) & dato,sizeof(dato));
   }
   …
   f.close()
}
```

Esquemas de operaciones: búsqueda

En general, recorrido de parte del fichero

Caso 1: tratamiento dentro del ciclo

```
ifstream f; char nombref[30];
bool encontrado = false;
f.open(nombref,ios::in|ios::binary);
if(f){
   f.read((char*) & dato,sizeof(dato));
   while(!f.eof() && !encontrado){
      if(dato es elemento buscado){
         encontrado=true;
         // tratamiento
      }
      f.read((char*) & dato,sizeof(dato));
   }
   f.close()
}
```

Esquemas de operaciones: búsqueda

En general, recorrido de parte del fichero

Caso 2: tratamiento fuera del ciclo

```
ifstream f; char nombref[30];
bool encontrado = false;
f.open(nombref,ios::in|ios::binary);
if(f){
   f.read((char*) & dato,sizeof(dato));
   while(!f.eof() && !encontrado){
      if(dato es elemento buscado){
         encontrado=true;
      }else{
         f.read((char*) & dato,sizeof(dato));
      }
   }
   if(!f.eof() && encontrado){
      // Tratamiento
   }
   f.close()
}
```

Esquemas de operaciones: modificación

Operaciones de modificación:

- Suponen siempre modificar contenido de fichero existente
- Necesario utilizar un fichero auxiliar en el que ir escribiendo
- Pasos:
  1. Abrir fichero original en modo lectura
  2. Crear fichero auxiliar en modo escritura
  3. Manipular los ficheros de forma que al final el contenido del fichero auxiliar sea el del fichero resultado
  4. Cerrar ambos ficheros
  5. Borrar el fichero original: `remove(<nombre_fichero_fisico>);`
  6. Cambiar el nombre del auxiliar: `rename(<nombreFichAnt>,<nombreFichNuevo>);`

Esquemas de operaciones: inserción fichero ordenado

```
ifstream fo; ofstream faux; char nombref[30];
fo.open(nombref,ios::in|ios::binary);
faux.open("Auxiliar.DAT",ios::out|ios::binary);
if(fo && faux){
   fo.read((char *) &dato,sizeof(dato));
   while(!fo.eof() && dato < datoAinsertar){
      faux.write((char *) &dato,sizeof(dato));
      fo.read((char *) &dato,sizeof(dato));
   }
   faux.write((char *) &datoAinsertar,sizeof(dato));
   while(!fo.eof()){
      faux.write((char *) &dato,sizeof(dato));
      fo.read((char *) &dato,sizeof(dato));
   }
   fo.close(); faux.close();
   remove(nombref); rename("Auxiliar.DAT",nombref);
}
```

Esquemas de operaciones: eliminación

```
ifstream fo; ofstream faux; char nombref[30];
fo.open(nombref,ios::in|ios::binary);
faux.open(“Auxiliar.DAT”,ios::out|ios::binary);
if(fo && faux){
   fo.read((char *) &dato,sizeof(dato));
   while(!fo.eof()){
      if(dato no es datoAeliminar){
         faux.write((char *) &dato,sizeof(dato));
      }
      fo.read((char *) &dato,sizeof(dato));
   }
   fo.close(); faux.close();
   remove(nombref); rename(“Auxiliar.DAT”,nombref);
}
```

Esquemas de operaciones: modificación

```
ifstream fo; ofstream faux; char nombref[30];
fo.open(nombref,ios::in|ios::binary);
faux.open("Auxiliar.DAT",ios::out|ios::binary);
if(fo && faux){
   fo.read((char *) &dato,sizeof(dato));
   while(!fo.eof()){
      if(dato es datoAmodificar){
          // Modificar dato
      }
      faux.write((char *) &dato,sizeof(dato));
      fo.read((char *) &dato,sizeof(dato));
   }
   fo.close(); faux.close();
   remove(nombref); rename("Auxiliar.DAT",nombref);
}
```