INTRODUCCIÓN A LA PROGRAMACIÓN
PRÁCTICA 6

Objetivos

- ✓ Estudiar la utilización de las sentencias de composición iterativa en programas C++ resolviendo problemas sencillos de recorrido que manejen secuencias del primer modelo de acceso secuencial.

1 Tipo máquina secuencial en C++

En el lenguaje C++ no se encuentra predefinido el tipo de dato máquina secuencial, tal como lo hemos estudiado en clase de teoría. Para ilustrar las operaciones del tipo de datos Secuencia, tanto del primer como del segundo modelo de acceso secuencial, vamos a utilizar los contenedores de la librería estándar de C++. Antes de introducirlos, vamos a estudiar las operaciones básicas del tipo de datos archivo en C++. El archivo es una estructura con información persistente cuyo tamaño puede variar durante la ejecución del programa.

Existen tres tipos de datos básicos para declarar archivos de texto de acceso secuencial en C++: `ifstream` (entrada) `ofstream` (salida) `fstream` (entrada y salida). Para utilizar estos tipos hay que incluir el archivo de cabecera `<fstream>`. Por ejemplo, supongamos la declaración

```
ofstream f;
```

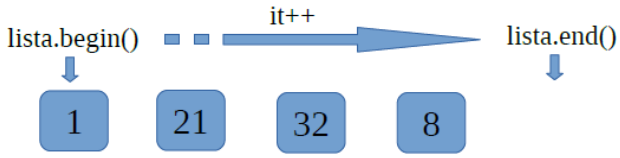
La variable f representa un archivo lógico de salida (modo escritura), pero todavía no está enlazada a un archivo físico reconocido por el sistema operativo. Para relacionar el archivo lógico con el archivo físico se emplea la operación `open`(). Por ejemplo:

```
f.open("miArchivoconDatos.txt");
```

Por defecto los archivos de entrada (ifstream) se abren sólo para lectura poniendo la ventana del archivo en el primer elemento del archivo. Además el archivo debe existir, si no se genera un error. Por defecto, los archivos de salida (ofstream) se abren sólo para escritura creando el archivo nuevo. Si el archivo ya existía es borrado. Los archivos de salida (fstream) se abren para lectura y escritura. Para conocer si en la apertura de un archivo se ha producido un error se emplea el operador ! sobre la variable de archivo.

Las operaciones de lectura y escritura sobre el archivo son similares a las que se realizan con las instrucciones `cin` y `cout`. Cuando ya no se van a realizar operaciones sobre un archivo hay que cerrarlo. Esta operación libera las estructuras que se han creado para abrirlo y escribe la información que hubiera quedado en el buffer intermedio. Para este fin se emplea la instrucción `close()`.

Las listas en C++, como casi todos los contenedores, admiten ser recorridas mediante iteradores externos. Estos iteradores funcionan “normalmente” (cuando se recorre en sentido ascendente) según el primer modelo de acceso secuencial. Supongamos una lista de números enteros llamada “lista” como la que muestra la figura:



En C++ declaramos una lista de enteros S de la siguiente forma:

```
list<int> S;
```

A continuación, declaramos el iterador EA para recorrer la lista:

```
list<int>::iterator EA;
```

La lista tiene dos operaciones (métodos) que devuelven valores que "apuntan", por así decirlo, al principio y al final. Esas operación son los siguientes:

- `begin`(): Esta operación devuelve un iterador que apunta al primer elemento de la lista directamente, sin necesidad de un avance. Si la lista está vacía, esta instrucción devuelve el valor de la siguiente operación:
- `end`(): Esta operación devuelve un iterador que apunta a una posición que se encuentra después del último elemento significativo de la lista (igual que sucede con la marca de fin del primer modelo de acceso secuencial).

De este modo, asignar a un iterador "it" el valor begin() equivale a posicionarlo en una posición equivalente al elemento actual tras la operación de Comenzar del primer modelo de acceso secuencial. Para acceder al elemento actual apuntado por un iterador "EA" se utiliza el operador de indirección: *EA. Para avanzar el iterador se utiliza el operador habitual de C de incremento: ++. Esa operación equivale al Avanzar. La siguiente tabla resume las equivalencias entre C++ y notación SP:

Primer modelo de acceso secuencial	Iteradores C++
`Comenzar(S)`	`EA = S.begin()`
`EA(S)`	`*EA`
`Avanzar(S)`	`EA++`
`MarcaFin`	`S.end()`

Con este tipo de iteradores se puede hacer un recorrido inverso (del último elemento al primero) utilizando prácticamente los mismos elementos léxicos. Podemos asignarle a un iterador el valor de "end()", lo cual lo coloca después del último elemento significativo (o antes del primero, ya que vamos a hacer un recorrido inverso, o sea, lo mismo que hace el Iniciar del segundo modelo de acceso secuencial. Para avanzar al siguiente elemento del recorrido (que es el anterior de la lista) podemos hacer retroceder el iterador con el operador habitual de decremento --, lo cual equivale al Avanzar. El final del recorrido lo detectamos cuando llegamos al "begin()", o sea, al primer elemento de la lista que es el último del recorrido inverso. De este modo, con estos iteradores podemos hacer un recorrido inverso según el segundo modelo de acceso secuencial:

Segundo modelo de acceso secuencial	Iteradores C++
`Iniciar (S)`	`EA = S.end()`
`EA(S)`	`*EA`
`Avanzar(S)`	`EA--`
`EsUltimo(S)/EsVacia(S)`	`EA == S.begin()`

En ambos modelos, las operaciones de creación son las siguientes:

Crear (S);	`S.clear ();`
Registrar (S, e);	`S.push_back (e);`
Marcar (S); (en el primer modelo)	No es necesario. La marca de fin siempre está presente en una lista tras el último elemento de ésta.

Ejemplo 1: El siguiente programa carga el contenido de un archivo de texto cuyo nombre es `entrada1.txt` en una lista de caracteres `S`. A continuación copia ésta en otra lista de caracteres `R`, sustituyendo cada carácter en minúscula por su correspondiente mayúscula y graba el contenido final de `R`, en un archivo de texto cuyo nombre es `salida1.txt`.

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
using namespace std;

const char MARCA_FIN_LINEA = '\n';

void Guardar_Fichero (list<char> S, string nombre) {
     ofstream f;
     list<char>::iterator EA;

     f.open (nombre);
     if (!f) {
           cout << "Error abriendo el fichero resultado" << endl;
     }
     else
     {
           EA = S.begin();
           while (EA != S.end()) {
                 if (*EA == MARCA_FIN_LINEA) {
                       f << endl;
                 } else {
                       f << *EA;
                 }
                 EA++;
           }
           f.close();
     }
}

void Cargar_Fichero (list<char> &S, string nombre) {
     ifstream f;
     string cadena;
     int i;

     f.open (nombre);
     if (!f) {
           cout << "Error abriendo el fichero de datos" << endl;
     }
     else {
           S.clear(); // Borra el contenido previo de la lista
```

```
            while (getline(f, cadena)) {
                  for (i=0; i<cadena.length(); i++) {
                        S.push_back (cadena[i]);
                  }
                  S.push_back (MARCA_FIN_LINEA);
            }
      }
      f.close();
}

int main()
{
/*  Distancia entre los juegos de caracteres mayúsculas y minúsculas: */
      const int Distancia = int('a') - int('A');

      list<char> S, R;
      list<char>::iterator EA;
      char c_conv;

      Cargar_Fichero (S, "entrada1.txt");

/* Primer esquema de recorrido del primer modelo de acceso secuencial*/
      EA = S.begin();
      while (EA != S.end()) {
            if ((*EA >= 'a') && (*EA <= 'z')) {
                  c_conv = char(int(*EA)-Distancia);
            }
            else {
                  c_conv = *EA;
            }
            R.push_back (c_conv); /* Registrar(R, c_conv) */
            EA++;
      }
      Guardar_Fichero (R, "salida1.txt");
      return 0;
}
```

Ejemplo 2: El siguiente programa carga el contenido de un archivo de texto cuyo nombre es `entrada2.txt` en una secuencia de enteros S. A continuación escribe por la salida estándar el contenido de esta secuencia al mismo tiempo que va registrando en otra secuencia R el cuadrado de cada número que aparece en S. Finalmente graba el contenido de R en un archivo de texto cuyo nombre es `salida2.txt`:

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>

using namespace std;
```

```
void Guardar_Fichero (list<int> S, string nombre) {
     /* Genera datos y escribe en fichero y muestra en pantalla */
     ofstream f;
     list<int>::iterator EA;

     f.open (nombre);
     if (!f) {
          cout << "Error abriendo el fichero resultado" << endl;
     }
     else
     {
          EA = S.begin();
          while (EA != S.end()) {
               f << *EA << " ";
               EA++;
          }
          f << endl;
          f.close();
     }
}

void Cargar_Fichero (list<int> &S, string nombre) {

     ifstream f;
     int dato;

     f.open (nombre);
     if (!f) {
          cout << "Error abriendo el fichero de datos" << endl;
     }
     else {
          S.clear ();
          while (f >> dato) { // mientras la lectura sea exitosa
               S.push_back (dato); // Registrar (S, dato)
          }
          f.close();
     }
}
```

```
int main()
{
     list<int> S, R;
     list<int>::iterator EA_S;

     Cargar_Fichero (S, "entrada2.txt");

   /* Primer esquema de recorrido del primer modelo de acceso secuencial */
     EA_S = S.begin();
     while (EA_S != S.end()) {
          cout << *EA_S << ", ";
          R.push_back ((*EA_S)*(*EA_S)); // Registrar(R,    (*EA_S)*(*EA_S))
          EA_S++;
     }
     Guardar_Fichero (R, "salida2.txt");

     return 0;
}
```

Ejemplo 3: El siguiente programa carga el contenido de un archivo de texto, que contiene las notas de un curso, cuyo nombre es `entrada3.txt` en una secuencia de reales `S` y muestra por pantalla el número de notas que superan el cinco (suponemos definidas las acciones `Cargar_Fichero` y `Guardar_Fichero` semejantes a las del ejemplo anterior, pero para el tipo `list<float>`).

```
int main()
{
     list<float> S;
     list<float>::iterator EA_S;
     int cont;

     Cargar_Fichero (S, "salida2.txt");

   /* Primer esquema de recorrido del primer modelo de acceso secuencial */
     EA_S = S.begin();
     cont = 0;
     while (EA_S != S.end()) {
          if (*EA_S >= 5.0) {
               cont++;
          }
          EA_S++;
     }
     cout << "El número de aprobados es: " << cont;
     return 0;
}
```

Problema 1: A continuación se muestran un par de versiones distintas de un mismo programa C++, que utiliza una lista de caracteres:

```
int main()
{
   list<char> S;
   list<char>::iterator EA_S;
   int cont;

Cargar_Fichero(S,"datosP5_2.txt");

   /* Primer esquema de recorrido del
primer modelo de acceso secuencial*/
   EA_S = S.begin();
   cont = 0;
   while (EA_S != S.end()) {
     cout << *EA_S << ", ";
     cont = cont + 1;
     EA_S++;
   }
   cout << endl << "El número de
datos recogidos es: " << cont <<
endl;
   return 0;
}
```

```
int main()
{
   list<char> S;
   list<char>::iterator EA_S;
   int cont;

Cargar_Fichero(S,"datosP5_2.txt");

   /* Primer esquema de recorrido del
primer modelo de acceso secuencial*/
   EA_S = S.begin();
   cont = 0;
   while (EA_S != S.end()) {
     EA_S++;
     cout << *EA_S << ", ";
     cont = cont + 1;
   }
   cout << endl << "El número de
datos recogidos es: " << cont <<
endl;
   return 0;
}
```

- ✓ ¿Qué esquema algorítmico se está tratando de utilizar?
- ✓ ¿Cuál de los dos es el correcto? ¿Por qué?
- ✓ Antes de ejecutarlos intente predecir cuál sería el resultado que se produciría al compilar y ejecutar cada uno de ellos. Realice estos procesos con ambos y compare los resultados con sus hipótesis.

Problema 2 :

Dada una secuencia de enteros *S*, escriba un algoritmo que cree una nueva secuencia *T* formada por la secuencia de sumas parciales de *S* (S_1, $S_1 + S_2$, $S_1 + S_2 + S_3$, ..). Por ejemplo para la secuencia de entrada: 1, 4, 5, 3, 5, 6. La secuencia de salida sería: 1, 5, 10, 13, 18, 24.

Problema 3 :

Escriba un algoritmo que obtenga la intersección de dos conjuntos de enteros representados como secuencias ordenadas crecientemente.

Ejemplo:

S1 = [-44, -21, -1, 3, 5, 7, 18, 24, 30, 33, 54, 66, 101]
S2 = [-32, -28, -21, 0, 3, 4, 6, 7, 14, 20, 24, 31, 33, 51, 66, 104, 200]
S1 ∩ S2 = R = [-21, 3, 7, 24, 33, 66]

Trabajo personal del alumno

Problema 1

Escriba una función que simule la evolución del marcador en un partido de tenis. Los datos de entrada son el marcador actual y qué jugador ha ganado el punto en juego. Como resultado se mostrará por pantalla el nuevo marcador.

Utilice esta función para escribir un algoritmo que simule la evolución de un partido de tenis, supuesto que la entrada es una secuencia *S* de caracteres: '1' y '2', que indica qué jugador ha ganado cada punto. Para cada elemento de la secuencia se indicará el nuevo marcador del partido. Se supondrá que el partido lo gana el jugador que consigue seis juegos. Por ejemplo, si *S* = "112211221", la salida sería: "quince–nada", "treinta–nada", "treinta–quince", "iguales a treinta", "cuarenta–treinta", "juego para el jugador 1", "nada–quince", "nada–treinta", "quince–treinta".

Problema 2

La Caja de ahorros del Mediterráneo gestiona la atención a los clientes mediante un sistema automático. Dispone de un número fijo de ventanillas de atención (N, constante entera positiva conocida). Cada cliente, para ser atendido, debe pulsar un botón de turno, si hay alguna ventanilla libre, una pantalla digital mostrará en mensaje ("Pase"), en caso contrario mostrará el mensaje ("Espere por favor") y el usuario se pondrá en una cola, hasta que pueda ser atendido, en el momento que un empleado quede libre, la pantalla mostrará el mensaje ("Pase") si la cola no está vacía.

Escriba un algoritmo que muestre en pantalla, en el momento oportuno, cada uno de estos dos mensajes. La entrada del algoritmo será una secuencia de valores enteros: "1", "2" , que sigue el primer modelo de acceso secuencial. El "1" indica una petición por parte de un cliente, el "2" que una ventanilla ha quedado libre. Inicialmente todas las ventanillas estarán libres.