Sesión 7. Cadenas de caracteres, TAD y VoV

Competencias previas

- Crear, importar, exportar, compilar, depurar y ejecutar un proyecto en Eclipse.
- Escribir pequeños programas en C++ con variables de tipos simples y vectores, operadores e instrucciones de lectura, escritura y asignación.
- Poder incluir en programas funciones de las librerías predefinidas de C++.
- Poder incluir en programas librerías implementadas por el programador.

Objetivos

- Codificar en C++ algoritmos sencillos que utilicen cadenas de caracteres.
- Utilización de librerías definidas por el programador para construir TAD.
- Utilización de Vector de ocupación Variable (VoV).

Cadenas de caracteres (string)

Si queremos almacenar y manipular palabras, frases o textos en general, debemos utilizar cadenas de caracteres. Internamente, una cadena de caracteres es simplemente un vector cuyas celdas almacenan un carácter (`char`). Además, detrás de la última posición ocupada se almacena, de forma transparente para el programador (es decir, sin que el programador tenga que preocuparse), el carácter ASCII nulo ('\0'), para indicar el final de la cadena.

Las cadenas de caracteres son vectores y se pueden manipular como tales a todos los efectos. Sin embargo, C++ nos permite trabajar con ellas en muchos aspectos de forma similar a como lo hacemos con variables de tipo simple.

Utilizando la librería **<string>** es posible definir y manipular variables de tipo *string* que permiten almacenar textos. Aunque estas variables pueden ser usadas como un vector, accediendo a cada una de sus celdas individualmente, lo más frecuente es usarlas como si se tratara de variables de un tipo simple (int, float, etc.), es decir, como si la información que contienen fuera un bloque indivisible. En ese sentido se les puede asignar un valor, leer, escribir y comparar como si se tratara de valores simples. También se puede utilizar el operador `+` para concatenar cadenas. Por ejemplo, son posibles las siguientes operaciones:

```
#include <string>
string cad, s;
cad="gato";
cin >> s;
if (s <= cad) ...
cout << cad + s;
```

En el siguiente esquema se representa el contenido de la variable `cad` *de tipo* `string`:

'g'	'a'	't'	'o'	'\0'	???	???	???	???	.....
cad[0]	*cad[1]*	*cad[2]*	*cad[3]*	*cad[4]*	*cad[5]*	*cad[6]*	*cad[7]*	*cad[8]*	.....

A continuación se muestran algunas de las operaciones ofrecidas por la librería `<string>`:

- **size** devuelve la longitud de la cadena
- **clear** elimina el contenido de la cadena
- **compare** compara dos cadenas
- **insert** incluye una cadena dentro de otra
- **find** busca una cadena dentro de otra

Podéis encontrar más información en la página http://www.cplusplus.com/reference/string/string

Aunque es posible leer una variable de tipo **string** utilizando **cin**, la operación no se llevará a cabo correctamente si se teclea una cadena que contiene espacios en blanco, ya que la lectura queda interrumpida

al llegar a un espacio. Por ejemplo, al teclear "Hola mundo", la operación **`cin >> s`** solo almacenaría en **`s`** la cadena "Hola". Para evitar este problema, basta con utilizar una variante de cin:

```
getline(cin, s)
```

Revisa el proyecto "***string.tar.gz***", donde podrás comprobar cómo algunas operaciones (comparación y concatenación) se realizan de forma muy cómoda. Sin embargo, las operaciones de lectura y las que nos permiten conocer el tamaño de la cadena tienen una sintaxis diferente a la que habéis visto hasta ahora en los programas C++.

Actividad 1: Operaciones básicas con "string"

- Descarga del aula virtual e importa en Eclipse el proyecto **string.tar.gz** que contiene un sencillo programa que permite probar las funciones descritas anteriormente.

- Modifica la concatenación de las cadenas para que los apellidos no queden completamente pegados al nombre, sino que haya un espacio en blanco entre ellas.

Tipos abstractos de datos (TAD)

La definición de Tipos Abstractos de Datos (TAD) nos permite "ampliar" el lenguaje añadiendo nuevos tipos de datos, acompañados de una serie de operaciones para trabajar con ellos. Para implementar un TAD, utilizaremos una librería constituida por dos ficheros: el fichero "cabecera" (extensión .h), incluirá la definición del nuevo tipo de dato así como las cabeceras de los módulos asociados a cada una de las operaciones del TAD. La implementación de estas operaciones se incluirá en el fichero con extensión .cpp.

Vectores de ocupación Variable (VoV)

Los Vectores de ocupación Variable (VoV) son estructuras de datos implementadas en base a un registro con dos campos:

- Un *vector* del tamaño máximo esperado
- Un contador con el número de elementos útiles en el vector (*ocupadas*)

```
const int MAX = 100;           // Número máximo de elementos
typedef TipoBase TVector[MAX]; // TipoBase: tipo de los elementos del VoV
struct VoV {
       TVector vector;
       int ocupadas;
};
```

En un VoV…

- Todas las casillas ocupadas estarán agrupadas al principio.
- Las operaciones que inserten y eliminen datos tendrán que ocuparse de actualizar el contador.
- Para hacer un procesamiento secuencial de los elementos útiles de un VoV usaremos un índice que recorrerá el *vector* desde la posición 0 hasta la posición indicada por *ocupadas.*

Si definimos una variable del tipo VoV, podremos hacer referencia a las distintas partes de dicha variable:

```
VoV v;           // Definición de una variable de tipo VoV

v                // Referencia a la estructura completa
v.ocupadas       // Referencia al campo ocupadas
v.vector         // Referencia al vector del VoV
v.vector[i]      // Referencia a la posición i-ésima del vector del VoV
```

Actividad 2: Tareas pendientes (Diciembre 2010)

Descargar del campus virtual el proyecto "***TareasPendientes.tar.gz***".

Debéis **completar la implementación de TAD Tareas** (*tadTareas.cpp*) conforme al siguiente enunciado:

Ante la cantidad de tareas y actividades que tenéis que realizar hasta el final del semestre, y para poder aprovechar mejor vuestro tiempo, habéis decidido hacer un pequeño programa que os ayude a gestionar las tareas pendientes. Tras analizar el problema, se ve la necesidad de almacenar, para cada tarea pendiente, un identificador (un entero), una descripción (una cadena de caracteres), la asignatura relacionada (un entero) y la duración prevista de la tarea en horas (un real).

La especificación del TAD **TareaPendiente** es la siguiente:

TAD

TareaPendiente es nuevo, obtenerIdentificador, obtenerDescripcion, obtenerAsignatura, obtenerDuracion.

Operaciones

nuevo (id: entero, descripción: cadena, asignatura: entero, duración: real)
- retorna: TareaPendiente
- efecto: Crea una tarea pendiente

obtenerIdentificador (tP: TareaPendiente)
- retorna: entero
- efecto: Devuelve el id de la tarea pendiente tP

obtenerDescripcion (tP: TareaPendiente)
- retorna: cadena
- efecto: Devuelve la descripción de la tarea pendiente tP

obtenerAsignatura (tP: TareaPendiente)
- retorna: entero
- efecto: Devuelve la asignatura relaciona con la tarea pendiente tP

obtenerDuracion (tP: TareaPendiente)
- retorna: real
- efecto: Devuelve la duración de la tarea pendiente tP

Podemos suponer que nunca vamos a tener más de 25 tareas pendientes.

La especificación del TAD Tareas, para gestionar las tareas pendientes, es la siguiente:

TAD

Tareas es iniciar, insertar, total, buscar

Operaciones

iniciar
- retorna: Tareas
- efecto: Inicia la estructura.

insertar (t: Tareas, tP: TareaPendiente)
- modifica: Tareas
- efecto: Inserta una nueva tarea pendiente tP. La nueva tarea estará **ordenada ascendentemente** por su duración.

total (t: Tareas)
- retorna: real
- efecto: Devuelve la duración total de todas las tareas pendientes.

buscar (t: Tareas, h: real)
retorna entero
efecto Devuelve el identificador de la tarea de mayor duración que se puede resolver en, como mucho, h horas. Podemos suponer que siempre existirá una tarea que cumpla el requisito.

Ejemplo de buscar: supongamos que tenemos 3 tareas con duraciones previstas de 1, 3 y 5 horas, respectivamente. Si usamos buscar con un valor 4, nos debería devolver el identificador de la tarea de duración 3: la tarea de mayor duración que se puede resolver en, como mucho, 4 horas. Si existiera una tarea de duración 4, nos devolvería el identificador de esa tarea.

1. Escribir la definición de tipos necesaria para almacenar la información de la estructura (TAD TareaPendiente y TAD Tareas).
2. Implementar de forma eficiente todas las operaciones del TAD.
3. Indicar cuál es el tamaño del problema y la complejidad de cada operación.

   Se valorará que la operación total tenga coste constante.

Nota. En el tadTareas se han añadido las dos operaciones siguientes, de utilidad para implementar las pruebas del TAD: *int cuantos (Tareas t)* y *void mostrar(Tareas t)*

Actividad 3: Recetas (Junio 2011)

Un vecino ha decidido mejorar sus dotes culinarias este verano y se ha comprado la Gran Enciclopedia de la Cocina. Te ha pedido ayuda para hacer un programa que le permita gestionar las recetas que va probando.

Quiere saber qué recetas ha hecho y cómo le han salido. Tras analizar el problema, determinamos que necesitamos guardar información sobre, como mucho, 200 recetas (aunque en la enciclopedia hay muchas más, no le va a dar tiempo a hacer más este verano). De cada receta necesitaremos almacenar el número de página donde se explica, las veces que la ha hecho y la puntuación más alta que le han asignado los comensales al probarla.

La especificación del TAD Receta es la siguiente:

TAD

Receta es nueva, actualiza, obtenerNumPagina, obtenerNumVeces, obtenerPuntuacion

Operaciones

nueva (numPágina: entero, puntuacion: entero)
retorna Receta
efecto Crea una nueva receta. El número de veces será 1.

actualiza (r: Receta, punt: entero)
modifica Receta
efecto Modifica la receta r incrementando el número de veces que se ha hecho y actualizando la puntuación con el valor *punt* si es mayor que la puntuación actual

obtenerNumPagina (r : Receta)
retorna entero
efecto Devuelve el número de página de la receta.

obtenerNumVeces (r : Receta)
retorna entero
efecto Devuelve el nº de veces que se ha hecho la receta

obtenerPuntuación (r : Receta)
retorna real
efecto Devuelve la puntuación máxima asignada a la receta

La especificación inicial del TAD Recetario, que debes implementar, es la siguiente:

TAD

Recetario es crear, insertar, masVeces, cuantas, borrar

Operaciones

crear retorna Recetario
efecto Inicia la estructura.

insertar (rs: Recetario, P: entero, Calif: real)
modifica Recetario
efecto Inserta la información de la receta que empieza en la página P. Si ya estaba esa receta en la estructura, se aumentará el número de veces que se ha hecho y se actualizará la puntuación con el valor de Calif si es mayor que la puntuación que había almacenada. Si no había información sobre esa receta, se incluirá en la estructura.

masVeces (rs: Recetario)
retorna entero
efecto Devuelve la página donde se encuentra la receta que más veces se ha hecho. Si hubiera más de una con el mismo número, devuelve una cualquiera.

cuantas (rs: Recetario, v: real)
retorna entero
efecto Devuelve el número de recetas que han obtenido una valoración mayor que v.

borrar (rs: Recetario, pag: entero)
modifica Recetas
efecto Elimina la información de la receta de la página pag. Si no existiera esa receta, no hace nada.

Se pide:

1. Escribir la definición de tipos necesaria para almacenar la información de la estructura.
2. Implementar de forma eficiente todas las operaciones del TAD.