Facultad de Informática – Universidad Complutense de Madrid

Fundamentos de la Programación

Curso 2017–2018, Grupos C, E y G

Examen de Junio

Duración del examen: 3 horas

Se pide construir un programa modular en C++ que permita gestionar departamentos de empleados. La solución constará de un programa principal (`main.cpp`) y tres módulos: *ListaTareas*, *ListaContratos* y *Departamento*.

Módulo *ListaTareas* (3 puntos) Máx. 15 tareas

Declara un tipo `tPrioridades` como tabla donde para cada año y tipo de tarea se guarda un número de prioridad entre 1 y 5. Añade un tipo `tTarea` con tres campos: código (entero), descripción y tiempo (horas al mes). Define también un tipo `tListaTareas` para listas con array estático de `tTarea` (máx. 15). Estas listas no se mantendrán ordenadas.

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargarPrioridades()`. Carga las prioridades de 5 años para 7 tipos de tareas desde un archivo `prioridades.txt` (ver ejemplo al final del enunciado).
- ✓ `cargarTareas()`. Carga una lista de tareas. Entre los datos que se encuentran en el archivo `contratos.txt` (ver ejemplo al final del enunciado) están las distintas listas de tareas que corresponden a cada empleado contratado. En estas listas aparece el número de tareas y cada tarea consta de dos líneas: código y descripción. Por cada una de estas tareas se obtiene el tiempo asignable.
- ✓ `calcularTiempo()`. Calcula el tiempo de una tarea según la media de prioridades de todos los años * 20 horas al mes. Redondea la media al entero más grande no mayor de x con la función floor(x).
- ✓ `mostrarTareas()`. Muestra la información de una lista de tareas según el formato dado en el ejemplo. Cada tarea tiene su código, descripción (con un ancho de 40) y tiempo asignado.

Módulo *ListaContratos* (3 puntos) Máx. 100 contratos

Declara un tipo `tContrato` con cinco campos: nombre, nif (8 dígitos y una letra en una cadena), tareas (una lista del tipo `tListaTareas`) y sueldo (double). Añade un tipo `tListaContratos` para listas de `tContrato` (máx. 100). Implementa esta lista con un **array estático de punteros a variables dinámicas** y mantenla **ordenada** por nif de menor a mayor.

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargarContratos()`. Carga una lista de contratos del archivo **`contratos.txt`**. El fichero comienza con el número de contratos que contiene, y a continuación aparece la información de cada uno de ellos. El archivo no está ordenado por lo que al cargar hay que insertar de forma ordenada.
- ✓ `insertarContrato()`. Inserta un nuevo contrato en una lista ordenada de contratos. Debe mantenerse el orden y no puede usarse ningún algoritmo de ordenación.
- ✓ `mostrarContratos()`. Muestra una lista de contratos según el formato dado en el ejemplo. En este formato, y para cada contrato, el nombre se escribe con ancho máx. de 30, el nif de 10 y el número de tareas aparece entre paréntesis.
- ✓ `seleccionarContrato()`. Permite seleccionar un contrato de la lista de contratos. Muestra la lista y el usuario puede elegir por teclado un contrato según el número de orden. Devuelve la posición en la lista (orden - 1).
- ✓ `liberar()`. Libera la memoria dinámica utilizada con una lista de contratos.

Módulo *Departamento* (4 puntos) Máx. 50 empleados

Define un tipo **`tEmpleado`** con un puntero a un contrato y un número. Declara también un tipo **`tDepartamento`** para listas de **`tEmpleado`** implementadas con **array dinámico** que guarden el identificador del departamento y el contador además del array dinámico. Estas listas no guardan ningún orden.

Implementa, al menos, los siguientes subprogramas:

- ✓ `inicializar()`. Dado un identificador, devuelve un departamento vacío.
- ✓ `buscarEmpleado()`. Busca un empleado en un departamento usando el nif. Devuelve -1 si no lo encuentra.
- ✓ `insertarEmpleado()`. Se inserta un empleado nuevo con respecto a un contrato que se selecciona mostrando al usuario la lista de contratos. El empleado no se inserta si ya existe su nif en el departamento. En ese caso no se hace nada. La inserción se realiza al final del departamento.
- ✓ `mostrarEmpleado()`. Dado un empleado muestra sus datos en una línea según el formato dado.
- ✓ `mostrarDepartamento()`. Muestra toda la información de un departamento siguiendo el formato dado. Debe implementarse utilizando una función recursiva.
- ✓ `liberar()`. Libera la memoria dinámica utilizada con un departamento.

Programa Principal

Carga el archivo **`prioridades.txt`** en la tabla de prioridades, carga el contenido del archivo **`contratos.txt`** en la lista maestra de contratos, crea un nuevo departamento vacío (solicitando el identificador y año de gestión) y muestra al usuario un menú con

tres opciones más la de salida (opción 0): insertar un empleado en el nuevo departamento, mostrar las tareas de un empleado del departamento y mostrar los empleados actuales del departamento. Al salir libera toda la memoria dinámica utilizada.

```
int main() {
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
    tDepartamento dpto;
    tListaContratos lista;
    tPrioridades prioridades;
    tEmpleado empleado;
    string id, nif;
    int opcion, pos = -1;
    bool ok = false;
    cargarPrioridades(prioridades, ok);
    if (ok) {
        cargarContratos(lista, prioridades, ok);
        if (ok) {
            cout << "Identificador del departamento: ";
            cin >> id;
            cout << endl;
            dpto = inicializar(id);
            do {
                opcion = menu();
                switch (opcion) {
                  case 1:
                     insertarEmpleado(lista, dpto, ok);
                     if (!ok) {
                       cout << "Error al insertar el empleado." << endl;
                     }
                     break;
                  case 2:
                     cout << "Nif del empleado: ";
                     cin >> nif;
                     cout << endl;
                     pos = buscarEmpleado(dpto, nif);
                     if (pos != -1) {
                        mostrarTareas(dpto.empleados[pos].puntero->lista);
                     } else {
                        cout << "El empleado no existe." << endl;
                     }
                     cout << endl;
                     break;
                  case 3:
                     if (dpto.contador != 0) {
                        cout << "Los empleados son:" << endl;
                        mostrarDepartamento(dpto, 0);
                     } else {
                        cout << "No hay empleados." << endl;
                     }
                     cout << endl;
                     break;
                }
```

```
            } while (opcion != 0);
            liberar(lista);
            liberar(dpto);
        } else {
            cout << "Error al cargar la lista de contratos." << endl;
        }
    } else {
        cout << "Error al cargar las prioridades." << endl;
    }
    return 0;
} // main
```

Ejemplo de archivo `prioridades.txt`

```
5 5 4 3 1 2 4
5 5 4 4 2 1 3
4 5 3 3 3 2 3
5 4 3 2 2 3 3
4 5 4 3 3 2 4
```

Ejemplo de archivo `contratos.txt`

El siguiente archivo **`contratos.txt`** contiene tres contratos. El primer contrato tiene una lista de dos tareas, el segundo de una tarea y el último de dos tareas.

```
3
Sanchez Gomez, Juan
12345678A
1700
2
1
Docencia en Ingenieria del Software
7
Publicacion de articulos y conferencias
Martin Vazquez, Alberto
87654321A
1600
1
7
Publicacion de articulos y conferencias
Gonzalez Fernandez, Patricia
32323232B
1700
2
2
Docencia en Ingenieria de Computadores
7
Publicacion de articulos y conferencias
```

Ejemplos de ejecución: opciones 1, 2 y 3

```
Identificador del departamento: id1

1. Insertar empleado en el nuevo departamento.
2. Mostrar las tareas de un empleado.
3. Mostrar los empleados actuales del departamento.
0. Salir.
Elige opcion: 1

La lista de contratos es:
1. Sanchez Gomez, Juan            12345678A  (2) 1700.00
2. Gonzalez Fernandez, Patricia   32323232B  (2) 1700.00
3. Martin Vazquez, Alberto        87654321A  (1) 1600.00

Introduce el numero de linea del contrato: 1
Numero para el empleado: 1

1. Insertar empleado en el nuevo departamento.
2. Mostrar las tareas de un empleado.
3. Mostrar los empleados actuales del departamento.
0. Salir.
Elige opcion: 2

Nif del empleado: 12345678A

La lista de tareas es:
1  Docencia en Ingenieria del Software        80
7  Publicacion de articulos y conferencias    60


1. Insertar empleado en el nuevo departamento.
2. Mostrar las tareas de un empleado.
3. Mostrar los empleados actuales del departamento.
0. Salir.
Elige opcion: 3

Los empleados son:
1. Sanchez Gomez, Juan            num: 1

1. Insertar empleado en el nuevo departamento.
2. Mostrar las tareas de un empleado.
3. Mostrar los empleados actuales del departamento.
0. Salir.
Elige opcion:
```

Se valorará la legibilidad, así como el uso adecuado de los esquemas de recorrido y búsqueda, de la comunicación entre funciones y de la memoria.

Entrega el código del programa (sólo .cpp y .h comprimidos en un ZIP) utilizando la herramienta de FTP del escritorio. Asegúrate de entregar una versión sin errores de compilación.

Memoria Dinámica

El comando para que se muestre la memoria no liberada es:

```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

// archivo checkML.h

```
#ifdef _DEBUG
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif
```

Entrega del Examen

1. Añade al inicio de tus archivos un comentario con tus datos:

```
/*
Apellidos y Nombre:
DNI:
Puesto:
*/
```

2. Abre la herramienta de entrega de exámenes por FTP que hay en el escritorio de tu ordenador.
3. Úsala para subir tus archivos (arrastra tus ficheros hacia la ventana derecha).
4. Pasa por el ordenador del profesor, pregúntale si tu archivo se ha recibido correctamente, y firma.