0.1 – Planificación de horarios (A)

Estructuras de Datos
Facultad de Informática - UCM

Supongamos que queremos planificar las actividades de un congreso. Cada actividad tiene una duración diferente. Para almacenar la duración de cada una de ellas utilizamos el siguiente tipo de datos:

```
struct Duracion {
  int dias;      // valor contenido entre 0 y 100
  int horas;     // valor contenido entre 0 y 23
  int minutos;   // valor contenido entre 0 y 59
  int segundos;  // valor contenido entre 0 y 59
};
```

Por otro lado, el congreso también tiene una duración máxima, y queremos saber si hay tiempo suficiente para poder abarcar todas las actividades propuestas. Supongamos que tenemos una lista con las duraciones de cada una de estas actividades, y que también sabemos la duración del congreso.

Implementa la siguiente función:

```
bool caben_todas(
    const vector<Duracion> &duracion_actividades,
    const Duracion &duracion_congreso
    );
```

Esta función recibe el listado con la duración de las actividades, la duración máxima del congreso y debe devolver `true` si el tiempo ocupado por todas las actividades es menor o igual que la duración del congreso, o `false` en caso contrario.

Solución

```cpp
#include <iostream>
#include <vector>

using namespace std;

struct Duracion {
  int dias;
  int horas;
  int minutos;
  int segundos;
};


void normalizar_duracion(Duracion &t) {
  if (t.segundos >= 60) {
    t.minutos++;
    t.segundos -= 60;
  }

  if (t.minutos >= 60) {
    t.horas++;
    t.minutos -= 60;
  }

  if (t.horas >= 24) {
    t.dias++;
    t.horas -= 24;
  }
}

Duracion sumar_duracion(const Duracion &t1, const Duracion &t2) {

  Duracion result;
  result.segundos = t1.segundos + t2.segundos;
  result.minutos = t1.minutos + t2.minutos;
  result.horas = t1.horas + t2.horas;
  result.dias = t1.dias + t2.dias;

  normalizar_duracion(result);

  return result;
}
```

```
bool duracion_menor_que(const Duracion &t1, const Duracion &t2) {
  return (t1.dias < t2.dias) ||
    t1.dias == t2.dias && (
        t1.horas < t2.horas ||
        t1.horas == t2.horas && (
          t1.minutos < t2.minutos ||
          t1.minutos == t2.minutos && t1.segundos < t2.segundos
        )
    );
}

Duracion duracion_cero() {
  return {0, 0, 0, 0};
}

bool caben_todas(const vector<Duracion> &duracion_actividades,
                 const Duracion &duracion_congreso) {

  Duracion actual = duracion_cero();

  for (int i = 0; i < duracion_actividades.size(); i++) {
    actual = sumar_duracion(actual, duracion_actividades[i]);
  }

  return duracion_menor_que(actual, duracion_congreso);
}
```