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 va a haber tiempo suficiente para poder abarcar todas las actividades propuestas. Vamos a suponer que tenemos un listado con las duraciones de cada una de estas actividades, una por línea, y cada una con el formato `HH:MM:SS`, donde `HH` indica el número de horas, `MM` indica el número de minutos y `SS` indica el número de segundos. No indicamos el número de días porque suponemos que todas las actividades duran menos de un día. El listado finaliza con la duración `00:00:00`.

Implementa la siguiente función:

```
bool caben_todas(const Duracion &duracion_congreso);
```

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

Solución

```
#include <iostream>

using namespace std;

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

const int LONG_STRING = 11;

Duracion from_string(const string &duracion_str) {
  if (duracion_str.length() == LONG_STRING) {
    return { stoi(duracion_str.substr(0, 2)),
             stoi(duracion_str.substr(3, 2)),
             stoi(duracion_str.substr(6, 2)),
             stoi(duracion_str.substr(9, 2)) };
  } else {
    return { 0,
             stoi(duracion_str.substr(0, 2)),
             stoi(duracion_str.substr(3, 2)),
             stoi(duracion_str.substr(6, 2)) };
  }
}


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 Duracion &duracion_congreso) {
  string hora_str;
  cin >> hora_str;

  Duracion actual = duracion_cero();

  while (hora_str != "00:00:00") {
    actual = sumar_duracion(actual, from_string(hora_str));
    cin >> hora_str;
  }

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