Práctica 2: Documentación de Software

Antonio Manuel Fernández Cantos

9 de noviembre de 2015

Índice

1. Introducción **2**

2. Clase Pregunta **2**
2.1. Especificación 2
2.2. Diferentes estructuras de datos para tipo rep. 2
2.3. tipo rep escogido 2
2.4. Función de Abstracción 3
2.5. Invariante de la representación 3
2.6. Operaciones 3

3. Clase ConjuntoPreguntas **3**
3.1. Especificación 3
3.2. Diferentes estructuras de datos para tipo rep. 3
3.3. tipo rep escogido 4
3.4. Función de Abstracción 4
3.5. Invariante de la representación 4
3.6. Operaciones 4

4. Clase Test **5**
4.1. Especificación 5
4.2. Diferentes estructuras de datos para tipo rep. 5
4.3. Tipo rep escogido 5
4.4. Funcion de Abstracción 5
4.5. Invariante de la representación 5
4.6. operaciones 5

5. Conclusiones **6**

1. Introducción

La práctica consiste en generar una prueba tipo test. Las preguntas y sus respectivas respuestas se obtiene de un archivo txt. Hay dos opciones de mostrar la prueba tipo test. La primera es por pantalla y la segunda opción es generando un archivo de texto con las preguntas y sus respuestas. Este documento contiene las funciones de abstracción, el invariante de representación, los tipo rep disponibles y el tipo rep escogido para el desarrollo de la práctica y las operaciones implementadas. Para conocer mejor que hacen las funciones de los distintos módulos lo mejor es utilizar doxygen.

2. Clase Pregunta

2.1. Especificación

Representa una pregunta con sus respuestas.

2.2. Diferentes estructuras de datos para tipo rep.

1. Primera posibilidad:

```
1 class Pregunta{
2 private:
3   char pregunta[1000];
4   char respuestas[10][1000];
5   int n;
6 };
```

2. Segunda posibilidad:

```
1 class Pregunta{
2 private:
3  string pregunta;
4  string *respuestas;
5  int n;
6 };
```

2.3. tipo rep escogido

El tipo rep escogido ha sido la segunda opción ya que aprovecharemos al máximo la memoria. Si usamos vectores estáticos estaremos desperdiciando mucha memoria cuando tengamos pregutas cortas. Además si tenemos más de 10 respuestas para una pregunta, no entrara la respuesta 11 dentro del vector.

2.4. Función de Abstracción

Si r es un objeto de tipo pregunta: En cada linea saldría cada uno de los elementos de la función. f(r) = 'r.pregunta' 'r.respuestas[0]' 'r.respuestas[1]' '...' 'r.respuestas[n-1]'

2.5. Invariante de la representación

Condiciones que hacen que el tipo rep sea valido para representar el T.D.A En la función anterior sería la siguiente: r.pregunta debe ser un string valido, r.respuestas debe ser otro string valido y r.n mayor estricto que uno.

2.6. Operaciones

1. Constructor con una pregunta y respuestas concretas
2. Constructor sin parámetros
3. Destructor
4. operador de asignación =
5. getPregunta
6. getNumRespuestas
7. getRespuesta(con una respuesta concreta)

3. Clase ConjuntoPreguntas

3.1. Especificación

Representa un conjunto de preguntas con las respuestas de cada pregunta.

3.2. Diferentes estructuras de datos para tipo rep.

1. Primera posibilidad:

```
1 class ConjuntoPreguntas{
2 private:
3   Pregunta *preguntas;
4   int num_preg;
5   int reservados;
6 };
```

2. Segunda posibilidad:

```
class ConjuntoPreguntas{
private:
  Pregunta preguntas[5000];
  int num_preg;

};
```

3.3. tipo rep escogido

El tipo rep escogido es la primera opción ya que como hemos visto en la clase pregunta, no desperdiciamos memoria.

3.4. Función de Abstracción

Si r es un objeto de tipo ConjuntoPreguntas Para numpreg mayor estricto que 0 y menor o igual que reservados:

f(r) = 'r.preguntas[numpreg].getPregunta' 'r.preguntas[numpreg].getRespuesta(posicion=0)' '...' 'r.respuestas[numpreg].getRespuesta(posicion=Número de respuestas - 1)'

Donde posición indica el número que ocupa la respuesta de una pregunta concreta.

3.5. Invariante de la representación

Esta sección indica las condiciones que hacen que el tipo rep sea valido para representar el TDA. En la función anterior sería que numpreg mayor estricto que 0 y que numpreg sea menor o igual que reservados. Además la posición que ocupa la respuesta dentro de la pregunta debe de estar dentro del rango de las respuestas guardadas en una pregunta concreta.

3.6. Operaciones

1. Constructor para reservar memoria. Si se introduce un número que no es valido, resserva 20 posiciones por defecto.
2. Destructor
3. Size
4. Resize para disminuir o aumentar el vector
5. getPregunta
6. getNumRespuestas
7. getRespuestaPregunta que devuelve una respuesta concreta de una pregunta concreta.

4. Clase Test

4.1. Especificación

Crea pruebas tipo test de un fichero con preguntas, el número de respuestas y sus respuestas.

4.2. Diferentes estructuras de datos para tipo rep.

1. Primera posibilidad:

```
1 class Test{
2 private:
3   Pregunta *preguntas;
4   int numeropreguntas;
5 };
```

2. Segunda posibilidad:

```
1 class Test{
2 private:
3   Pregunta preguntas[1000];
4   int numeropreguntas;
5 };
```

4.3. Tipo rep escogido

Siguiendo la misma linea que las anteriores, he cogido la primera opción ya que usa mejor la memoria del programa.

4.4. Funcion de Abstracción

Si r es un objeto de tipo Test su función sería la siguiente: f(r) = 'r.pregunta[0].getPregunta' 'r.pregunta[0].getRespuesta(0)' 'r.pregunta[0].getRespuesta(1)' ... 'r.pregunta[numeropreguntas-1].getRespuesta(0)' ... 'r.pregunta[numeropreguntas-1].getRespuesta(r.pregunta[numeropreguntas-1].getNumRespuestas-1)'

4.5. Invariante de la representación

Condiciones que hacen que el tipo rep sea valid para representar el TDA. En este caso número de preguntas tiene que ser mayor estricto que 0, el número de respuestas tiene que estar dentro del rango de las respuestas.

4.6. operaciones

1. Constructor para almacenar un número concreto de preguntas. Se recibe un conjunto de preguntas y de ahí las obtenemos.

2. Destructor

3. Clase amiga ofstream para escribir las preguntas y respuestas en un fichero.

4. Clase amiga ostream para escribir las preguntas y respuestas por pantalla.

5. Conclusiones

Con la realización de la práctica he aprendido a usar mejor los const de los métodos de las clases, el uso de las funciones amigas ifstream, ofstream y ostream. He mejorado el uso del makefile ya que antes ponía todos los includes en el .h y .cpp y en esta práctica lo he hecho de forma diferente como puedes ver en los archivos .h y .cpp. Otra de las mejoras ha sido el uso de doxygen ya que nunca lo había usado y es un buen método para comentar los métodos. Uno de los aspectos negativos es que no he podido probar a implementar las clases usando celdas enlazadas ya que no he dispuesto de mucho tiempo para realizar la práctica. En definitiva la práctica ayuda a mejorar la técnica de programar separando las clases en módulos haciendo más comoda la detección de errores y la implementación de los métodos.