Diseño Orientado a Objetos

Trabajo Práctico Final

Universidad Nacional de Rosario, Instituto Politécnico, Dto. de Informática,
informatica@ips.edu.ar,
WWW home page: http://informatica.ips.edu.ar

Resumen Este trabajo final busca que el que realicen un proyecto que cierre con los conocimientos que se fueron dando, incorporen en el mismo sistema muchas de las características del lenguaje.
El proyecto fue adaptado de otro, no diseñado por esta cátedra, que pertenece a la Universidad de Bar Ilán, de la ciudad de Ramat Gan, en la carrera de Ciencias de la Computación.
Se explica, abajo, con detalle lo que se pretende y se recomiendo lecturas complementarias. Lean cudadosamente los enunciados.
Se implementará un Manejador de Cache Genérico, que se salvará un *pair* compuesto por clave y valor, ambos en RAM y en disco (files).
Dada una clave, el cache manager accede al valor. Si el valor existe en la memoria es retornado. En caso que no exista en memoria, entonces el cache deberá buscar la clave en el archivo y retornar el *pair* si es encontrado.
El caché también debe implementar un tamaño límite de capacidad, lo que significa una restricción sobre cuántos elementos pueden existir a la vez en la memoria. Una vez que se ha alcanzado el límite, el menos utilizado recientemente (LRU) debe eliminarse de la memoria para ingresar al nuevo.
En el momento de la inserción, todo nuevo par $< key, valor >$ debe guardarse tanto en la memoria como en el archivo.

1. Especificaciones

1.1. Class CacheManager

Las funciones públicas que deben existir en su clase son:

void insert (string key, T Obj)
- Si existe, actualizará al objeto tanto en cache como en el archivo.
- De lo contrario, escribirá el objeto en el sistema de archivos y la caché
- Se actualizará el MRU del Obj para que sea el mayor.

T get (string key)
- Si el obj existe, sea en la caché o en el archivo, lo retorna y actualiza el MRU.
- Si la clave no existe tanto en el sistema principal como en el sistema de archivos, el programa debe informarlo

CacheManager(string capacity)
- Debe inicializar la capacidad de la cache.

IMPORTANTE cualquier llamada que utiliza una clave (insertar u obtener) actualizará el par $< clave, < obj, valor >>$ para ser el Mayor Recientemente Usado.

- La clave (key) siempre será un *string*.
- El Obj debe ser un objeto de tipo T creado en el main y pasado a las funciones. Se detallará mas adelante las especificaciones de dicho objeto.
- El constructor solo debe aceptar la capacidad del cache.

Puede y debe agregar toda función que crea necesaria para el funcionamiento y la operatividad de la cache sea en RAM o en el archivo

1.2. T Object

El tipo T debe ser un tipo de una clase que se implemente en el main. En los archivos adjuntos está el ejemplo de main.

Se deberá tener en cuenta:

- Los miembros deben ser tipos primitivos (string, int, float, double, etc.
- No deben usarse datos complejos ni punteros.
- Cada clase que se implemente debe tener un *static* con el nombre de la clase. Debe ser inicializado en el main.
- Por cada clase que se implemente debe tenerse una función print() o la respectiva sobrecarga del operador.
- No hay limites de los datos miembros de las clases

1.3. Files

- Cada llamada a insert(), La inserción del T Obj se realizará en la cache y en el archivo. Si ya existía la clave, actualiza el Obj.
- Los nombres de los archivos serán arbitrarios y los podrán elegir segun deseen.
- Al realizar un get(), si no se encuentra en la cache en RAM se lo buscará en el archivo. Si se lo encuentra, se retorna y guarda en la cache de RAM, si esta esta completa, se quitará el menos usados (LRU) más bajo.

En el archivo no debe guardarse el MRU ya que este es válido para la cache de RAM. Ejemplo de lo que se ha dicho anteriormente. Si al buscar la clave, no esta en RAM, pero se encuentra en el archivo, este Obj se incorpora a la cache RAM y el MRU se seteará con el valor mas alto, o sea, si el MRU existente es 10, el nuevo par incresará con MRU 11.

1.4. Presentación del TP

Deben subirse todos los archivos que se generen.

Compilen en linux usando -Wall -std=c++11

El archivo *cache.h* sería el siguiente:

```
#include <iostream>
#include <fstream>
#include <map>
#include <utility>
#include <string>

using namespace std;

template<class T>
class CacheManager {
//members (private)
   int capacity;
   map <string, pair<T, int>> cache_data; //<Clave,<Obj, Indice de Uso>>

   bool write_file(string,T);

public:

   CacheManager(int); //recibe la capacidad en el int
   ~CacheManager();

   void insert(string key, T obj);
   T get(string key);

   // Agregar todas las funciones necesarias

};

template <class T>
CacheManager<T>::CacheManager(int cap) {
   capacity = cap;
}

template <class T>
CacheManager<T>::~CacheManager() {}

template <class T>
bool CacheManager<T>::write_file(string key, T obj) {

   return true;
}

//INSERT
template <class T>
void CacheManager<T>::insert(string key, T obj) {

}

```

```
49  template <class T>
50  T CacheManager<T>::get(string key) {
51
52  }
```

Y el *main.cpp* podría ser por ejemplo el siguiente:

```
1   class Student
2   {
3       int id;
4       int value;
5       const char* data;
6   public:
7       static const string class_name; //Cada clase tiene un
            static llamado: class_name
8       Student(int _key, int _value, const char* _data): id(_key
            ), value(_value), data(_data) {}
9
10      void print() {
11          cout<<"Student Object: "<<id<<", "<<value<<", "<<data
                <<endl;
12      };
13      ~Student(){};
14
15  };
16  const string Student::class_name = "StudentClass";
17
18  int main() {
19      CacheManager<Student> my_cache(3);
20
21      my_cache.insert("0", Student(0,22,"student1"));
22      my_cache.insert("1", Student(1,23,"student2"));
23      my_cache.insert("2", Student(2,25,"student3"));
24      my_cache.insert("3", Student(3,29,"student4"));
25
26      my_cache.show_cache();
27
28      cout << "---------------------- UPDATE
            ---------------------- " << endl;
29      my_cache.insert("2", Student(22,222,"EZE"));
30      my_cache.show_cache();
31
32      cout << "
            <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
            " << endl;
33      Student return_obj = my_cache.get("0"); //not in cache
            but in filesystem
```

```
34      cout << "
            <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
            " << endl;
35
36      my_cache.insert("2", Student(2,25,"EZE"));
37      my_cache.show_cache();
38
39      my_cache.insert("9", Student(1,5,"Nine"));
40      my_cache.insert("9", Student(1,5,"Nine"));
41      my_cache.insert("9", Student(1,5,"Nine"));
42      my_cache.show_cache();
43
44      return 0;
45  }
```

Tenga encuenta hacer de forma completa los casos de prueba.