МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЧЁТ по лабораторной работе №1**

**по дисциплине «программирование»**

**«Изучение циклических алгоритмов, операторов цикла,**

**программирование циклического вычислительного процесса»**

|  |  |
| --- | --- |
| Факультет: АВТ  Группа: АБс-223  Студент: Басалаев Ярослав | Преподаватели: Ершов И.А, Архипова А.Б |

Новосибирск, 2023 г.

**Разработать no-SQL СУБД, отвечающую следующим требованиям:**

**1. База данных умеет хранить данные в следующих контейнерах:**

**1.1 Множество**

**1.2 Стэк**

**1.3 Очередь**

**1.4 Хэш-таблица.**

**2. Для простоты ЛР необходимо поддерживать только строковый**

**тип данных.**

**3. Каждый тип данных поддерживает операцию вставки, доступа**

**к данным и удаления.**

**4. Все данные СУБД сохраняет в указанный файл.**

**5. Все операции выполняются за O(1)**Код программы на С++:

laba.cpp

#include <iostream>

#include <string>

#include <fstream>

#include “hashmap.h”

#include “queue.h”

#include “set.h”

#include “stack.h”

#include <map>

extern nti int capacity = 200;

int main() {

std::string command, container\_name;

std::map<std::string, Node\*> Stacks;

std::map<std::string, List\*> Queues;

Set MySet[capacity];

Hashtb MyHash[capacity];

init\_table\_hash(MyHash);

init\_table\_set(MySet);

List\* Queue;

Node\* Stack;

while (1)

{

std::string line;

getline(std::cin, line);

std::string words[4];

std::string word = “”;

int j = 0;

for (size\_t i = 0; i < line.length(); i++)

{

if (i == line.length() – 1)

{

word += line[i];

words[j++] = word;

word = “”;

continue;

}

if (line[i] == ‘ ‘)

{

words[j++] = word;

word = “”;

continue;

}

word += line[i];

}

command = words[0];

container\_name = words[1];

if (command[0] == ‘S’ && command[1] == ‘P’) //Stack

{

std::map <std::string, Node\*>::iterator it;

it = Stacks.find(container\_name);

if (it != Stacks.end())

{

Stack = it->second;

if (command == “SPUSH”)

{

SPUSH(&Stack, words[2]);

Stacks[container\_name] = Stack;

}

else if (command == “SPOP”)

{

std::cout << SPOP(&Stack) << ‘\n’;

Stacks[container\_name] = Stack;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

}

else

{

Stack = new Node();

init\_stack(Stack);

if (command == “SPUSH”)

{

SPUSH(&Stack, words[2]);

}

else if (command == “SPOP”)

{

std::cout << “Error” << ‘\n’;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

Stacks.insert(make\_pair(container\_name, Stack));

}

}

else if (command[0] == ‘Q’) //Queue

{

std::map<std::string, List\*>::iterator it;

it = Queues.find(container\_name);

if (it != Queues.end())

{

Queue = it->second;

if (command == “QPUSH”)

{

QPUSH(Queue, words[2]);

Queues[container\_name] = Queue;

}

else if (command == “QPOP”)

{

std::cout << QPOP(Queue) << ‘\n’;

Queues[container\_name] = Queue;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

}

else

{

Queue = new List();

init\_list(Queue);

if (command == “QPUSH”)

{

QPUSH(Queue, words[2]);

}

else if (command == “QPOP”)

{

std::cout << “Error” << ‘\n’;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

Queues.insert(make\_pair(container\_name, Queue));

}

continue;

}

else if (command[0] == ‘S’) //Set

{

if (command == “SADD”)

{

SADD(MySet, words[1]);

}

else if (command == “SREM”)

{

SREM(MySet, words[1]);

}

else if (command == “SISMEMBER”)

{

std::cout << SISMEMBER(MySet, words[1]) << ‘\n’;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

continue;

}

else if (command[0] == ‘H’) //Hash

{

if (command == “HSET”)

{

HSET(MyHash, stoi(words[1]), words[2]);

}

else if (command == “HDEL”)

{

HDEL(MyHash, stoi(words[1]));

}

else if (command == “HGET”)

{

std::cout << HGET(MyHash, stoi(words[1])) << ‘\n’;

}

else

{

std::cout << “Unknown command!” << ‘\n’;

}

continue;

}

}

}

stack.cpp

#include <iostream>

#include <string>

#include “stack.h”

void init\_stack(Node\* head)

{

head->next = NULL;

}

void SPUSH(Node\*\* head, std::string new\_value)

{

Node\* new\_node = new Node();

new\_node->value = new\_value;

new\_node->next = (\*head);

\*head = new\_node;

}

std::string SPOP(Node\*\* head)

{

if ((\*head) == NULL)

{

return “Empty stack”;

}

Node\* new\_node = (\*head);

std::string result;

result = new\_node->value;

(\*head) = new\_node->next;

delete new\_node;

return result;

}

queue.cpp

#include <iostream>

#include <string>

#include “queue.h”

#include “stack.h”

void init\_list(List\* ls)

{

ls->head = NULL;

ls->tail = NULL;

ls->size = 0;

}

void QPUSH(List\* ls, std::string new\_value)

{

Node\* new\_node = new Node();

new\_node->value = new\_value;

new\_node->next = ls->tail;

if (ls->size == 0)

{

ls->head = new\_node;

}

else

{

ls->tail->next = new\_node;

}

ls->tail = new\_node;

ls->size++;

}

std::string QPOP(List\* ls)

{

if (ls->size == 0)

{

return “Empty queue”;

}

Node\* new\_node = ls->head;

std::string result = ls->head->value;

if (ls->size == 1)

{

ls->head = NULL;

ls->tail = NULL;

}

else

{

ls->head = new\_node->next;

}

delete new\_node;

ls->size--;

return result;

}

set.cpp

#include <iostream>

#include <string>

#include “set.h”

extern nti int capacity;

long hashFuncSet(std::string key)

{

long hash = 5381;

for (nti = 0; i < key.length(); i++)

{

hash = ((hash << 5) + hash);

}

return hash;

}

void init\_table\_set(Set\* tbl)

{

for (nti = 0; i < capacity; i++)

{

tbl[i].key = ‘0’;

tbl[i].data = ‘0’;

}

}

void SADD(Set\* tbl, std::string key)

{

int index = hashFuncSet(key);

if (tbl[index % capacity].key == “0”)

{

tbl[index % capacity].key = key;

std::cout << “Key “ << key << “ has inserted” << ‘\n’;

}

else

{

std::cout << “Hash collision” << ‘\n’;

}

}

void SREM(Set\* tbl, std::string key)

{

int index = hashFuncSet(key);

std::string value;

if (tbl[index % capacity].key == “0”)

{

return ;

}

else

{

value = tbl[index % capacity].key;

tbl[index % capacity].key = “0”;

}

}

std::string SISMEMBER(Set\* tbl, std::string key)

{

int index = hashFuncSet(key);

if (tbl[index % capacity].key == “0”)

{

return “No”;

}

else

{

if (tbl[index % capacity].key == key)

{

return “Yes”;

}

}

}

Hashmap.cpp

#include <iostream>

#include <cmath>

#include <string>

#include “hashmap.h”

extern nti int capacity;

int hashFunc(int key)

{

return (key % capacity);

}

void init\_table\_hash(Hashtb\* tbl)

{

for (nti = 0; i < capacity; i++)

{

tbl[i].key = 0;

tbl[i].data = ‘0’;

}

}

void HSET(Hashtb\* tbl, int key, std::string data)

{

int index = hashFunc(key);

if (tbl[index].data == “0”)

{

tbl[index].key = key;

tbl[index].data = data;

std::cout << “Key “ << key << “ has inserted” << ‘\n’;

}

else

{

std::cout << “Hash collision” << ‘\n’;

}

}

void HDEL(Hashtb\* tbl, int key)

{

int index = hashFunc(key);

std::string value;

if (tbl[index].data == “0”)

{

std::cout << “No such key exists” << ‘\n’;

return ;

}

else

{

value = tbl[index].data;

tbl[index].key = 0;

tbl[index].data = “0”;

}

return ;

}

std::string HGET(Hashtb\* tbl, int key)

{

int index = hashFunc(key);

if (tbl[index].data == “0”)

{

return “No such key exists”;

}

else

{

return tbl[index].data;

}

}

stack.h

#pragma once

#include <iostream>

#include <string>

#include “stack.h”

struct Node {

std::string value;

Node\* next;

};

void init\_stack(Node\* head);

void SPUSH(Node\*\* head, std::string new\_value);

std::string SPOP(Node\*\* head);

queue.h

#pragma once

#include <iostream>

#include <string>

#include “queue.h”

#include “stack.h”

struct List {

Node\* head;

Node\* tail;

size\_t size;

};

void init\_list(List\* ls);

void QPUSH(List\* ls, std::string new\_value);

std::string QPOP(List\* ls);

hashmap.h

#pragma once

#include <iostream>

#include <string>

#include “hashmap.h”

struct Hashtb {

int key;

std::string data;

};

int hashFunc(int key);

void init\_table\_hash(Hashtb\* tbl);

void HSET(Hashtb\* tbl, int key, std::string data);

void HDEL(Hashtb\* tbl, int key);

std::string HGET(Hashtb\* tbl, int key);

set.h

#pragma once

#include <iostream>

#include <string>

#include “set.h”

struct Set {

std::string key;

std::string data;

};

long hashFuncSet(std::string key);

void init\_table\_set(Set\* tbl);

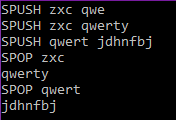
void SADD(Set\* tbl, std::string key);

void SREM(Set\* tbl, std::string key);

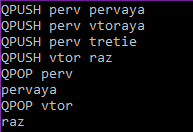
std::string SISMEMBER(Set\* tbl, std::string key);

Пример работы программы

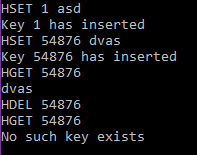
Стэк:



Очередь:



Хэш-таблица:



Множество:

