Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “ Динамические списки”

Выполнил студент группы 21ВВ3:

Ванюшин И.А

Приняли:

Митрохин М.А., Юрова О.В.

Пенза 2022

**Название**

Динамические списки.

**Цель работы:** выполнить лабораторные указания 1 реализуя приоритетную очередь, путем добавления элемента в список в соответствии с приоритетом объекта. Создать структуры данных для лабораторных заданий 2 – 3, реализовав для них определенные функции.

**Лабораторное задание:**

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).
2. \* На основе приведенного кода реализуйте структуру данных *Очередь*.
3. \* На основе приведенного кода реализуйте структуру данных *Стек*.

**Задание 1:**

**Листинг**

Файл “PriorityQueueModule.h”:

#pragma once

#include "GlobalLibs.h"

#include <map>

class PriorityQueueModule {

private:

struct Node {

Data data;

Node\* next;

};

Node\* head, \* tail;

std::map<int, std::pair<Node\*, Node\*>> pQueue;

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

public:

PriorityQueueModule() {

head = NULL;

tail = NULL;

}

// Push data at the end of priority list.

void Push(int priority, Data data) {

SetPriorHead(priority);

if (it == pQueue.end()) {

head = new Node();

head->data = data;

head->next = NULL;

tail = head;

pQueue.insert(std::make\_pair(priority, std::make\_pair(head, tail)));

}

else {

Node\* Next = new Node();

Next->data = data;

Next->next = NULL;

tail->next = Next;

tail = Next;

}

}

// Pop data from priority list's begining.

Data Pop() {

it = pQueue.begin();

head = it->second.first;

tail = it->second.second;

Node\* temp;

Data data;

if (head->next == NULL) {

data = { head->data };

free(head);

head = NULL;

tail = NULL;

pQueue.erase(it);

}

else {

data = { head->data };

temp = head->next;

free(head);

head = temp;

pQueue[it->first] = std::make\_pair(head, tail);

}

return data;

}

// Find list's adress by priority and set into this.head and this.tail.

void SetPriorHead(int priority) {

it = pQueue.find(priority);

if (it != pQueue.end()) {

head = it->second.first;

tail = it->second.second;

}

}

// Delete entire queue.

void DeleteQueue(){

while(pQueue.size() != 0) {

it = pQueue.begin();

head = it->second.first;

tail = it->second.second;

Node\* prev = head, \* current = head->next;

if (current != NULL) {

while (current->next != NULL) {

free(prev);

prev = current;

current = current->next;

}

free(prev);

free(current);

}

else {

free(head);

head = NULL;

tail = NULL;

pQueue.erase(it);

};

}

}

~PriorityQueueModule() {

DeleteQueue();

}

};

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

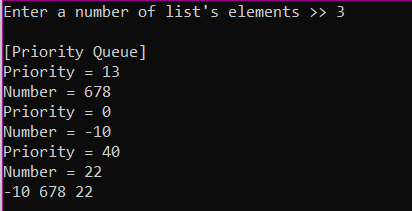
****

Рис. 1

Результат работы программы совпадает с ожидаемым.

**Задание 2:**

**Листинг**

Файл “QueueModule.h”:

#pragma once

#include "GlobalLibs.h"

class QueueModule {

private:

struct Node {

Data data;

Node\* next;

};

Node\* head, \* tail;

public:

QueueModule() {

head = NULL;

tail = NULL;

}

// Push data at the end of queue.

void Push(Data data) {

Node\* Next = new Node();

Next->next = NULL;

Next->data = data;

if (head == NULL) {

head = Next;

}

else {

tail->next = Next;

}

tail = Next;

}

// Pop data from queue's begining.

Data Pop() {

Data data = { -1 };

if (head != NULL) {

Node\* temp;

data = { head->data };

if (head->next != NULL) {

temp = head->next;

free(head);

head = temp;

}

else {

free(head);

head = NULL;

}

}

return data;

}

// Return true if data in queue.

bool Find(Data data) {

Node\* Current = head;

if (head != NULL) {

while (Current->next != NULL) {

if (Current->data == data) {

return true;

}

Current = Current->next;

}

}

return false;

}

// Print entire queue in console.

void ReadList() {

if (head != NULL) {

Node\* Current = head;

while (Current->next != NULL) {

std::cout << Current->data.data << " ";

Current = Current->next;

}

std::cout << Current->data.data;

}

else std::cout << "Elements not found.";

};

void DeleteNode(Data data) {

Node\* current, \* prev = head;

if (head == NULL) { return; }

current = head->next;

if (head->data == data) {

head = head->next;

free(prev);

return;

}

if (current == NULL) { return; }

while (current->next != NULL) {

if (current->data == data) {

prev->next = current->next;

free(current);

return;

}

prev = current;

current = current->next;

}

if (current->data == data) {

prev->next = NULL;

free(current);

return;

}

}

// Delete entire queue.

void DeleteQueue() {

if (head != NULL) {

Node\* prev = head, \* current = head->next;

if (current != NULL) {

while (current->next != NULL) {

free(prev);

prev = current;

current = current->next;

}

free(prev);

free(current);

}

else {

free(head);

head = NULL;

}

}

}

~QueueModule() {

DeleteQueue();

}

};

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

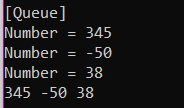
****

Рис. 1

Результат работы программы совпадает с ожидаемыми.

Вывод: в результате выполнения второго задания были получены навыки реализации структуры данных – очередь.

**Задание 3:**

**Листинг**

Файл “StackModule.h”:

#pragma once

#include "GlobalLibs.h"

class StackModule {

private:

struct Node {

Data data;

Node\* next;

};

Node\* head;

public:

StackModule() {

head = NULL;

}

// Push data at the begining of stack.

void Push(Data data) {

Node\* Next = new Node();

Next->data = data;

if (head == NULL) {

Next->next = NULL;

}

else {

Next->next = head;

}

head = Next;

}

// Pop data from stack's begining.

Data Pop() {

Data data = { -1 };

if (head != NULL) {

Node\* temp;

data = head->data;

if (head->next == NULL) {

free(head);

head = NULL;

}

else {

temp = head->next;

free(head);

head = temp;

}

}

return data;

}

// Return true if data in stack.

bool Find(Data data) {

Node\* current = head;

if (head != NULL) {

while (current->next != NULL) {

if (current->data == data) {

return true;

}

current = current->next;

}

}

return false;

}

// Print entire stack in console.

void ReadList() {

if (head != NULL) {

Node\* Current = head;

while (Current->next != NULL) {

std::cout << Current->data.data << " ";

Current = Current->next;

}

std::cout << Current->data.data;

}

else std::cout << "Elements not found.";

};

void DeleteNode(Data data) {

Node\* current, \* prev = head;

if (head == NULL) { return; }

current = head->next;

if (head->data == data) {

head = head->next;

free(prev);

return;

}

if (current == NULL) { return; }

while (current->next != NULL) {

if (current->data == data) {

prev->next = current->next;

free(current);

return;

}

prev = current;

current = current->next;

}

if (current->data == data) {

prev->next = NULL;

free(current);

return;

}

}

// Delete entire stack.

void DeleteStack() {

if (head != NULL) {

Node\* prev = head, \* current = head->next;

if (current != NULL) {

while (current->next != NULL) {

free(prev);

prev = current;

current = current->next;

}

free(prev);

free(current);

}

else {

free(head);

head = NULL;

}

}

}

~StackModule() {

DeleteStack();

}

};

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

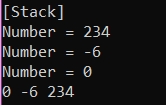
****

Рис. 1

Результат работы программы совпал с ожидаемыми.

Вывод: в результате выполнения третьего задания были получены навыки реализации структуры данных – стек.

**Вывод:** выполнил лабораторные указания 1-3 реализуя приоритетную очередь, путем добавления элемента в список в соответствии с приоритетом объекта, обычную очередь и стек.