Министерство образования РоссийскойФедерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по теме «Динамические списки»

Выполнили:

Студенты группы 21ВВ2

Назаров Е.А.

Макаров И.С.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза, 2022

**Цель работы:**

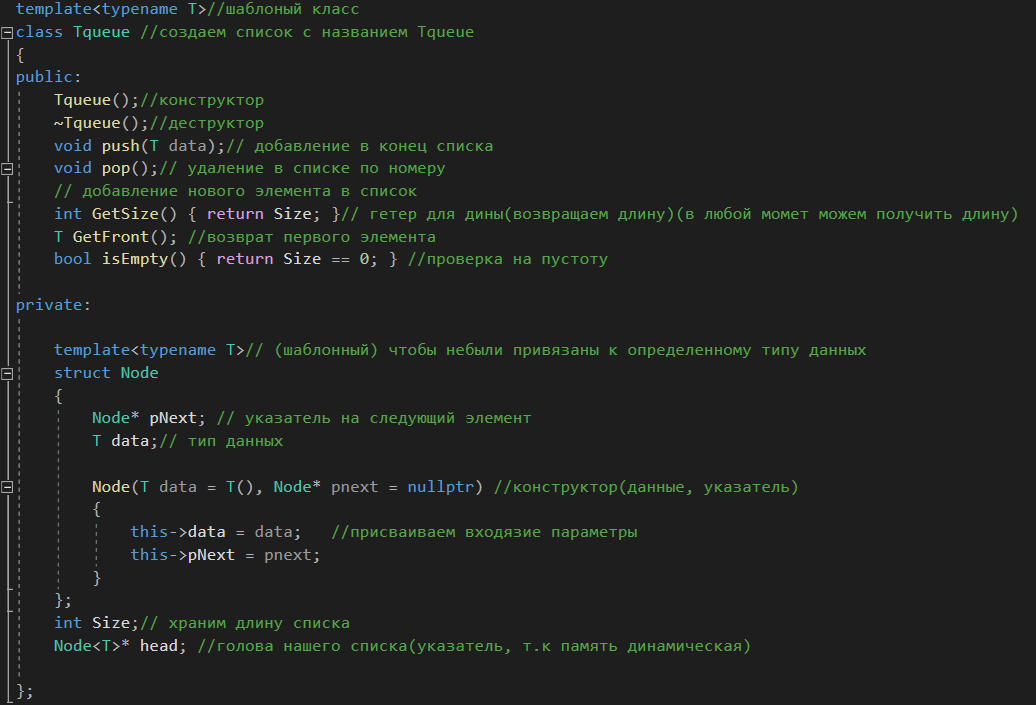
Изучение организации списковых структур и построение реальных структур данных на базе списков.

**Лабораторные работы:**

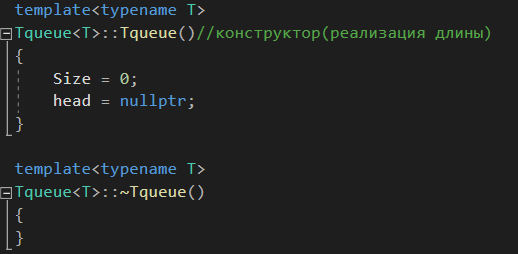
**Задание:**

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*.

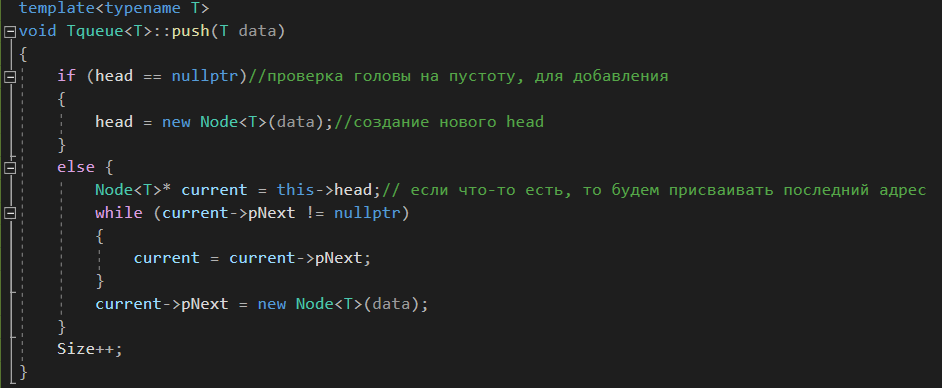
**Пояснительный текст к программе:**



Во всех работах был реализован один и тот же шаблонный односвязный список. В public версии были реализованы все основные функции нашего списка, в private был реализован сам класс элемента, тоже шаблонный. Он хранит в себе указатель на следующий элемент списка и тип переданных данных. Также были сразу созданные переменные отвечающие за начало списка и его длину.



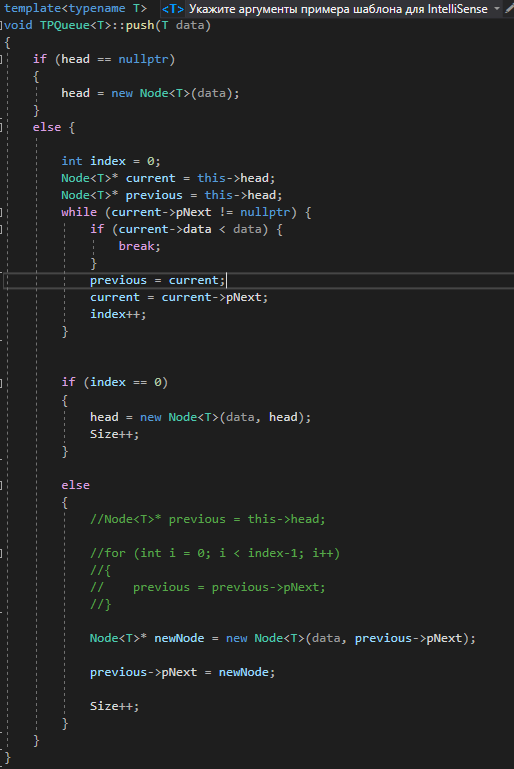
Реализация конструктора и деструктора. При создании списка Голове присваивается пустота и ее длина.

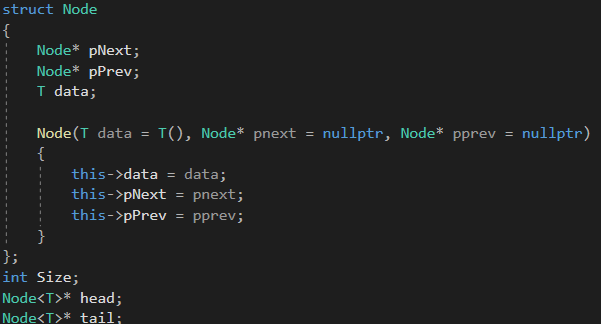


В 3 задании добавление происходит в конец списка по правилу очереди и стека. При добавлении первого элемента ему присваивается голова, в дальнейших случаях, проходит поиск последнего адреса, после чего ему присваиваются данные и указатель на следующий элемент(пустой).

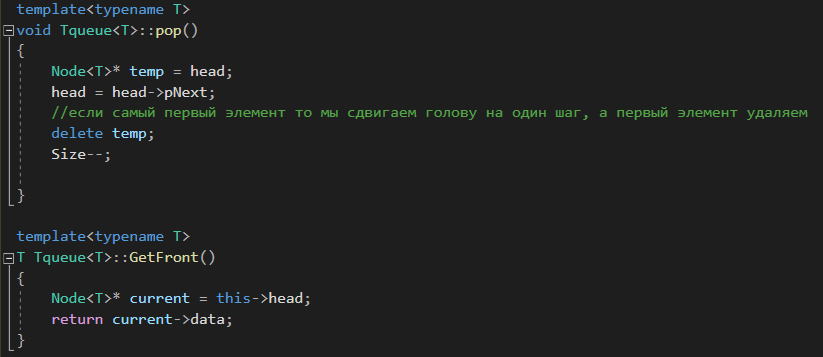


Во втором задании добавление происходит по принципу двухсвязного списка. Указателю на последний элемент списка (его хвосту), присваивается новый элемент.

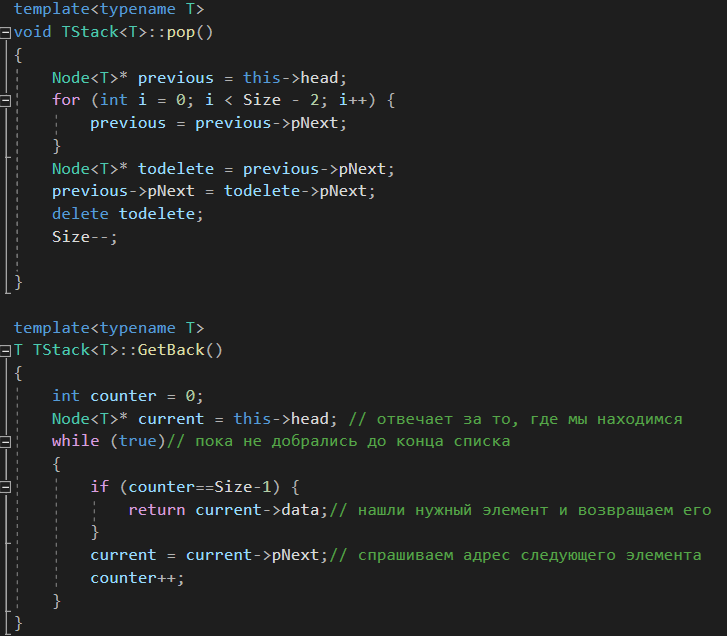




В 1 задании добавление в элемент списка происходит в соответствии правила приоритета (вперед идут переменные большие по значению). Если элемент при оказался в приоритете больше, то он встает на свое место получая адрес элемента, который стоял до этого на этом месте, тому же элементу присваивается новый адрес, сдвигая и все следующие элементы.



В заданиях 1 и 2 реализация операции удаления(pop) и вывода(GetFront) первого элемента в списке происходит одинаково. В первой процедуре, временной переменной присваивается голова списка, после чего она освобождается. Во второй процедуре, на экран выводится голова списка.



В 3 задание удаление и вывод происходят иначе, также создаются временные переменные, которые на этот раз, с помощью цикла, доходят до конца списка (последнего элемента), где и происходит удаление или вывод этой ячейки.

**Листинг:**

**Queue.cpp**

#include <iostream>

using namespace std;

template<typename T>

class Tqueue

{

public:

Tqueue();

~Tqueue();

void push(T data);

void pop();

int GetSize() { return Size; }

T GetFront();

bool isEmpty() { return Size == 0; }

private:

template<typename T>

struct Node

{

Node\* pNext;

Node\* pPrev;

T data;

Node(T data = T(), Node\* pnext = nullptr, Node\* pprev = nullptr)

{

this->data = data;

this->pNext = pnext;

this->pPrev = pprev;

}

};

int Size;

Node<T>\* head;

Node<T>\* tail;

};

template<typename T>

Tqueue<T>::Tqueue()

{

Size = 0;

head=tail = nullptr;

}

template<typename T>

Tqueue<T>::~Tqueue()

{

}

template<typename T>

void Tqueue<T>::push(T data)

{

Node<T>\* temp = new Node<T>(data);

temp->pNext = 0;

temp->data = data;

temp->pPrev = tail; //двухсвязный

if (tail != nullptr)

tail->pNext = temp;

if (Size == 0) {

head = tail = temp;

}

else {

tail = temp;

}

Size++;

/\* if (head == nullptr)

{

head = new Node<T>(data);

}

else {

Node<T>\* current = this->head; // односвязный

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

Size++;\*/

}

template<typename T>

void Tqueue<T>::pop()

{

Node<T>\* temp = head;

head = head->pNext;

delete temp;

Size--;

}

template<typename T>

T Tqueue<T>::GetFront()

{

Node<T>\* current = this->head;

return current->data;

}

int main()

{

Tqueue<int> a;

a.push(13);

a.push(23);

a.push(33);

a.push(43);

cout << a.GetFront();

cout << endl << a.GetSize();

a.pop();

cout << endl << a.GetFront();

cout << endl << a.GetSize();

return 0;

}

**Priority\_queue.cpp**

#include <iostream>

using namespace std;

template<typename T>

class TPQueue

{

public:

TPQueue();

~TPQueue();

void push(T data);

void pop();

int GetSize() { return Size; }

T GetFront();

bool isEmpty() { return Size == 0; }

private:

template<typename T>

struct Node

{

Node\* pNext;

T data;

Node(T data = T(), Node\* pnext = nullptr)

{

this->data = data;

this->pNext = pnext;

}

};

int Size;

Node<T>\* head;

};

template<typename T>

TPQueue<T>::TPQueue()

{

Size = 0;

head = nullptr;

}

template<typename T>

TPQueue<T>::~TPQueue()

{

}

template<typename T>

void TPQueue<T>::push(T data)

{

if (head == nullptr)

{

head = new Node<T>(data);

}

else {

int index = 0;

Node<T>\* current = this->head;

Node<T>\* previous = this->head;

while (current->pNext != nullptr) {

if (current->data < data) {

break;

}

previous = current;

current = current->pNext;

index++;

}

if (index == 0)

{

head = new Node<T>(data, head);

Size++;

}

else

{

//Node<T>\* previous = this->head;

//for (int i = 0; i < index-1; i++)

//{

// previous = previous->pNext;

//}

Node<T>\* newNode = new Node<T>(data, previous->pNext);

previous->pNext = newNode;

Size++;

}

}

}

template<typename T>

void TPQueue<T>::pop()

{

Node<T>\* temp = head;

head = head->pNext;

//если самый первый элемент то мы сдвигаем голову на один шаг, а первый элемент удаляем

delete temp;

Size--;

}

template<typename T>

T TPQueue<T>::GetFront()

{

Node<T>\* current = this->head;

return current->data;

}

int main()

{

TPQueue<int> a;

a.push(13);

a.push(23);

a.push(33);

a.push(93);

a.push(43);

cout << a.GetFront();

a.pop();

cout << endl << a.GetFront();

a.pop();

cout << endl << a.GetFront();

a.pop();

cout << endl << a.GetFront();

a.pop();

cout << endl << a.GetFront();

return 0;

}

**Stack.cpp**

#include <iostream>

using namespace std;

template<typename T>//шаблоный класс

class TStack //создаем список с названием TStack

{

public:

TStack();//конструктор

~TStack();//деструктор

void push(T data);// добавление в конец списка

void pop();// удаление в списке по номеру

// добавление нового элемента в список

int GetSize() { return Size; }// гетер для дины(возвращаем длину)(в любой момет можем получить длину)

T GetBack(); //возврат первого элемента

bool isEmpty() { return Size == 0; } //проверка на пустоту

private:

template<typename T>// (шаблонный) чтобы небыли привязаны к определенному типу данных

struct Node

{

Node\* pNext; // указатель на следующий элемент

T data;// тип данных

Node(T data = T(), Node\* pnext = nullptr) //конструктор(данные, указатель)

{

this->data = data; //присваиваем входязие параметры

this->pNext = pnext;

}

};

int Size;// храним длину списка

Node<T>\* head; //голова нашего списка(указатель, т.к память динамическая)

};

template<typename T>

TStack<T>::TStack()//конструктор(реализация длины)

{

Size = 0;

head = nullptr;

}

template<typename T>

TStack<T>::~TStack()

{

}

template<typename T>

void TStack<T>::push(T data)

{

if (head == nullptr)//проверка головы на пустоту, для добавления

{

head = new Node<T>(data);//создание нового head

}

else {

Node<T>\* current = this->head;// если что-то есть, то будем присваивать последний адрес

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

Size++;

}

template<typename T>

void TStack<T>::pop()

{

Node<T>\* previous = this->head;

for (int i = 0; i < Size - 2; i++) {

previous = previous->pNext;

}

Node<T>\* todelete = previous->pNext;

previous->pNext = todelete->pNext;

delete todelete;

Size--;

}

template<typename T>

T TStack<T>::GetBack()

{

int counter = 0;

Node<T>\* current = this->head; // отвечает за то, где мы находимся

while (true)// пока не добрались до конца списка

{

if (counter==Size-1) {

return current->data;// нашли нужный элемент и возвращаем его

}

current = current->pNext;// спрашиваем адрес следующего элемента

counter++;

}

}

int main()

{

TStack<int> a;

a.push(13);

a.push(23);

a.push(33);

a.push(43);

cout << a.GetBack();

cout << endl << a.GetSize();

a.pop();

cout << endl << a.GetBack();

cout << endl << a.GetSize();

return 0;

}

**Вывод:** Мы изучили организацию списковых структур и построение реальных структур данных на базе списков.