**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**КАФЕДРА САПР**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: **«Представление строки символов с маркером, ее обработка»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0336 |  | Орешкин А.С. |
| Преподаватель |  | Калмычков В.А. |

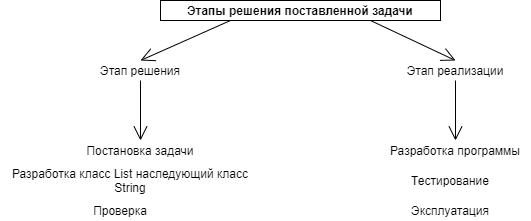
Санкт-Петербург

2021

Исходная формулировка задания

Заменить значение всех элементов с заданным значением;

1. Постановка задачи:
2. Организовать представление однонаправленного линейного списка – в элементе строка символов с маркером.
3. Обеспечить эффективное выполнение операция по преобразованию списка.
4. Реализовать чтение списка из файла любого размера.

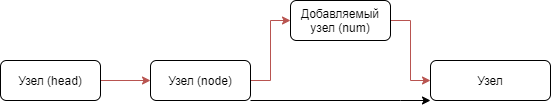


1. Анализ вариантов хранения и обработки информации.

Так как целью является реализация задачи представления однонаправленного линейного списка – в элементе строка символов с маркером, то класс List должен унаследовать класс строки (String).

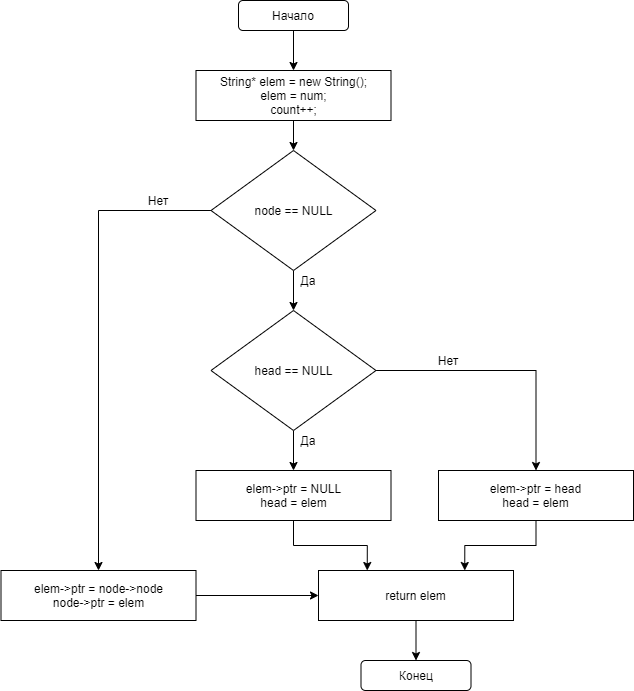
В классе списка необходимо создать указатель на строку, являющуюся началом списка. В классе строки создать указатель на следующий элемент.

1. Описание методов.
2. Конструктор класса List – корень списка приравниваем к NULL.
3. Метод int getCount() возвращает количество узлов в списке. Поле int count приравнивается к 0 при инициализации нового списка, а также при очистке списка методом Clear, инкремент переменной выполняется в методе добавления (Add) машины, декремент в методе удаления (Delete) машины.
4. Метод bool isEmpty() проверяет корневой элемент списка на пустоту - return head == NULL.
5. Метод String\* List::Add(String\* num, String\* node) добавляет узел *num* после *node* и возвращает его, если *node* будет равен NULL, то узел добавляется в начало списка.

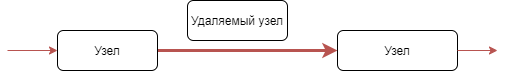


Необходимо:

* создать новый узел.
* переставить указатель узла, предшествующего добавляемому, на добавляемый узел.
* установить указатель добавляемого узла на следующий узел.

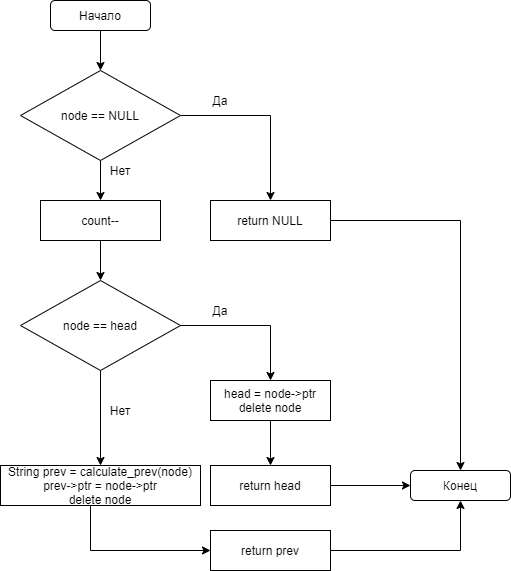


1. Метод удаления String\* List::Delete(String\* node) удаляет узел, переданный в качестве параметра из списка. Функция возвращает указатель на узел, следующий за удаляемым.



Необходимо:

* Установить указатель предыдущего узла на узел, следующий за удаляемым.
* Освобождение памяти удаляемого узла.



1. Метод void List::Print() выводит список в консоль. Для начало необходимо проверить пусти ли список, если он не пуст, то в цикле проходимся от корня списка до узла, у которого указатель на следующий элемент равен NULL.
2. Метод bool List::contains(String\* in) проверяет содержится ли в коллекции заданный узел. Для этого проверяем список на пустоту, далее проходимся от корня списка до узла, у которого указатель на следующий элемент равен NULL и проверяем каждый узел с узлом, переданным в качестве параметра на равенство.
3. Метод String\* List::next(String\* node) получает следующий узел. Необходимо проверить список на пустоту и вывести следующий элемент узла.
4. Метод String\* List::get\_node(String\* in) аналогичен методу bool List::contains(String\* in), за исключением того, что он возвращает найденный узел.
5. Метод void List::Clear() очищает список, путем удаления каждого узла.

Опишем метод List read\_strings(const char\* path), который считывает файл с множеством строк. Строки в файле должны быть разделены символом перевода строки. Необходимость в данном методе выражена в том, что метод класса String, предназначенный для чтения строки из файла, читает полностью весь файл в одну строку, данный же метод считывает каждую строку и добавляет ее в список.

1. Программа на C++:

Файл Source.cpp:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

template<typename T>

class List

{

public:

List();

~List();

void pop\_front();

void push\_back(T data);

void clear();

int getSize() { return Size; }

T& operator[](const int index);

protected:

template<typename T>

class Node {

public:

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>

List<T>::List() {

Size = 0;

head = nullptr;

}

template<typename T>

List<T>::~List() {

clear();

}

template<typename T>

void List<T>::pop\_front() {

Node<T>\* temp = head;

head = head->pNext;

delete temp;

Size--;

}

template<typename T>

void List<T>::push\_back(T data) {

if (head == nullptr) {

head = new Node<T>(data);

}

else {

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

while (current->pNext != nullptr) {

current = current->pNext;

}

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

}

Size++;

}

template<typename T>

void List<T>::clear() {

while (Size) {

pop\_front();

}

}

template<typename T>

T& List<T>::operator[](const int index) {

int counter = 0;

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

while (current != nullptr) {

if (counter == index) {

return current->data;

}

current = current->pNext;

counter++;

}

}

int main() {

List <int> list;

fstream fs;

int length = 0;

fs.open("myFile.txt", fstream::out | fstream::in);

if (!fs.is\_open()) {

cout << "Error open File!";

}

else {

cout << "Open File!" << endl;;

string z;

while (!fs.eof()) {

z = "";

getline(fs, z);

const char\* d = z.c\_str();

int res = atoi(d);

list.push\_back(res);

length++;

}

}

fs.close();

int variable = 10;

list.clear();

for (int i = 0; i < length; i++) {

list.push\_back(variable);

}

fs.open("myFile.txt", fstream::out | fstream::in);

if (!fs.is\_open()) {

cout << "Error open File!";

}

else {

cout << "Open File!" << endl;;

for (int i = 0; i < length; i++) {

fs << list[i] << '\n';

}

}

fs.close();

list.clear();

return 0;

}

1. Тестирование
2. Файл *myFile.txt*

*1*

*2*

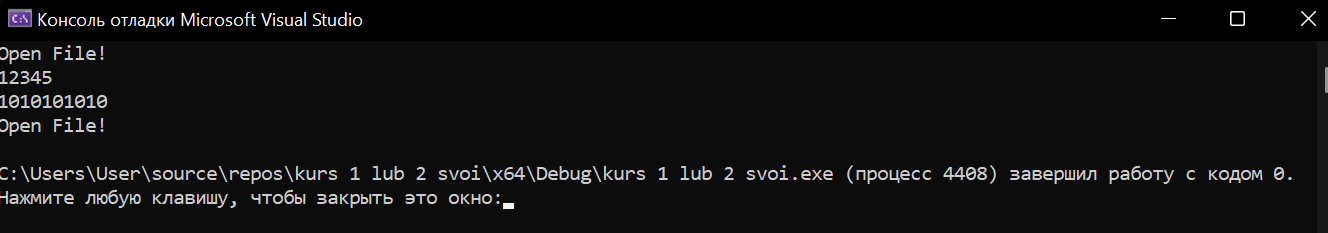
*3*

*4*

*5*

Вставим элементы в список и заменим все элементы на “10”.

Результат:



1. Вывод:
2. Организовано представление однонаправленного линейного списка – в элемент строка символов с маркером.
3. Обеспечено эффективное выполнение операций по преобразованию списка.
4. Реализовано чтение списка из файла любого размера.