**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. |

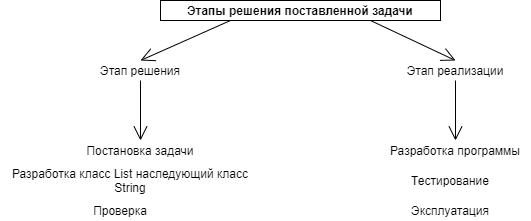
Санкт-Петербург

2021

Исходная формулировка задания

Заданное число элементов перед элементом, имеющим заданное значение;

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.

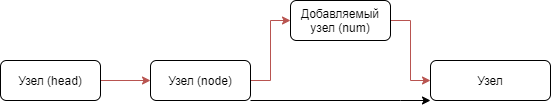


1. Анализ вариантов хранения и обработки информации.

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

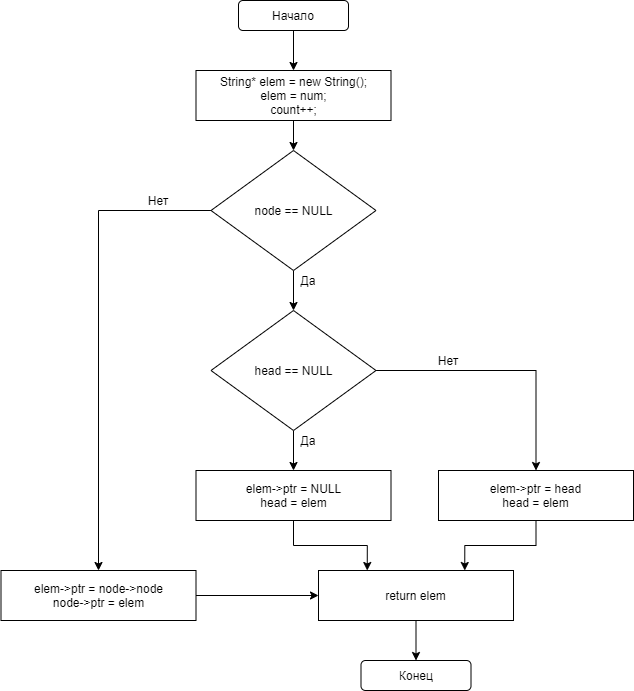
В классе списка необходимо создать указатель на строку, являющуюся началом списка. В классе строки создать указатель на следующий элемент.

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.

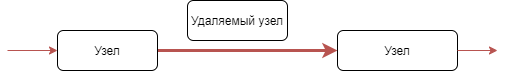


Необходимо:

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

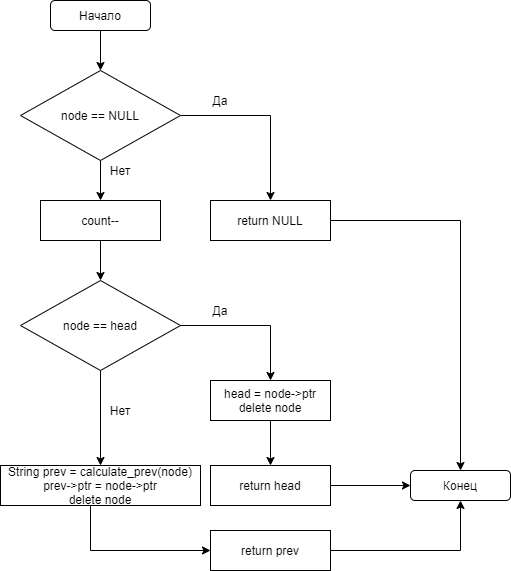


1. Метод удаления String\* List::Delete(String\* node) удаляет узел, переданный в качестве параметра из списка. Функция возвращает указатель на узел, следующий за удаляемым.



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 "List.h"

List read\_strings(const char\* path);

int main() {

system("chcp 1251");

List lst = read\_strings("file.txt");

lst.Print(); ///печатаем прочитанный список

int k = 0;

do { //кол-во вставляемх элементов

cout << "k = ";

cin >> k;

} while (k <= 0);

cout << "Вставить " << k << " элементов перед: ";

String\* in = new String();

cin.ignore();

in->input\_data();

if (lst.contains(in)) {

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

String\* node = lst.get\_node(in); //получаем узел

in = new String();

cout << "Строка: "; in->input\_data();

lst.Add(in, lst.calculate\_prev(node)); //вставляем строку, определяем предыдущий узел

}

}

lst.Print(); //печаем изменный список

lst.Clear(); //очищаем /удаляем

lst.Print(); //печаем пустой список

system("pause");

return 0;

}

//читаем файл с множеством строк, строки разделены символом перевода строки

//обычный метод чтения, нахождящийся в классе строки считывает весь файл в одну строки, поэтому

//опишем метод чтения файла построчно с сохранением в список

List read\_strings(const char\* path) {

fstream fin(path, ios::in); //открываем существующий файл

List lst;

if (fin.is\_open()) {

String\* ss = lst.getLast();

char c = lst.MARKER;

int const size = 500; //установим фиксированный размер строки

char\* str = new char[size];//инициализируем указатель на массив символов

int cur = 0; //указатель на текущий эемент

while (!fin.eof()) {

char c = fin.get();

if (c == '\n' || cur > size - 1) {

str[cur] = lst.MARKER;

String\* st = new String(str);

ss = lst.Add(st, ss);

delete[] str;

str = new char[size];

cur = 0;

}

else

str[cur++] = c;

}

delete[] str;

return lst;

}

return lst;

}

Файл String.cpp:

#include "String.h"

bool String::read\_data\_from\_file(const char\* path) { //читаем полностью файл

fstream fin(path, ios::in); //открываем существующий файл

if (fin.is\_open()) {

int size = 100; //установим фиксированный размер строки

const int add\_size = 150; //при недостатке памяти расширим ее на фиксированное кол-во символов

if (str != nullptr) {

delete[] str;

}

str = new char[size];//инициализируем указатель на массив символов

int cur = 0; //указатель на текущий эемент

while (!fin.eof()) {

char c = fin.get();

if (cur > size - 1) { //

allocate(size + add\_size);

size += add\_size; ///увелииваем текущий размер

}

str[cur++] = c;

}

if (cur != 0)

cur--;

str[cur] = MARKER; //при проверке окончания файла eof читается еще один символ, он будет являться лишним

return true;

}

return false;

}

void String::allocate(int size\_new) {

int size\_old = this->length();

char\* obj = new char[size\_new];

for (int i = 0; i < size\_old; ++i) {

obj[i] = str[i];

}

delete[] str;

this->str = obj;

}

int String::num\_substring(String& t) {

if (t.length() == 0) {

return 0;

}

const char\* begin\_str = str;

const char\* end\_str = str + strlen(str);

const char\* begin\_substr = t.str;

const char\* end\_substr = t.str + strlen(t.str);

int count = 0;

while (begin\_str != end\_str)

{

begin\_str = std::search(begin\_str, end\_str, begin\_substr, end\_substr);

if (begin\_str != end\_str)

{

++count;

begin\_str += min(begin\_str, end\_str, begin\_substr, end\_substr);

}

}

return count;

}

void String::set(const char\* str) {

if (this->str != nullptr) { //избегаем утечку памяти

delete[] this->str;

}

if (str == NULL)

return;

int len = strlen(str); //длина копируемой строки

this->str = new char[len + 1]; // устанавливаем длину на 1 больше для 0-символа

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

this->str[i] = str[i]; //копируем

}

this->str[len] = MARKER; //конец строки

}

int String::min(const char\* b1, const char\* e1, const char\* b2, const char\* e2)

{

return std::min(std::distance(b1, e1), std::distance(b2, e2));

}

String::String() { //конструктор по умолчанию

str = nullptr;

}

String::String(const char\* str) { //конструктор с параметрами

set(str);

}

String::String(const String& other) { //конструктор копирования

set(other.str);

}

void String::set(const String& other) {

set(other.str);

}

String::~String() { //деконструктор

delete[] this->str;

}

String& String::operator =(const String& other) { //перегруженная операция копирования одной строки

//в другую

set(other.str);

return \*this;

}

ostream& operator<<(ostream& out, const String& obj) {

if (obj.str != nullptr)

out << obj.str;

return out;

}

void String::input\_data() {

int size = 40; //установим фиксированный размер строки

const int add\_size = 20; //при недостатке памяти расширим ее на фиксированное кол-во символов

str = new char[size];//инициализируем указатель на массив символов

int cur = 0; //указатель на текущий эдемент

while (true) {

char c = cin.get();

if (c == '\n')

break;

if (cur > size - 1) { //

allocate(size + add\_size);

size += add\_size; ///увелииваем текущий размер

}

str[cur++] = c;

}

str[cur] = MARKER;

}

int String::length() { //длина строки

if (str != nullptr)

return strlen(str);

return 0;

}

bool operator== (const String& c1, const String& c2) {

if (strcmp(c1.str, c2.str) == 0)

return true;

return false;

}

bool operator!= (const String& c1, const String& c2) {

return !(c1 == c2);

}

Файл String.h:

#pragma once

#include <algorithm>

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class String

{

int min(const char\* b1, const char\* e1, const char\* b2, const char\* e2);

void allocate(int size);

char MARKER = '\0';

public:

char\* str;

String\* ptr;

void set(const char\* str);

void set(const String& other);

String();

String(const char\* str);

String(const String& other);

~String();

String& operator =(const String& other);

int length();

int num\_substring(String& t);

void input\_data();

bool read\_data\_from\_file(const char\* path);

friend ostream& operator<<(ostream& out, const String& obj);

friend bool operator== (const String& c1, const String& c2);

friend bool operator!= (const String& c1, const String& c2);

};

Файл List.cpp:

#include "List.h"

String\* List::Delete(String\* node) {

if (node == NULL) { return NULL; } // В списке нет узлов

count--;

if (node == head) // Удаление корневого узла

{

head = node->ptr;

delete node;

return head;

}

String\* prev = calculate\_prev(node); // Удаление промежуточного узла

prev->ptr = node->ptr;

delete node;

return prev;

}

String\* List::getLast() { //полечение последнеего узла

String\* p = head;

while (next(p) != NULL)

p = next(p);

return p;

}

String\* List::Add(String\* num, String\* node) { //добавление num после node

String\* elem = new String();

elem = num;

count++;

if (node == NULL) // Добавление нового корня

{

if (head == NULL) {

elem->ptr = NULL;

head = elem;

}

else {

elem->ptr = head;

head = elem;

}

return elem;

}

elem->ptr = node->ptr; // Добавление узла после текущего

node->ptr = elem;

return elem;

}

String\* List::next(String\* node)//полчение следующего узла

{

if (isEmpty()) return NULL;

return node->ptr;

}

String\* List::get\_node(String\* in) { //получить узел с нужным нам значением строки

if (isEmpty()) {

return NULL;

}

String\* p = head;

do {

if (\*in == \*p) {

return p;

}

p = next(p);

} while (p != NULL);

return nullptr;

}

String\* List::calculate\_prev(String\* node) //определить предыдущий узхел данного

{

if (isEmpty())

return NULL;

if (node == head)

return NULL;

String\* p = head;

while (p->ptr != node)

p = p->ptr;

return p;

}

void List::Clear() //очистить список

{

String\* p = head;

if (p == NULL) return;

do {

String\* d = p;

p = next(p);

delete d;

} while (p != NULL);

count = 0;

head = NULL;

}

void List::Print() //вывести

{

if (isEmpty()) {

cout << "Список пуст" << endl;

return;

}

String\* p = head;

do {

cout << \*p << endl;

p = next(p);

} while (p != NULL);

}

bool List::contains(String\* in) { //определяем содержится ли в коллеккции данная строка

if (isEmpty()) {

return false;

}

String\* p = head;

do {

if (\*in == \*p) {

return true;

}

p = next(p);

} while (p != NULL);

return false;

}

Файл List.h:

#pragma once

#include "String.h"

class List : public String

{

String\* head;

int count;

public:

char MARKER = '\0';

List() { head = NULL; } // Инициализация списка

int getCount() { return count; } // Получение количества узлов списка

bool isEmpty() { return head == NULL; } // Проверка, пуст ли список

String\* getFirst() { return head; }

void Clear(); // Очистка списка

String\* next(String\*);

String\* Add(String\* num, String\* node = NULL);

String\* Delete(String\* node);

bool contains(String\*);

void Print();

String\* calculate\_prev(String\* node);

String\* get\_node(String\* in);

String\* getLast();

};

1. Тестирование
2. Файл *file.txt*

*string1*

*hell*

*aaa*

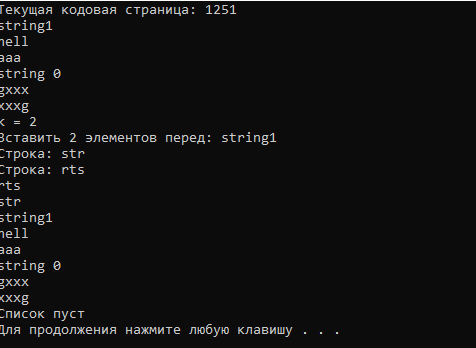
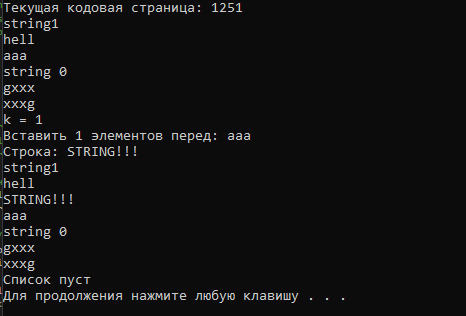
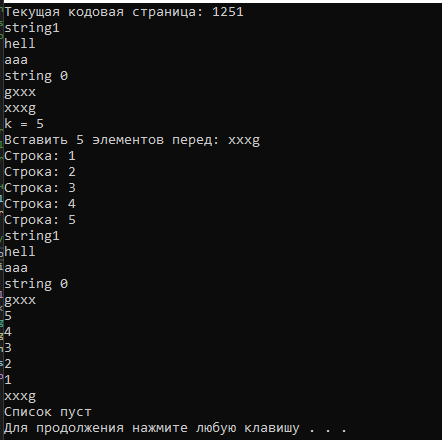
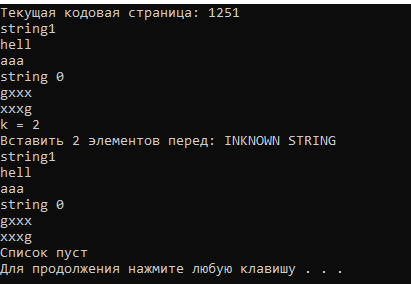
*string 0*

*gxxx*

*xxxg*

Вставим элементы в 1) начало списка, 2) в середину и 3) в конец. Также проверим, можно ли 4) вставить элементы перед несуществующим узлом.

Результат:

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