

Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ИСПОЛЬЗОВАНИЕ УКАЗАТЕЛЕЙ В C++.

Лабораторная работа № 2
по курсу «Объектно-ориентированное программирование»

Вариант № 11

| | | | |
|-----------|------------------------------|-----------|--------------|
| Выполнил: | студент группы 220601 | _____ | Белым А.А. |
| | | (подпись) | |
| Проверил: | к. ф.-м. н., доцент каф. АТМ | _____ | Середин О.С. |
| | | (подпись) | |

Тула 2013

Цель работы

Научиться использовать в программах на C++ указатели. Написать программу, выполняющую определенные действия с указателями.

Задание

Используя списки, преобразовать введенный текст следующим образом. Если строка начинается с цифры, то перед ней поставить левую квадратную скобку, а в конец строки – правую квадратную скобку.

Теоретическая справка

[illegible]

[illegible]

Реализация класса

Ниже представлено определение класса, файл List.h:

```
#ifndef LIST_H
#define LIST_H
/*!Класс "односвязный список".*/
class List {
private:
    /*!Тип "элемент списка".*/
    struct Elem{
        char c; /*!<значение элемента списка
        Elem* next;/*!<указатель на следующий элемент
    };
    Elem *first,/*!<указатель на первый элемент списка
        *last;/*!<указатель на последний элемент списка
    mutable Elem *current;/*!<указатель на текущий элемент списка
public:
    /*!Конструктор по умолчанию.*/
    List();
    /*!Конструктор по ASCIIZ строке.
    * \param source исходная строка
    */
    List(const char* source);
    /*!Конструктор копирования.
    * \param other копируемый список
    */
    List(const List& other);
    /*!Деструктор класса.*/
    ~List();
    /*!Установка значения по ASCIIZ строке.
    * \param source исходная строка
    */
    void FromChar(const char* source);
    /*!Метод добавления символа в конец списка.
    * \param new_char добавляемый символ
    */
    void Append(char new_char);
    /*!Метод добавления символа в начало списка.
    * \param new_char добавляемый символ
    */
    void Prepend(char new_char);
    /*!Вывод списка на экран.
    */
    void Print() const;
    /*!Изменяет позицию в списке с текущего элемента
    * на следующий за ним.
    */
    void Next() const;
    /*!Изменяет позицию в списке с текущего элемента
    * на начальный элемент.
    */
    void Reset() const;
    /*!Метод получения значения текущего элемента.
    * \return значение текущего элемента.
    */
    char Value() const;
    /*!Проверяет, достигнут ли конец списка.
    */
    bool isEnd() const;
    /*!Удаляет текущий элемент.
    * Текущим становится следующий после удаляемого
    * элемент.
    */
    void Delete();
    /*!Удаляет следующий после текущего элемент.*/
```

```

void DeleteNext();
/*!Вставляет новый элемент на место текущего.
 * Текущий элемент становится следующим после вставленного.
 * \param с добавляемый символ
 */
void Insert(char c);
/*!Вставляет новый элемент после текущего.
 * \param с добавляемый символ
 */
void InsertNext(char c);
/*!Очистка списка, удаление всех содержащихся в списке элементов.
 */
void Clear();
/*!Копирование другого списка.
 * \param other копируемый список
 */
void Copy(const List&);
/*!Перегрузка оператора присваивания.
 * \param other присваиваемый список
 * \return данный список
 */
List& operator=(const List& other);
};

#endif //LIST_H

```

Ниже представлена реализация класса, файл List.cpp:

```

#include "List.h"
#include "cstdlib"
#include <iostream>
using namespace std;
List::List():first(NULL),
              last(NULL),
              current(NULL)
{
}

List::List(const char* source):first(NULL),
                               last(NULL),
                               current(NULL)
{
    FromChar(source);
}

void List::Next() const{
    if(current&&current->next)
        current=current->next;
}

char List::Value() const{
    if(current)
        return current->c;
    else
        return '\0';
}

void List::Reset() const{
    current=first;
}

bool List::isEnd() const{
    return current==last||current->next==NULL||current==NULL;
}

void List::Append(char new_char){

```

```

Elem* new_elem=(Elem*)malloc(sizeof(Elem));
if(new_elem==NULL){
    cerr<<"Error of memory allocation for new element!"<<endl;
    return;
}
if(last!=NULL)
    last->next=new_elem;
else {
    first=new_elem;
    current=new_elem;
}
last=new_elem;;
last->next=NULL;
last->c=new_char;
}

void List::Prepend(char new_char){
    Elem* new_elem=(Elem*)malloc(sizeof(Elem));
    if(new_elem==NULL){
        cerr<<"Error of memory allocation for new element!"<<endl;
        return;
    }
    if(first==NULL){
        last=new_elem;
        current=new_elem;
    };
    new_elem->next=first;
    first=new_elem;
    first->c=new_char;
}

void List::FromChar(const char* source){
    const char* ptr=source;
    for(ptr;*ptr;ptr++){
        Append(*ptr);
    }
}

void List::Clear(){
    current=first;
    if(current==NULL);
        return;
    Elem *prev=current;
    while(!isEnd()){
        Next();
        free(prev);
        prev=current;
    }
    free(prev);
    current=NULL;
    first=NULL;
    last=NULL;
}

List::~~List(){
    Clear();
}

void List::Print() const{
    Elem* backup=current;
    Reset();
    while(!isEnd()){
        std::cout<<Value();
        Next();
    }
    std::cout<<Value()<<std::endl;
    current=backup;
}

```

```

void List::Delete() {
    if(!current)
        return;
    Elem* tmp_next=current->next;
    if(tmp_next){
        current->c=current->next->c;
        current->next=current->next->next;
        free(tmp_next);
    } else {
        if(current==first) {
            first=NULL;
            last=NULL;
            current=NULL;
            free(tmp_next);
        } else {
            Reset();
            while(current->next&&current->next->next)
                Next();
            current->next=NULL;
            last=current;
            free(tmp_next);
        }
    }
}

void List::DeleteNext() {
    if(!current)
        return;
    Elem* tmp_next=current->next;
    if(tmp_next){
        current->next=current->next->next;
        free(tmp_next);
    } else
        current->next=NULL;
}

void List::Insert(char c){
    if(current==NULL)
        Prepend(c);
    Elem *new_elem=(Elem*)malloc(sizeof(Elem));
    if(new_elem==NULL)
        std::cout<<"malloc error"<<std::endl;
    new_elem->c=current->c;
    new_elem->next=current->next;
    current->c=c;
    current->next=new_elem;
    if(current==last)
        last=new_elem;
}

void List::InsertNext(char c){
    if(current==NULL||current==last)
        Append(c);
    Elem *new_elem=(Elem*)malloc(sizeof(Elem));
    if(new_elem==NULL)
        return;
    new_elem->c=c;
    new_elem->next=current->next;
    current->next=new_elem;
}

void List::Copy(const List& other){
    if(this==&other)
        return;
    Clear();
    Elem* backup=other.current;

```

```

        other.Reset();
        while(!other.isEnd()){
            Append(other.Value());
            other.Next();
        }
        Append(other.Value());
        other.current=backup;
    };

List::List(const List& other):first(NULL),
                                last(NULL),
                                current(NULL)
{
    Copy(other);
}

List& List::operator=(const List& other){
    Copy(other);
    return *this;
}

```

Демонстрационная программа

Далее приводится демонстрационная программа, файл lab2.cpp:

```

#include "List.hpp"
#include <iostream>
#include <string>
#include <cctype>

int main(){
    std::string str;
    std::cout<<"Введите любую строку\n";
    std::cin>>str;

    List a(str.c_str());

    if(isdigit(a.Value())){
        a.Prepend('[');
        a.Append(']');
    };
    a.Print();
    return 0;
}

```

Инструкция программисту

Далее приводится описание функций, методов, типов данных и классов.

Структура List::Elem

Тип "элемент списка".

Открытые атрибуты

- char c
значение элемента списка
- Elem * next
указатель на следующий элемент

Класс List

Класс "односвязный список".

```
#include <List.h>
```

Классы

- struct **Elem**

Открытые члены

- **List** ()
- **List** (const char *source)
- **List** (const **List** &other)
- **~List** ()
- void **FromChar** (const char *source)
- void **Append** (char new_char)
- void **Prepend** (char new_char)
- void **Print** () const
- void **Next** () const
- void **Reset** () const
- char **Value** () const
- bool **isEnd** () const
- void **Delete** ()
- void **DeleteNext** ()
- void **Insert** (char c)
- void **InsertNext** (char c)
- void **Clear** ()
- void **Copy** (const **List** &)
- **List** & **operator=** (const **List** &other)

Закрытые данные

- **Elem * first**
указатель на первый элемент списка
- **Elem * last**
указатель на последний элемент списка
- **Elem * current**
указатель на текущий элемент списка

Конструкторы

List::List ()

Конструктор по умолчанию.

List::List (const char * *source*)

Конструктор по ASCIIZ строке.

Аргументы конструктора представлены в таблице 1.

Аргументы:

Таблица 1 - Аргументы конструктора char*

| | |
|---------------|-----------------|
| <i>source</i> | исходная строка |
|---------------|-----------------|

List::List (const List & *other*)

Конструктор копирования.

Аргументы конструктора представлены в таблице 2.

Аргументы:

Таблица 2 - Аргументы конструктора копирования

| | |
|--------------|-------------------|
| <i>other</i> | копируемый список |
|--------------|-------------------|

List::~~List ()

Деструктор класса.

Методы

void List::Append (char *new_char*)

Метод добавления символа в конец списка.

Аргументы метода представлены в таблице 3.

Аргументы:

Таблица 3 – Аргументы метода добавления в конец списка

| | |
|-----------------|--------------------|
| <i>new_char</i> | добавляемый символ |
|-----------------|--------------------|

void List::Clear ()

Очистка списка, удаление всех содержащихся в списке элементов.

void List::Copy (const List & *other*)

Копирование другого списка.

Аргументы метода представлены в таблице 4.

Аргументы:

Таблица 4 – Аргументы метода копирования списка

| | |
|--------------|-------------------|
| <i>other</i> | копируемый список |
|--------------|-------------------|

void List::Delete ()

Удаляет текущий элемент. Текущим становится следующий после удаляемого элемент.

void List::DeleteNext ()

Удаляет следующий после текущего элемент.

void List::FromChar (const char * *source*)

Установка значения по ASCIIZ строке.

Аргументы метода представлены в таблице 5.

Аргументы:

Таблица 5 - Аргументы метода установки значения

| | |
|---------------|-----------------|
| <i>source</i> | исходная строка |
|---------------|-----------------|

void List::Insert (char *c*)

Вставляет новый элемент на место текущего. Текущий элемент становится следующим после вставленного.

Аргументы метода представлены в таблице 6.

Аргументы:

Таблица 6 - Аргументы метода вставки

| | |
|----------|--------------------|
| <i>c</i> | вставляемый символ |
|----------|--------------------|

void List::InsertNext (char *c*)

Вставляет новый элемент после текущего.

Аргументы метода представлены в таблице 7.

Аргументы:

Таблица 7 - Аргументы метода вставки после текущего

| | |
|----------|--------------------|
| <i>c</i> | вставляемый символ |
|----------|--------------------|

bool List::isEnd ()

Проверяет, достигнут ли конец списка.

void List::Next ()

Изменяет позицию в списке с текущего элемента на следующий за ним.

void List::Prepend (char *new_char*)

Метод добавления символа в начало списка.

Аргументы метода представлены в таблице 8.

Аргументы:

Таблица 8 - Аргументы метода добавления в начало списка

| | |
|-----------------|--------------------|
| <i>new_char</i> | добавляемый символ |
|-----------------|--------------------|

void List::Print ()

Вывод списка на экран.

void List::Reset ()

Изменяет позицию в списке с текущего элемента на начальный элемент.

char List::Value ()

Метод получения значения текущего элемента.

Возвращает:

значение текущего элемента.

List & List::operator= (const List & *other*)

Перегрузка оператора присваивания.

Аргументы оператора представлены в таблице 9.

Аргументы:

Таблица 9 - Аргументы оператора присваивания

| | |
|--------------|----------------------|
| <i>other</i> | присваиваемый список |
|--------------|----------------------|

Возвращает:

данный список.

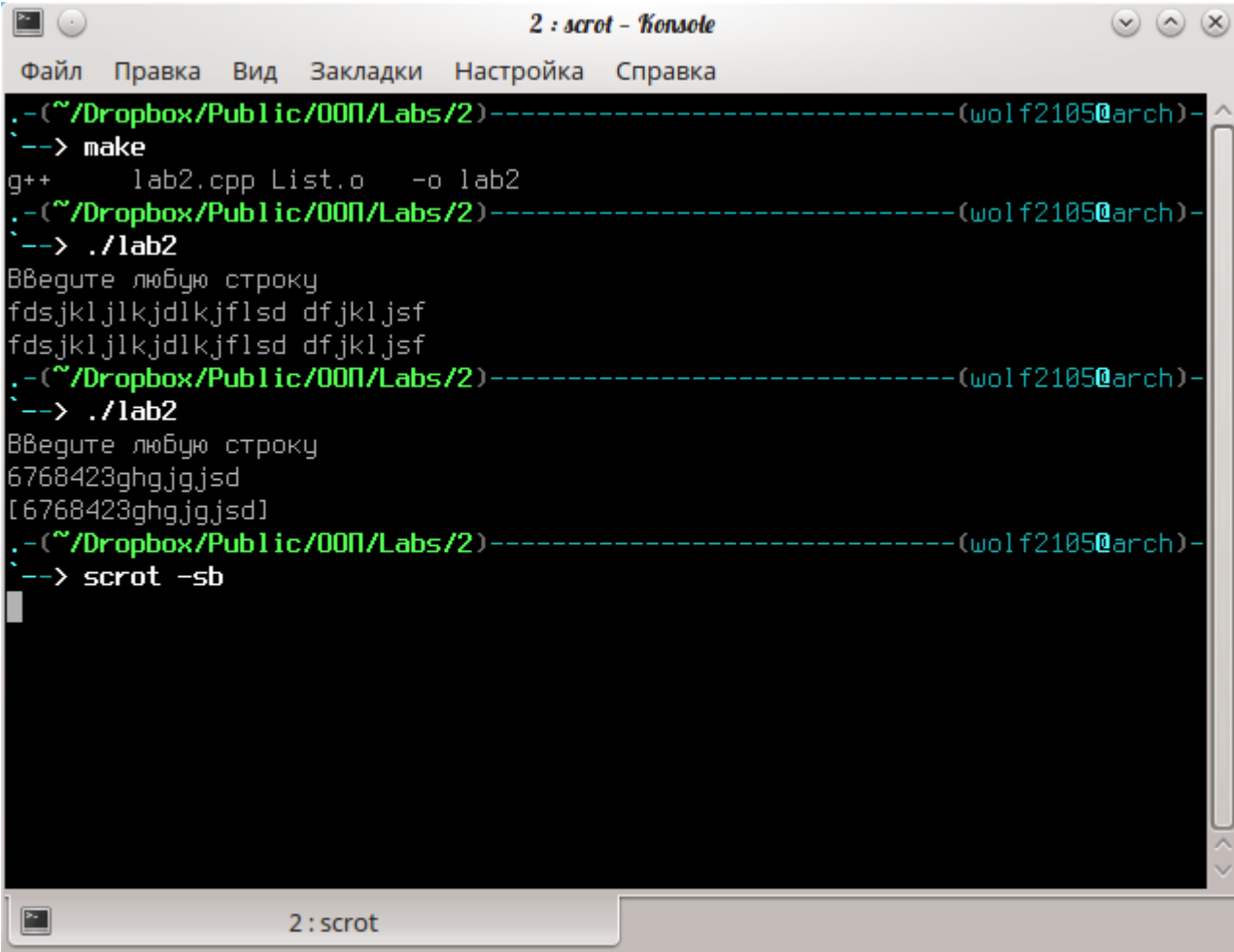
Инструкция пользователю

Данная программа заключает строку в квадратные скобки, если она начинается с цифры.

Для работы программы введите произвольную строку. Если строка начинается с цифры, то она выводится заключенной в квадратные скобки; иначе она будет выведена без изменений.

Контрольный пример

На рисунке 1 представлен пример работы программы, работающей со списком.



```
2 : scrot - Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка

~/.Dropbox/Public/ООП/Labs/2)----- (wolf2105@arch)-
--> make
g++ lab2.cpp List.o -o lab2
~/.Dropbox/Public/ООП/Labs/2)----- (wolf2105@arch)-
--> ./lab2
Введите любую строку
fdsjklj1kjd1kjflsd dfjkljsf
fdsjklj1kjd1kjflsd dfjkljsf
~/.Dropbox/Public/ООП/Labs/2)----- (wolf2105@arch)-
--> ./lab2
Введите любую строку
6768423ghg.jg.jsd
[6768423ghg.jg.jsd]
~/.Dropbox/Public/ООП/Labs/2)----- (wolf2105@arch)-
--> scrot -sb
```

Рисунок 1— Пример работы программы, работающей со списком

Вывод

В данной лабораторной работе я изучил работу с указателями в языке Си++. Была реализована программа, использующая класс «связный список», который использует указатели и динамическую память для управления своими элементами.