МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического занятия 6

Тема: Однонаправленный динамический список

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Хвостов В. В.

Группа ИКБО-01-20

Москва 2021

Содержание

Ответы на вопросы

1. Существует три уровня представления данных: уровень пользователя (предметная область), логический и физический.

   Каждый объект предметной области характеризуется своими атрибутами, каждый атрибут имеет имя и значение.

   Логический (концептуальный) уровень - это абстрактное представление (абстрактный уровень) данных, независимое от представления в ЭВМ.

   Физический уровень - это практическая реализация базы данных на том или ином носителе в ЭВМ. Сюда входят и программные средства управления этими носителями.

2. Тип данных определяет то, чем именно представляются данные, операции над ними, а также способы хранения значений типа.

3. Структура данных определяет множество связей между ними, а также способ их организации.

4. Структура данных — программная единица, позволяющая хранить и обрабатывать множество однотипных или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

5. Линейная структура данных - это такая структура данных, в которой операции осуществляются линейно. ps: https://askinglot.com/what-is-linear-data-structure-explain-with-example

6. Линейный однонаправленный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей.

7. Стек - это линейная структура данных, в которой операции получения и удаления элемента структуры осуществляется в определенном порядке LIFO - новый элемент стека становится последним и он же становится следующим удаляемым. Позиция данного элемента называется вершиной стека.

8. Очередь - это линейная структура данных, в которой операции получения и удаления элемента структуры осуществляется в определенном порядке FIFO - новый элемент стека становится последним, первый добавленный элемент является следующим удаляемым.

9. Стек может быть построен на списке, но список не всегда будет стеком т.к. в стеке соблюдается специальный порядок обработки данных LIFO (был рассмотрен выше).

10. Двунаправленный список

11. Сложность вставки в произвольную позицию массива размера n равна $O(n)$ т.к. сама вставка занимает $O(1)$ + сдвиг элементов массива в худшем случае $O(n)$.

    Сложность вставки в произвольную позицию списка происходит за $O(1)$ + получение адреса элемента списка в худшем случае $O(n)$. Отсюда сложность $O(n)$.

12. Удаление элемента из массива и из списка имеет аналогичное вставке поведение, т.е. сложность $O(n)$.

13. Трюк Вирта заключается в проходе списка при помощи одного указателя. Можно хранить XOR двух указателей, на предыдущий элемент списка и на следующий, таким образом имея оба под рукой в одном указателе.

14. template<typename T> class Node T value; Node* next;;

15. Приведен ниже в отчете главного задания.

16. Код предоставлен таккже в отчете главного задания.

17. В этом коде лишней является ветка условного оператора с проверкой на нулевой указатель т.к. при обращении по нулевому указателю произойдет ошибка программы. Код вставляет новый узел в последующий после LL узел.

Главное задание

2.1. Постановка задачи

Требуется реализовать программу решения следующих задач варианта No9 по использованию линейного однонаправленного списка:

1. Информационная часть узла содержит символы, которые формируют "слова", разделенные пробелом.
2. Разработать функцию для создания исходного списка, используя функцию вставки нового узла перед первым узлом.
3. Разработать функцию вывода списка.
4. Разработать функцию, которая находит последнее слово и переставляет его в начало списка.
5. Разработать функцию, которая удаляет второе слово.
6. Разработать функцию, которая заменяет k-ое слово на новое слово. Длина нового слова может быть больше длины k-ого слова.
7. В основной программе выполнить тестирование каждой функции.

2.2. Определение операций над списком

Структура узла линейного списка

Согласно варианту No9 в качестве информационной части узла списка используются символы. Класс узла хранит в себе значение данного узла и указатель на следующий элемент в списке. Сам список хранит в себе корень списка, через который происходит доступ к остальным узлам списка, а также размер данного списка (количество узлов).

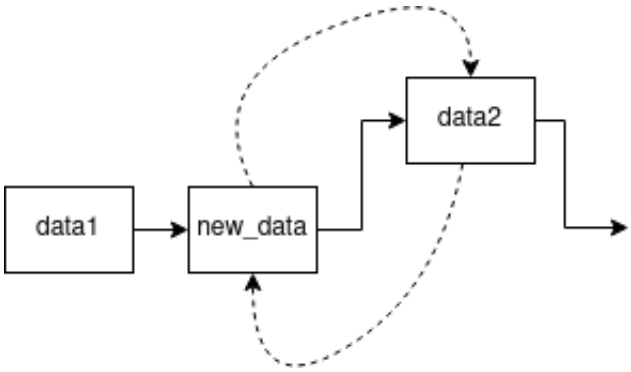

Рис. 1 - Изображение вставки узла перед элементом (перед data2)

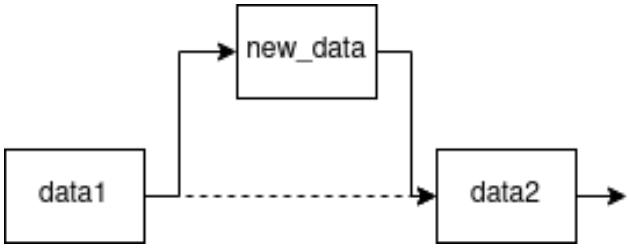

Рис. 2 - Изображение вставки узла после элемента (после data1)

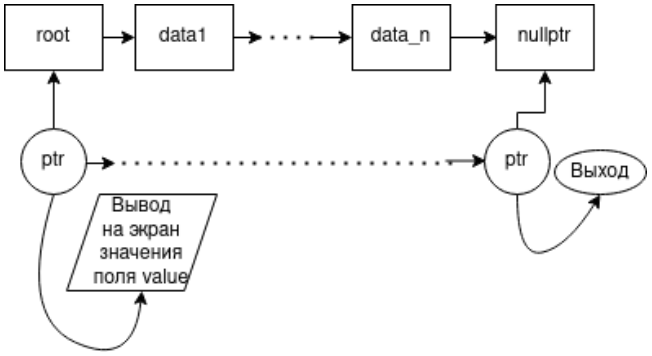

Рис. 3 - Алгоритм вывода элементов списка

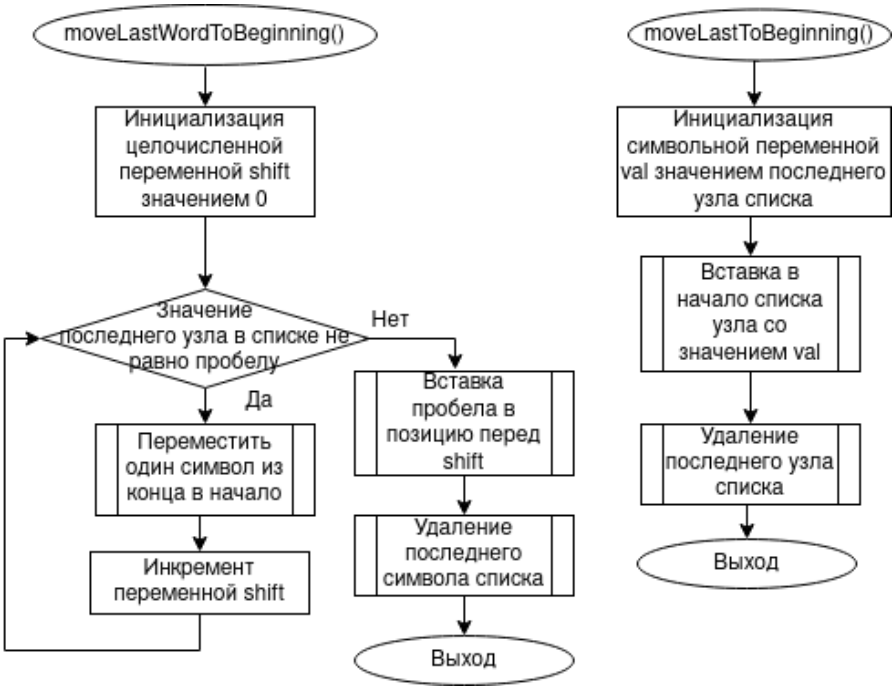

Рис. 4 - Алгоритм переставления слова в начало списка

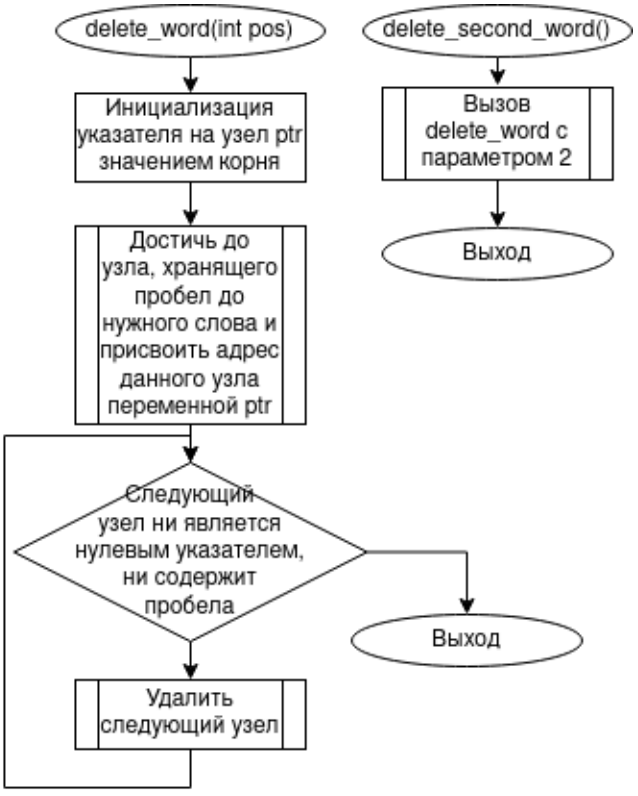

Рис. 5 - Алгоритм удаления второго слова

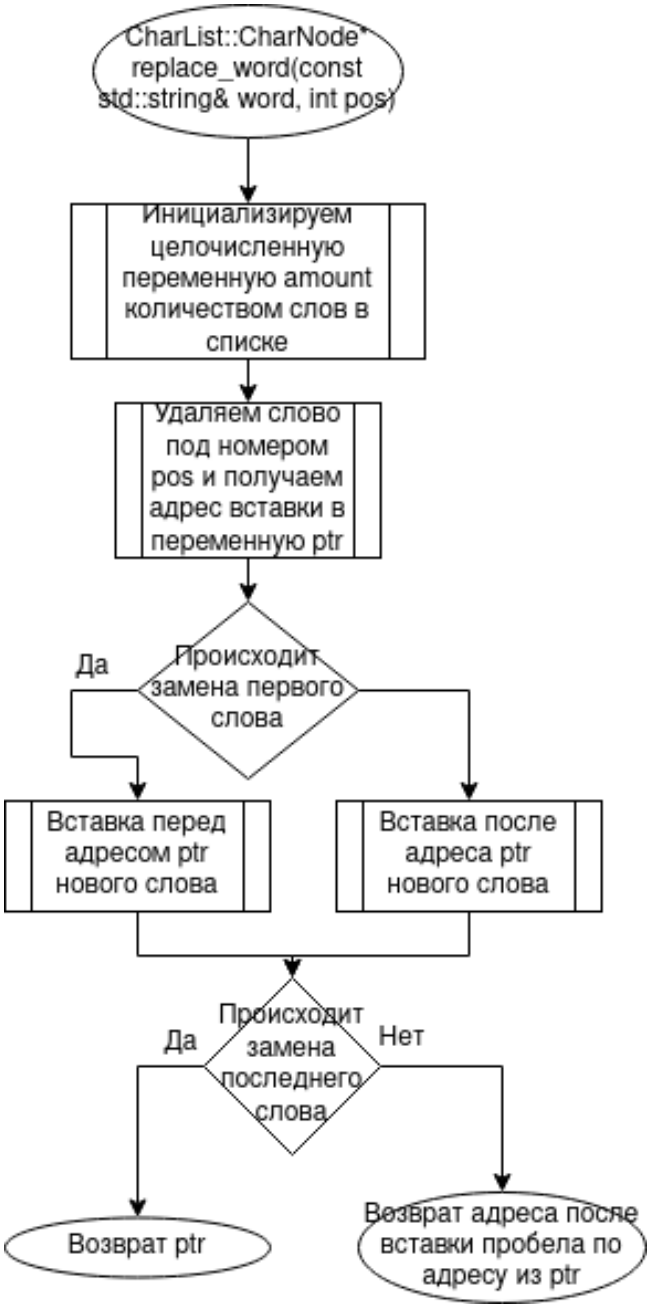

Рис. 6 - Алгоритм замены любого слова в списке

2.3. Код программы

Реализация на C++:

Код файла charlist.h

```
#include <string>

class CharList {
public:
  struct CharNode {
      char value;
      CharNode* next;
    };

  CharList();
  CharList(char new_value);
  CharList(const std::string &word);

 ~CharList();

 CharNode* insert_after(char new_value, int pos);
 CharNode* insert_after(char new_value, int pos, int count);
 CharNode* insert_after(char new_value, CharNode* ptr);
 CharNode* insert_before(char new_value, int pos);
 CharNode* insert_before(char new_value, CharNode* ptr);
 void push_back(char new_value);

 CharNode* insert_word_after(const std::string &word, int pos);
 CharNode* insert_word_after(const std::string &word, CharNode* p);
 CharNode* insert_word_before(const std::string &word, int pos);
 CharNode* insert_word_before(const std::string &word, CharNode* p);
 void push_back(const std::string &word);

 CharNode* delete_after(int pos);
 CharNode* delete_after(CharNode* ptr);
 void pop_front();

 CharNode* delete_word(int word_pos);
 CharNode* delete_second_word();

 CharNode* replace_word(const std::string &word, int pos);

 void moveLastToBeginning();
 void moveLastWordToBeginning();

 void printList() const;

 inline size_t getSize() const {
   return size;
```

```
    }

    int amountOfWords();
private:
    CharNode* root = nullptr;
    size_t size = 0;

    CharNode* end() const;
};
```

Код файла charlist.cpp

```
// I think I need to change the structure (add the element before the first
// to use my operations)
#include <iostream>
#include <string>

#include "charlist.h"

CharList::CharList() = default;

CharList::CharList(char new_value) {
    insert_before(new_value, 0);
}

CharList::CharList(const std::string &word) {
    insert_word_before(word, 0);
}

CharList::~CharList() {
    for (; root != nullptr;) {
        CharNode* ptr = root;
        root = root->next;
        delete ptr;
    }
}

CharList::CharNode* CharList::insert_after(char new_value, int pos) {
    CharNode* ptr = root;
    for (int i = 0; i < pos && ptr->next != nullptr; ++i, ptr = ptr->next) {
    }
    CharNode* node = new CharNode();
    ++size;
    node->value = new_value;
    node->next = ptr->next;
    ptr->next = node;
    return node;
}
```

```
CharList::CharNode* CharList::  insert_after (char new_value, CharNode* ptr) {
  CharNode* node = new CharNode();
  ++size;
  node->value = new_value;
  node->next = ptr->next;
  ptr->next = node;
  return node;
}

CharList::CharNode* CharList::  insert_after (char new_value, int pos, int count) {
  CharNode* node;
  for (int i = 0; i < count; ++i) {
    node =  insert_after (new_value, pos+i);
  }
  return node;
}

CharList::CharNode* CharList::  insert_before (char new_value, int pos) { // before nullptr ?
  CharNode* ptr = root;
  for (int i = 0; i < pos && ptr->next != nullptr; ++i, ptr = ptr->next) {
  }
  CharNode* node = new CharNode();
  ++size;
  if (ptr != nullptr) {
    node->value = ptr->value;
    node->next = ptr->next;
    ptr->value = new_value;
    ptr->next = node;
    return ptr;
  } else {
    root = node;
    node->value = new_value;
    node->next = nullptr;
    return node;
  }
}

CharList::CharNode* CharList::  insert_before (char new_value, CharNode* ptr) {
  CharNode* node = new CharNode();
  ++size;
  if (ptr != nullptr) {
    node->value = ptr->value;
    node->next = ptr->next;
    ptr->value = new_value;
    ptr->next = node;
    return ptr;
  } else {
    root = node;
```

```
    node->value = new_value;
    node->next = nullptr;
    return node;
  }
}
void CharList::push_back(char new_value) {
  insert_after(new_value, end());
}

CharList::CharNode* CharList::insert_word_after(const std::string &word, int pos) {
  CharNode* node;
  for (int i = 0; i < static_cast<int>(word.size()); ++i) {
    node = insert_after(word[i], pos+i);
  }
  return node;
}

CharList::CharNode* CharList::insert_word_after(const std::string &word, CharNode* p) {
  CharNode* node = p;
  for (int i = 0; i < static_cast<int>(word.size()); ++i) {
    node = insert_after(word[i], node);
  }
  return node;
}

void CharList::push_back(const std::string &word) {
  insert_word_after(word, end());
}

CharList::CharNode* CharList::insert_word_before(const std::string &word, int pos) {
  CharNode* node;
  node = insert_before(word[0], pos);
  for (int i = 1; i < static_cast<int>(word.size()); ++i) {
    node = insert_after(word[i], pos+i);
  }
  return node;
}

CharList::CharNode* CharList::insert_word_before(const std::string &word, CharNode* p) {
  CharNode* node = p;
  node = insert_before(word[0], p);
  for (int i = 1; i < static_cast<int>(word.size()); ++i) {
    node = insert_after(word[i], node);
  }
  return node;
}

CharList::CharNode* CharList::delete_after(int pos) {
```

```
  CharNode* ptr = root;
  for (int i = 0; i < pos && ptr->next != nullptr; ++i, ptr = ptr->next);
  if (ptr->next != nullptr) {
    CharNode* p = ptr->next;
    ptr->next = p->next;
    delete p;
    --size;
  }
  return ptr;
}

CharList::CharNode* CharList::delete_after(CharNode* ptr) {
  if (ptr->next != nullptr) {
    CharNode* p = ptr->next;
    ptr->next = p->next;
    delete p;
    --size;
  }
  return ptr;
}

void CharList::pop_front() {
  if (root->next != nullptr) {
    CharNode* ptr = root;
    root = root->next;
    delete ptr;
  }
}

CharList::CharNode* CharList::delete_word(int word_pos) {
  CharNode* ptr = root;
  for (int i = 1; i < word_pos; ++i) {
    for (; ptr->value != ' '; ptr = ptr->next);
    if (ptr->next != nullptr && i != word_pos-1) {
      ptr = ptr->next;
    }
  }
  if (word_pos == 1) {
    for (; root->value != ' ';) {
      this->pop_front();
    }
    this->pop_front();
    return root;
  }
  for (; ptr->next != nullptr && ptr->next->value != ' ';) {
    delete_after(ptr);
  }
  if (ptr->next != nullptr) {
```

```
        delete_after (ptr);
    }
    return ptr;
}

CharList::CharNode* CharList::delete_second_word() {
    return delete_word(2);
}

CharList::CharNode* CharList::replace_word(const std::string &word, int pos) {
    int amount = amountOfWords();
    CharNode* ptr = delete_word(pos);
    if (pos == 1) {
        ptr = insert_word_before(word, ptr);
    } else {
        ptr = insert_word_after(word, ptr);
    }
    if (pos == amount) {
        return ptr;
    }
    return insert_after(' ', ptr);
}

// Amount of the spaces + 1 = amount of the words in the list
int CharList::amountOfWords() {
    int amount = 0;
    for (CharNode* ptr = root; ptr != nullptr; ptr = ptr->next) {
        if (ptr->value == ' ') {
            ++amount;
        }
    }
    return amount + 1;
}

void CharList::moveLastToBeginning() {
    CharNode* ptr = end();
    char val = ptr->value;
    delete_after(size-2);
    insert_before(val, 0);
}

CharList::CharNode* CharList::end() const {
    CharNode* ptr = root;
    for (; ptr->next != nullptr; ptr = ptr->next);
    return ptr;
}

// can move with only swapping values?
```

```
void CharList::moveLastWordToBeginning() {
  int shift = 0;
  for (;end()->value != ' '; ++shift) {
    moveLastToBeginning();
  }
   insert_before(' ', shift);
   delete_after(size-2);
}

void CharList::printList() const {
  CharNode* ptr = root;
  for (; ptr != nullptr; ptr = ptr->next) {
    if (ptr->value == ' ') {
      std::cout << '_';
    } else {
      std::cout << ptr->value;
    }
  }
}
```

Код файла list_lin.cpp

```
#include <algorithm>
#include <iostream>
#include <random>
#include <string>
#include <string_view>

#include "charlist.h"

const std::string lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing elit,\
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.\
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris\
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in\
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla\
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in\
culpa qui officia deserunt mollit anim id est laborum.";

template<typename Stream>
void debugList(const CharList& list, Stream& stream) {
   list.printList();
  stream << std::endl;
}

void debugOutputList(const CharList& list, const std::string& message) {
  std::cout << message;
   list.printList();
  std::cout << '\n';
```

```
}

std :: vector<std :: string > createLoremWords() {
  std :: vector<std :: string > lorem_words;
  std :: string_view sv(lorem);
  for (; sv.find(' ') != std :: string_view :: npos;
       sv.remove_prefix(sv.find(' ') + 1)) {
    lorem_words.push_back(std :: string(sv.substr(0, sv.find(' '))));
  }
  lorem_words.push_back(std :: string(sv));
  return lorem_words;
}

std :: string getRandomLoremWord(const std :: vector<std :: string >& lorem_words) {
  std :: random_device rd;
  std :: mt19937 gen(rd());
  std :: uniform_int_distribution <int> dist(1, lorem_words.size());
  return lorem_words[dist(gen)];
}

int main() {
  std :: vector<std :: string > lorem_words = createLoremWords();
  for (int i = 0; i < 5; ++i) {
    CharList list("Hello ");
    for (int k = 0; k < 5; ++k) {
      list.push_back(getRandomLoremWord(lorem_words));
      if (k != 4) {
        list.push_back(' ');
      }
    }
    debugList(list, std :: cout);
    list.moveLastWordToBeginning();
    debugOutputList(list, "Last word moved to the beginning: ");
    list.delete_second_word();
    debugOutputList(list, "Second word was deleted: ");
    auto lorem_word = getRandomLoremWord(lorem_words);
    list.replace_word(lorem_word, i + 1);
    debugOutputList(list, "Word " + std :: to_string(i+1)
        + " was replaced by " + lorem_word + ": ");
    std :: cout << std :: endl;
  }
  return 0;
}
```

2.4. Результаты тестирования

```
Hello_ea_enim_amet,_consequat._in
Last word moved to the beginning: in_Hello_ea_enim_amet,_consequat.
Second word was deleted: in_ea_enim_amet,_consequat.
Word 1 was replaced by ipsum : ipsum_ea_enim_amet,_consequat.

Hello_laborum._sit_anim_elit,sed_aliquip
Last word moved to the beginning: aliquip_Hello_laborum._sit_anim_elit,sed
Second word was deleted: aliquip_laborum._sit_anim_elit,sed
Word 2 was replaced by ea : aliquip_ea_sit_anim_elit,sed

Hello_sunt_tempor_cillum_Excepteur_inreprehenderit
Last word moved to the beginning: inreprehenderit_Hello_sunt_tempor_cillum_Excepteur
Second word was deleted: inreprehenderit_sunt_tempor_cillum_Excepteur
Word 3 was replaced by aliqua.Ut : inreprehenderit_sunt_aliqua.Ut_cillum_Excepteur

Hello_sint_est_proident,_et_incididunt
Last word moved to the beginning: incididunt_Hello_sint_est_proident,_et
Second word was deleted: incididunt_sint_est_proident,_et
Word 4 was replaced by eiusmod : incididunt_sint_est_eiusmod_et

Hello_sint_in_aliquip_dolor_velit
Last word moved to the beginning: velit_Hello_sint_in_aliquip_dolor
Second word was deleted: velit_sint_in_aliquip_dolor
Word 5 was replaced by sint : velit_sint_in_aliquip_sint
```

Рис. 7 - Результаты тестирования программы

Выводы

В ходе выполнения работы была реализована структура «однонаправленный список», поддерживающий следующие обязательные операции: вывод хранящихся значений узлов списка, добавление элемента перед и после узла, перемещение последнего слова в начало, удаление второго слова из списка, замена одного слова списка на другое по позиции в списке. Также в ходе выполнения работы были усвоены основы работы с односвязными списками. Тестирование подтвердило правильность работы методов.

Список информационных источников

1. Thomas H. Cormen, Clifford Stein и другие: Introduction to Algorithms, 3rd Edition. Сентябрь 2009. The MIT Press.

2. N. Wirth: Algorithms and Data Structures. Август 2004. https://people.inf.ethz.ch/wirth/AD.pdf.

3. Linked list // Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Linked_list (Дата обращения: 30.04.2021)

4. Курс Algorithms, part 1 // Coursera [Электронный ресурс]. URL: https://www.coursera.org/learn/algorithms-part1 (Дата обращения: 26.04.2021)