|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Однонаправленный динамический список»** | |
|  | |
| Выполнил студент группы ИКБО-07-22 | Козлов К.И. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получить знания и практические навыки управления динамическим однонаправленным списком.

# **Постановка задачи**

1. Разработать функцию для создания исходного списка, его вывода и добавления узла.
2. Информационная часть узла списка определена вариантом.
3. Разработать функции дополнительного задания варианта.
4. В основной программе выполнить тестирование каждой функции, описанной в задании. Программа должна позволять пользователю непрерывно выполнять операции над списком в произвольном порядке.
5. Составить отчет по выполненному заданию. В отчет включить ответы на вопросы к практической работе.

Требования к упражнениям:

Вариант №9. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | Дан линейный однонаправленный список L, содержащий текст. В каждом узле один символ. Слова разделены одним пробелом.  1 Разработать функцию, которая находит последнее слово и переставляет его в начало списка.  2 Разработать функцию, которая удаляет второе слово.  3 Разработать функцию, которая заменяет k-ое слово на новое слово. Длина нового слова может быть больше длины k-ого слова. |

# **Решение**

В данной практической работе была описана структуры node и list. Эти структуры выполняют функции узла в списке и самого списка – набора узлов соответственно.

Сведения по однонаправленному динамическому списку

Однонаправленный список – это совокупность узлов. Узел в односвязном динамическом списке – это его часть. В узле хранится информация о нем самом (какие-то данные, переменные) и информация о следующем узле (указатель на следующий узел. Указатель на следующий узел у последнего объекта равен нулевому указателю). Добавление элемента происходит в конце списка(push\_back).

Для решения упражнения были описаны несколько структур и написано несколько функций: структура node, реализующая узел списка.

|  |
| --- |
| struct node  {  char data;  node\* nextNodePtr;  node(char userData)  {  data = userData;  nextNodePtr = nullptr;  }  }; |

Также была описана структура list, реализующая однонаправленный динамический список. List имеет методы: Empty() – проверка на пустой список, PushBack – добавление элемента в конец списка, ShowList() – вывод спика на экран.

|  |
| --- |
| struct list  {  node\* head;  int length;  list()  {  head = nullptr;  length = 0;  }  bool Empty()  {  if (head == nullptr)  {  return (true);  }  else  {  return (false);  }  }  void PushBack(char userdata)  {  node\* app = new node(userdata);  if (Empty())  {  head = app;  length++;  }  else  {  node\* temp = head;  while (temp->nextNodePtr != nullptr)  {  temp = temp->nextNodePtr;  }  temp->nextNodePtr = app;  length++;  }  }  void ShowList()  {  if (Empty())  {  return;  }  else  {  node\* ptr = head;  while (ptr != nullptr)  {  std::cout << ptr->data;  ptr = ptr->nextNodePtr;  }  }  }  int SpaceCounter()  {  int spaceCounter = 0;  node\* ptr = head;  while (ptr != nullptr)  {  if (ptr->data == ' ')  {  spaceCounter++;  }  ptr = ptr->nextNodePtr;  }  return(spaceCounter);  }  }; |

Функция DeleteSecondWord() имеет 1 парамер: Sentence – структура list.

Эта функция реализует удаление второго слова из списка. Слово – последовательность узлов, разделенных пробелами.

|  |
| --- |
| list DeleteSecondWord(list Sentence)  {  if (Sentence.Empty())  {  return(Sentence);  }  else  {  node\* tempNode = Sentence.head;  while (tempNode->nextNodePtr->data != ' ')  {  if (tempNode->nextNodePtr->data == ' ')  {  break;  }  tempNode = tempNode->nextNodePtr;  }  node\* firstNodeDelete = tempNode;  int spaceCounter = 0;  while (spaceCounter != 2)  {  if (tempNode->nextNodePtr->data == ' ' || tempNode->nextNodePtr->data == '.')  {  spaceCounter++;  }  tempNode = tempNode->nextNodePtr;  }  node\* lastNodeDelete = tempNode;  firstNodeDelete->nextNodePtr = lastNodeDelete;  return(Sentence);  }  } |

Функция ShiftLastWord() имеет все параметры функции DeleteSecondWorl

Функция сдвигает последнее слово в начало спика.

|  |
| --- |
| list ShiftLastWord(list Sentence)  {  node\* tempNode = Sentence.head;  node\* firstNodeShift = nullptr;  node\* link = nullptr;  while (tempNode != nullptr)  {  if (tempNode->data == ' ')  {  link = tempNode;  firstNodeShift = tempNode->nextNodePtr;  }  else if (tempNode ->nextNodePtr->data == '.')  {  node\* lastNodeShift = tempNode;  link->nextNodePtr = lastNodeShift->nextNodePtr;  lastNodeShift->nextNodePtr = Sentence.head;  Sentence.head = firstNodeShift;  return(Sentence);  }  tempNode = tempNode->nextNodePtr;  }  return (Sentence);  } |

Функция AddUserWord() имеет такие же параметры как и функция ShiftLastWord ().

Функция заменяет выбранное по порядку слово на новое введенное.

|  |
| --- |
| list AddUserWorld(list Sentence, int k)  {  node\* deleteNodeOne = Sentence.head;  node\* deleteNodeTwo = nullptr;  node\* tempNode = Sentence.head;  int spaceCounter = 0;  while (tempNode != nullptr)  {  if (tempNode->data == ' ' || tempNode->data == '.')  {  spaceCounter++;  deleteNodeOne = deleteNodeTwo;  deleteNodeTwo = tempNode;  tempNode = tempNode->nextNodePtr;  if (deleteNodeOne == nullptr)  {  deleteNodeOne = Sentence.head;  }  if (spaceCounter == k)  {  char buff;  int blya = 0;  node\* lastNodeCreate = nullptr;  do  {  std::cin>>(buff);  if (buff == '.')  {  break;  }  node\* writeNode = new node(buff);  blya++;  if (blya == 1)  {  deleteNodeOne->nextNodePtr = writeNode;  writeNode->nextNodePtr = deleteNodeTwo;  lastNodeCreate = writeNode;  }  else  {  lastNodeCreate->nextNodePtr = writeNode;  lastNodeCreate = lastNodeCreate->nextNodePtr;  writeNode->nextNodePtr = deleteNodeTwo;  }  } while (true);  return(Sentence);  }  }  tempNode = tempNode->nextNodePtr;  }  return (Sentence);  } |
|  |

При запуске программы пользователь увидит начальный интерфейс программы

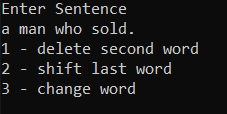


Рисунок 1. Интерфейс программы

# **4. Тестирование**

Протестируем работу программы. Для этого введем 1 – удалить второе слово.

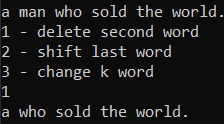


Рисунок 2. Проверка сдвига последнего слова

Из предложения удалилось второе слово man.

Протестируем замену слова на слово пользователя.

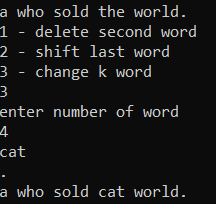


Рисунок 3. Проверка замены слова

Пользовательское слово cat заменило слово the, которое было четвертым

Протестируем сдвиг последнего слова

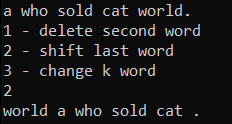


Рисунок 4. Проверка смещения последнего слова

Все функции и методы работают корректно.

# **5. Выводы**

В результате выполнения работы я:

1. Освоил навыки по реализации многоэлементных структур данных задачи С++
2. Освоил навыки по реализации однонаправленного динамического списка.

# **6. Исходный код программы**

|  |
| --- |
| #include <iostream>  struct node  {  char data;  node\* nextNodePtr;  node(char userData)  {  data = userData;  nextNodePtr = nullptr;  }  };  struct list  {  node\* head;  int length;  list()  {  head = nullptr;  length = 0;  }  bool Empty()  {  if (head == nullptr)  {  return (true);  }  else  {  return (false);  }  }  void PushBack(char userdata)  {  node\* app = new node(userdata);  if (Empty())  {  head = app;  length++;  }  else  {  node\* temp = head;  while (temp->nextNodePtr != nullptr)  {  temp = temp->nextNodePtr;  }  temp->nextNodePtr = app;  length++;  }  }  void ShowList()  {  if (Empty())  {  return;  }  else  {  node\* ptr = head;  while (ptr != nullptr)  {  std::cout << ptr->data;  ptr = ptr->nextNodePtr;  }  }  }  int SpaceCounter()  {  int spaceCounter = 0;  node\* ptr = head;  while (ptr != nullptr)  {  if (ptr->data == ' ')  {  spaceCounter++;  }  ptr = ptr->nextNodePtr;  }  return(spaceCounter);  }  };  list DeleteSecondWord(list Sentence)  {  if (Sentence.Empty())  {  return(Sentence);  }  else  {  node\* tempNode = Sentence.head;  while (tempNode->nextNodePtr->data != ' ')  {  if (tempNode->nextNodePtr->data == ' ')  {  break;  }  tempNode = tempNode->nextNodePtr;  }  node\* firstNodeDelete = tempNode;  int spaceCounter = 0;  while (spaceCounter != 2)  {  if (tempNode->nextNodePtr->data == ' ' || tempNode->nextNodePtr->data == '.')  {  spaceCounter++;  }  tempNode = tempNode->nextNodePtr;  }  node\* lastNodeDelete = tempNode;  firstNodeDelete->nextNodePtr = lastNodeDelete;  return(Sentence);  }  }  list ShiftLastWord(list Sentence)  {  node\* tempNode = Sentence.head;  node\* firstNodeShift = nullptr;  node\* link = nullptr;  while (tempNode != nullptr)  {  if (tempNode->data == ' ')  {  link = tempNode;  firstNodeShift = tempNode->nextNodePtr;  }  else if (tempNode ->nextNodePtr->data == '.')  {  node\* lastNodeShift = tempNode;  link->nextNodePtr = lastNodeShift->nextNodePtr;  lastNodeShift->nextNodePtr = Sentence.head;  Sentence.head = firstNodeShift;  return(Sentence);  }  tempNode = tempNode->nextNodePtr;  }  return (Sentence);  }  list AddUserWorld(list Sentence, int k)  {  node\* deleteNodeOne = Sentence.head;  node\* deleteNodeTwo = nullptr;  node\* tempNode = Sentence.head;  int spaceCounter = 0;  while (tempNode != nullptr)  {  if (tempNode->data == ' ' || tempNode->data == '.')  {  spaceCounter++;  deleteNodeOne = deleteNodeTwo;  deleteNodeTwo = tempNode;  tempNode = tempNode->nextNodePtr;  if (deleteNodeOne == nullptr)  {  deleteNodeOne = Sentence.head;  }  if (spaceCounter == k)  {  char buff;  int blya = 0;  node\* lastNodeCreate = nullptr;  do  {  std::cin>>(buff);  if (buff == '.')  {  break;  }  node\* writeNode = new node(buff);  blya++;  if (blya == 1)  {  deleteNodeOne->nextNodePtr = writeNode;  writeNode->nextNodePtr = deleteNodeTwo;  lastNodeCreate = writeNode;  }  else  {  lastNodeCreate->nextNodePtr = writeNode;  lastNodeCreate = lastNodeCreate->nextNodePtr;  writeNode->nextNodePtr = deleteNodeTwo;  }  } while (true);  return(Sentence);  }  }  tempNode = tempNode->nextNodePtr;  }  return (Sentence);  }  int main()  {  list Sentence;  char userData;  std::cout << "Enter Sentence" << '\n';  do  {  std::cin.get(userData);  if (userData == NULL)  {  Sentence.PushBack(' ');  }  Sentence.PushBack(userData);  } while (userData != '.');  int control;  do  {  std::cout << "1 - delete second word" << std::endl;  std::cout << "2 - shift last word" << std::endl;  std::cout << "3 - change k word" << std::endl;  std::cin >> control;  switch (control)  {  case 1:  {  Sentence = DeleteSecondWord(Sentence);  Sentence.ShowList();  std::cout << std::endl;  break;  }  case 2:  {  Sentence = ShiftLastWord(Sentence);  Sentence.ShowList();  std::cout << std::endl;  break;  }  case 3:  {  int wordPos;  do  {  std::cout << "enter number of word" << std::endl;  std::cin >> wordPos;  } while (wordPos > Sentence.SpaceCounter());  Sentence = AddUserWorld(Sentence, wordPos);  Sentence.ShowList();  std::cout << std::endl;  break;  }  }  } while (control == 1 || control == 2 || control == 3);  } |