ГУАП

# КАФЕДРА №43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  | 28.09.2023 |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 2 |
| ЛИНЕЙНЫЕ И ЦИКЛИЧЕСКИЕ СПИСКИ |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232 |  | 28.09.2023 |  | М.В.Куриш |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Цель работы:**

Целью работы является изучение структур данных «линейный список» и «циклический список», а также получение практических навыков их реализации.

1. **Задание на лабораторную работу:**

Реализовать структуры данных «линейный список» и «циклический список» в соответствии с заданным вариантом. Дополнительно программа должна осуществлять следующие операции:

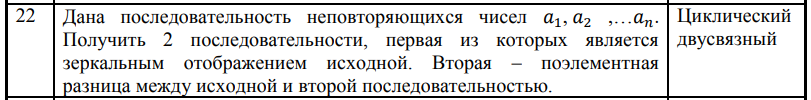
1) Добавление/удаление элемента в список (с клавиатуры);

2) Вывод исходного и результирующего списков на экран;

3) Если списки являются многочленами, в выводе должна быть отражена степень каждого элемента.

Варианты задания приведены в таблице 2. Элементы последовательности, или коэффициенты многочлена (в зависимости от варианта) – числовые значения элемента списка, количество таких элементов списка равно длине последовательности, или количеству коэффициентов многочлена.

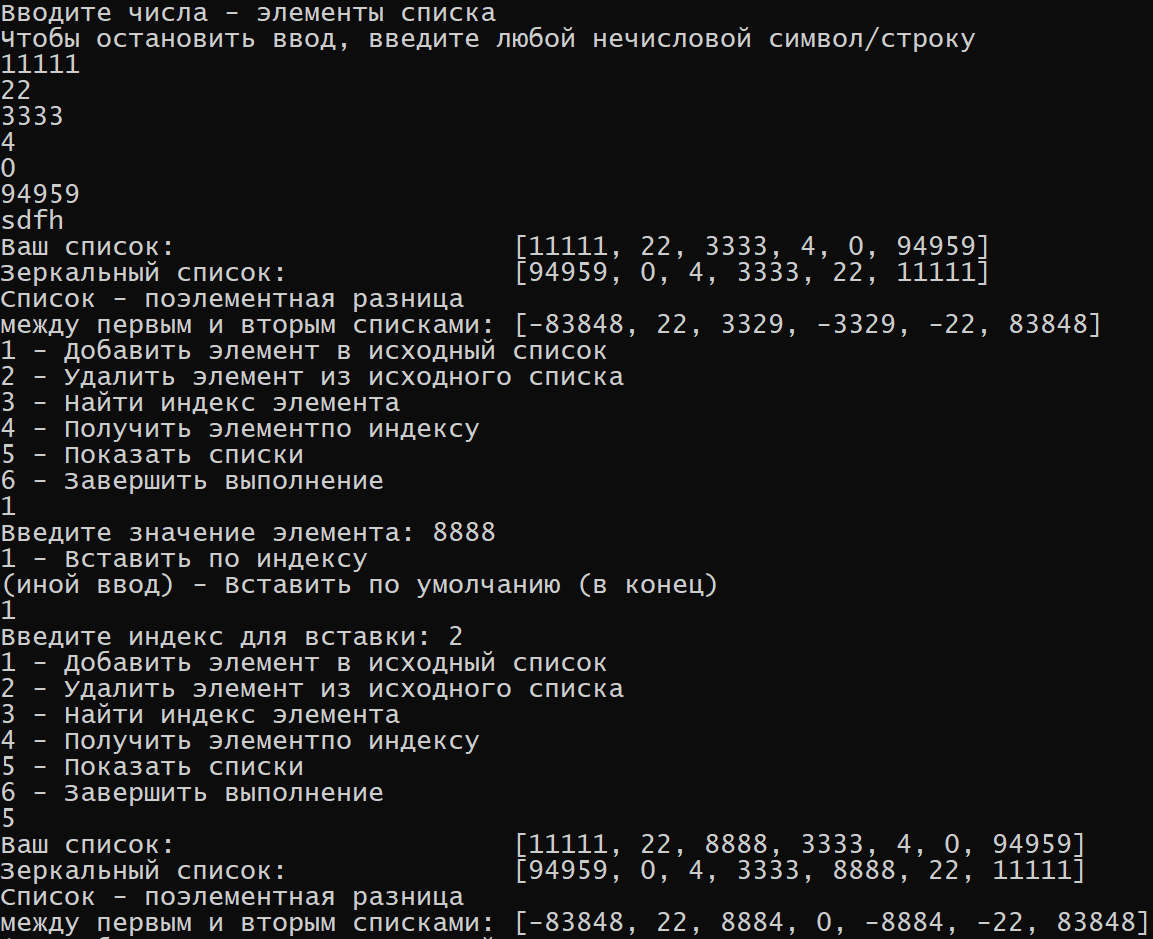
Вариант 22.

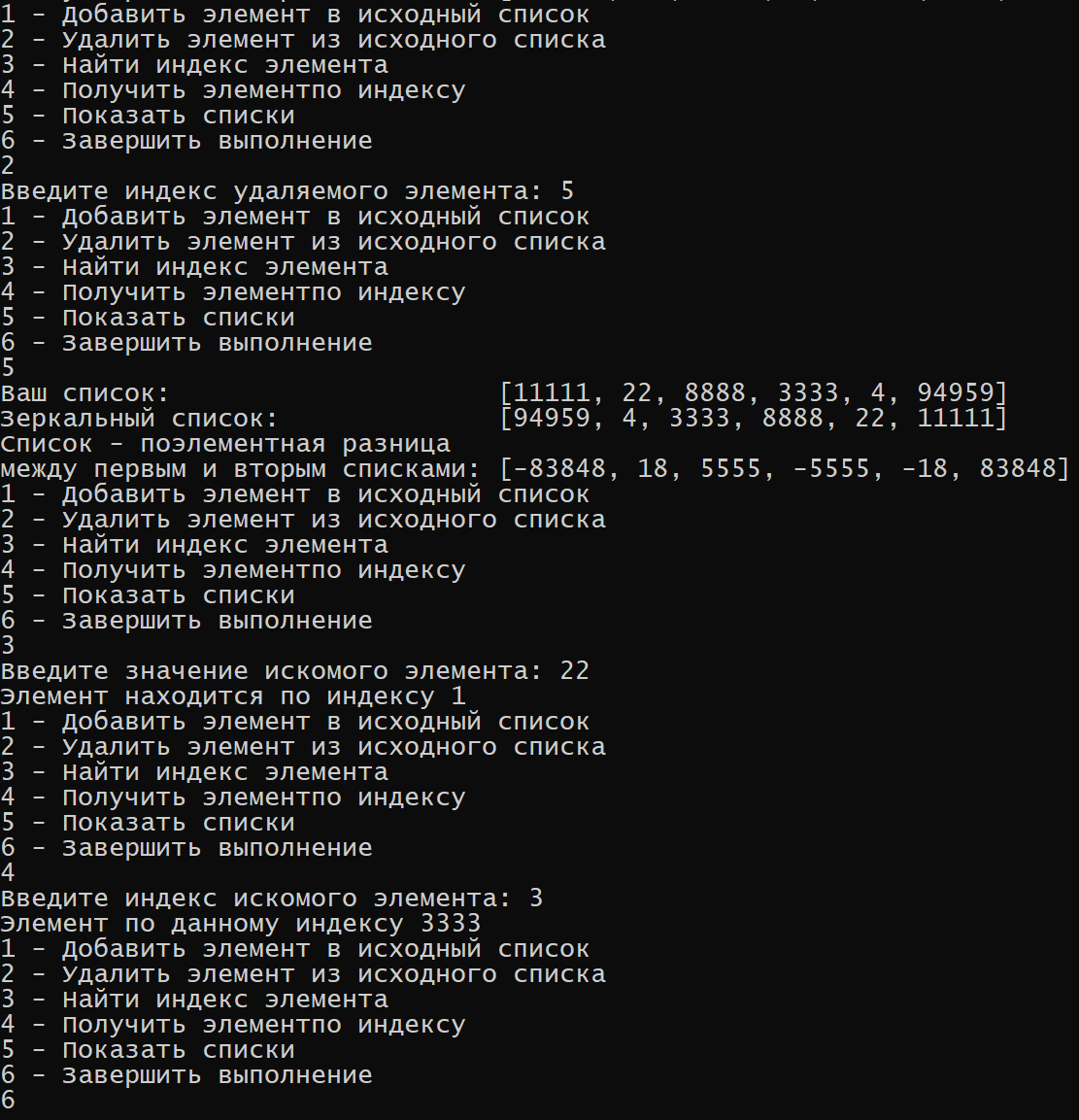


1. **Листинг программы:**

|  |
| --- |
| #include <iostream>  #include <stdexcept>  using namespace std;  int intInput() {  int a;  cin.clear();  cin.ignore(999, '\n');  cin >> a;  while (!cin) {  cout << "Неверный ввод. (Ожидается число)\n";  cin.clear();  cin.ignore(999, '\n');  cout << "Введите заново: ";  cin >> a;  }  return a;  }  template <typename T>  class DoubleLinkedCycledList {  private:  template <typename T>  class Node {  public:  T value;  Node<T>\* next;  Node<T>\* prev;  Node(T value, Node<T>\* prev = nullptr, Node<T>\* next = nullptr) {  this->value = value;  this->next = next;  this->prev = prev;  }  };  Node<T>\* head;  int size;  public:  DoubleLinkedCycledList() {  head = nullptr;  size = 0;  }  void add(T value, int index) {  if (index < 0 || index > size) {  cout << "Индекс вне границ списка\n";  return;  }  if (size == 0) {  head = new Node<T>(value);  head->next = head;  head->prev = head;  }  else {  if (index == 0) {  Node<T>\* additional = new Node<T>(value, head->prev, head);  head->prev->next = additional;  head->prev = additional;  head = additional;  }  else if (index > 0 && index < size / 2) {  Node<T>\* curr = head;  while (index != 0) {  curr = curr->next;  index--;  }  Node<T>\* additional = new Node<T>(value, curr->prev, curr);  curr->prev->next = additional;  curr->prev = additional;  }  else if (index >= size / 2 && index < size) {  Node<T>\* curr = head->prev;  while (index != size - 1) {  curr = curr->prev;  index++;  }  Node<T>\* additional = new Node<T>(value, curr->prev, curr);  curr->prev->next = additional;  curr->prev = additional;  }  else {  Node<T>\* additional = new Node<T>(value, head->prev, head);  head->prev->next = additional;  head->prev = additional;  }  }  size++;  }    void add(T value) {  add(value, size);  }  void remove(int index) {  if (size == 0) {  cout << "Список пуст\n";  return;  }  if (index >= size || index < 0) {  cout << "Индекс за границами (index = " << index << "; size = " << size << ")\n";  return;  }  if (size == 1) {  delete head;  }  else {  Node<T>\* curr = head;  if (index > 0 && index < size / 2) {  while (index != 0) {  curr = curr->next;  index--;  }  }  else if (index >= size / 2 && index < size){  while (index != size) {  curr = curr->prev;  index++;  }  }  curr->prev->next = curr->next;  curr->next->prev = curr->prev;  if (index == 0)  head = curr->next;  delete curr;  }  size--;  }  void remove() {  remove(size - 1);  }  void show() {  cout << "[";  Node<T>\* curr = head;  for (int i = 0; i < size; i++) {  cout << curr->value;  curr = curr->next;  if (i != size - 1) cout << ", ";  }  cout << "]\n";  }  int indexOf(T value) {  int ind = 0;  Node<T>\* curr = head;  while (ind < size) {  if (value == curr->value)  return ind;  ind++;  curr = curr->next;  }  throw invalid\_argument("Значение не найдено.\n");  }  T& getElement(int index) {  Node<T>\* curr = head;  if (index >= size)  throw invalid\_argument("Индекс за границами.\n");  while (index != 0) {  curr = curr->next;  index--;  }  return curr->value;  }  int getSize() {  return size;  }  void operator = (const DoubleLinkedCycledList<T>& other) {  this->~DoubleLinkedCycledList();  Node<T>\* curr = other.head;  for (int i = 0; i < other.size; i++) {  this->add(curr->value);  curr = curr->next;  }  }  DoubleLinkedCycledList<T> mirrorizeList() {  DoubleLinkedCycledList<T> resultList;  Node<T>\* curr = head->prev;  for (int i = size; i != 0; i--) {  resultList.add(curr->value);  curr = curr->prev;  }  return resultList;  }  DoubleLinkedCycledList<T> differenceBtwLists(DoubleLinkedCycledList<T>& deductible) {  DoubleLinkedCycledList<T> resultList;  Node<T>\* first = head;  Node<T>\* second = deductible.head;  for (int i = 0; i < size; i++) {  resultList.add(first->value - second->value);  first = first->next;  second = second->next;  }  return resultList;  }  ~DoubleLinkedCycledList() {  while (size > 0) {  remove();  }  }  };  int main()  {  setlocale(LC\_ALL, "RUS");  DoubleLinkedCycledList<int> list;  cout << "Вводите числа - элементы списка\n";  cout << "Чтобы остановить ввод, введите любой нечисловой символ/строку\n";  int a;  cin >> a;  while (cin.good()) {  list.add(a);  cin >> a;  }  cout << "Ваш список:\t\t\t";  list.show();  DoubleLinkedCycledList<int> mirroredList = list.mirrorizeList();  cout << "Зеркальный список:\t\t";  mirroredList.show();  DoubleLinkedCycledList<int> diffList = list.differenceBtwLists(mirroredList);  cout << "Список - поэлементная разница\nмежду первым и вторым списками:\t";  diffList.show();  bool flag = true;  int aaa;  while (flag) {  cout << "1 - Добавить элемент в исходный список\n";  cout << "2 - Удалить элемент из исходного списка\n";  cout << "3 - Найти индекс элемента\n";  cout << "4 - Получить элементпо индексу\n";  cout << "5 - Показать списки\n";  cout << "6 - Завершить выполнение\n";  switch (intInput()) {  case 1:  cout << "Введите значение элемента: ";  aaa = intInput();  cout << "1 - Вставить по индексу\n";  cout << "(иной ввод) - Вставить по умолчанию (в конец)\n";  if (intInput() == 1) {  cout << "Введите индекс для вставки: ";  list.add(aaa, intInput());  }  else {  list.add(aaa);  }  break;  case 2:  cout << "Введите индекс удаляемого элемента: ";  list.remove(intInput());  break;  case 3:  cout << "Введите значение искомого элемента: ";  try {  cout << "Элемент находится по индексу " << list.indexOf(intInput()) << endl;  }  catch (invalid\_argument& e) {  cout << e.what();  }  break;  case 4:  cout << "Введите индекс искомого элемента: ";  try {  cout << "Элемент по данному индексу " << list.getElement(intInput()) << endl;  }  catch (invalid\_argument& e) {  cout << e.what();  }  break;  case 5:  cout << "Ваш список:\t\t\t";  list.show();  mirroredList = list.mirrorizeList();  cout << "Зеркальный список:\t\t";  mirroredList.show();  diffList = list.differenceBtwLists(mirroredList);  cout << "Список - поэлементная разница\nмежду первым и вторым списками:\t";  diffList.show();  break;  case 6:  flag = false;  break;  default:  cout << "Ошибка выбора пункта меню\n";  break;  }  }  } |

1. **Контрольные примеры:**





1. **Вывод:**

Изучены структуры данных «линейный список» и «циклический список». Получен опыт их реализации. Отработан навык обработки двусвязного циклического списка.