**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Программирование»**

| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Писарев А.С. |

Санкт-Петербург

2024 г.

# Исходная формулировка

Требуется разработать программу обеспечивающую:

1. построение циклического односвязного списка (информационные части узлов должны иметь целочисленный тип и заполняться путем последовательного ввода значений с клавиатуры; признак окончания ввода – введенный 0);
2. обработку созданного списка с выполнением следующих функций:
   1. вставку узлов с информационной частью = 1 во все позиции списка предшествующие узлам с отрицательной информационной частью
   2. удаление всех узлов списка, содержащих в информационной части отрицательные числа;
   3. расчет количества вхождений заданного с клавиатуры числа в информационные поля узлов;
   4. рекурсивное удаление всех узлов списка.

# Ход работы программы

В начале работы программы создаётся список.

| 6 | List<int> list; |
| --- | --- |

Затем в консоль выводится сообщение о вводе чисел. После ввода чисел выводится необработанный список.

| 8  9  10  11 | std::cout << "Enter list of integers (0 to stop):" << std::endl;  Utils::enterList(list);  std::cout << "List:" << std::endl;  Utils::printList(list); |
| --- | --- |

После этого в список добавляются элементы с информационной частью = 1 и список выводится повторно.

| 13  14  15 | std::cout << "List after inserting nodes before negative values:" << std::endl;  Utils::insertNodesBeforeNegative(list);  Utils::printList(list); |
| --- | --- |

Затем из списка удаляются элементы с отрицательной информационной частью и список выводится в третий раз.

| 17  18  19 | std::cout << "List after removing nodes with negative values:" << std::endl;  Utils::removeNodesWithNegativeValues(list);  Utils::printList(list); |
| --- | --- |

После этого выводится сообщение о вводе числа, для подсчёта количества элементов с такой информационной частью. Затем выводится это число элементов.

| 21  22  23  24 | int number;  std::cout << "Enter number to count occurrences: ";  std::cin >> number;  std::cout << "Number of occurrences: " << Utils::countOccurrences(list, number) << std::endl; |
| --- | --- |

Затем все рекурсивно удаляются все элементы из списка и выводится пустой список.

| 26  27  28 | std::cout << "List after removing all nodes recursively:" << std::endl;  Utils::recursiveRemoveAllNodes(list);  Utils::printList(list); |
| --- | --- |

# Текст программы

| Main.cpp | |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include "Utils.h"  #include <iostream>  int main()  {  List<int> list;    std::cout << "Enter list of integers (0 to stop):" << std::endl;  Utils::enterList(list);  std::cout << "List:" << std::endl;  Utils::printList(list);  std::cout << "List after inserting nodes before negative values:" << std::endl;  Utils::insertNodesBeforeNegative(list);  Utils::printList(list);  std::cout << "List after removing nodes with negative values:" << std::endl;  Utils::removeNodesWithNegativeValues(list);  Utils::printList(list);  int number;  std::cout << "Enter number to count occurrences: ";  std::cin >> number;  std::cout << "Number of occurrences: " << Utils::countOccurrences(list, number) << std::endl;  std::cout << "List after removing all nodes recursively:" << std::endl;  Utils::recursiveRemoveAllNodes(list);  Utils::printList(list);  return 0;  } |
| Utils.h | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | #pragma once  #include "List.h"  class Utils  {  public:  static void enterList(List<int>& list)  {  int temp = 1;  while (temp != 0)  {  std::cin >> temp;  if (temp != 0)  list.push\_back(temp);  }  }  static void printList(const List<int>& list)  {  for (int i = 0; i < list.size(); i++)  std::cout << list[i] << " ";    if (!list.empty())  std::cout << std::endl;  }  static void insertNodesBeforeNegative(List<int>& list)  {  for (int i = 0; i < list.size(); i++)  {  if (list[i] < 0)  {  list.insert(i, 1);  i++;  }  }  }  static void removeNodesWithNegativeValues(List<int>& list)  {  for (int i = 0; i < list.size();)  if (list[i] < 0)  list.erase(i);  else  i++;  }  static int countOccurrences(const List<int>& list, int number)  {  int count = 0;  for (int i = 0; i < list.size(); i++)  if (list[i] == number)  count++;  return count;  }  static void recursiveRemoveAllNodes(List<int>& list)  {  list.clear();  }  }; |
| List.h | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233 | #pragma once  #include <iostream>  template <typename T>  class List  {  private:  struct Node  {  T data;  Node\* next;  Node(const T& data, Node\* next): data(data), next(next) {}  ~Node() {}  };  Node\* head;  Node\* tail;  int length;  public:  List(): head(nullptr), tail(nullptr), length(0) {}  void push\_back(const T& data)  {  Node\* newNode = new Node(data, nullptr);  if (head == nullptr)  {  head = newNode;  tail = newNode;  }  else  {  tail->next = newNode;  tail = newNode;  }  tail->next = head;  length++;  }  void push\_front(const T& data)  {  Node\* newNode = new Node(data, nullptr);  if (head == nullptr)  {  head = newNode;  tail = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  tail->next = head;  length++;  }  void pop\_back()  {  if (empty())  {  std::cout << "List is empty" << std::endl;  return;  }  Node\* temp = head;  Node\* toDelete = tail;  while (temp->next != tail)  {  temp = temp->next;  }  tail = temp;  tail->next = head;  delete toDelete;  length--;  }  void pop\_front()  {  if (empty())  {  std::cout << "List is empty" << std::endl;  return;  }  Node\* toDelete = head;  head = head->next;  tail->next = head;  delete toDelete;  length--;  }  void insert(int index, const T& data)  {  if (index < 0 || index > length)  {  std::cout << "Invalid index" << std::endl;  return;  }  if (index == 0)  {  push\_front(data);  return;  }  if (index == length)  {  push\_back(data);  return;  }  Node\* temp = head;  Node\* newNode = new Node(data, nullptr);  for (int i = 0; i < index - 1; i++)  {  temp = temp->next;  }  newNode->next = temp->next;  temp->next = newNode;  length++;  }  void erase(int index)  {  if (empty())  {  std::cout << "List is empty" << std::endl;  return;  }  if (index < 0 || index >= length)  {  std::cout << "Invalid index" << std::endl;  return;  }  if (index == 0)  {  pop\_front();  return;  }  if (index == length - 1)  {  pop\_back();  return;  }  Node\* temp = head;  for (int i = 0; i < index - 1; i++)  {  temp = temp->next;  }  Node\* toDelete = temp->next;  temp->next = toDelete->next;  delete toDelete;  length--;  }  T& operator[](int index) const  {  if (empty())  {  throw std::out\_of\_range("List is empty");  }  if (index < 0 || index >= length)  {  throw std::out\_of\_range("Invalid index");  }  Node\* temp = head;  for (int i = 0; i < index; i++)  {  temp = temp->next;  }  return temp->data;  }  void clear()  {  if (head == nullptr)  {  return;  }  tail->next = nullptr;  Node\* current = head;  while (current != nullptr)  {  Node\* next = current->next;  delete current;  current = next;  }  head = nullptr;  tail = nullptr;  length = 0;  }  int size() const  {  return length;  }  bool empty() const  {  return length == 0;  }  ~List()  {  clear();  }  }; |

# Результаты работы программы

