**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Программирование»**

**Тема: Работа со списком**

Студент гр. 2302 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Николаев В.Ю.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калмычков В.А.

Санкт-Петербург

2023 г.

# Исходная формулировка

Вставить элемент после каждого элемента, имеющего заданное значение.

# **Математическая постановка задачи**

В дальнейшем элемент, после которого необходимо вставить новый будем называть v1, а элемент, который надо вставить будем назвать v2.

Используемые условия:

С1: haveErrors

C2: list[i] == v1

C3: size != 0

C4: data != NULL

C5: fin >> c && c != '\n'

C6: data == NULL

C7: !fin.is\_open()

C8: fin >> c

C9: n-- && !v1.Input(fin)

C10: n-- && !v2.Input(fin)

C11: n-- && data[n].Input(fin)

# Контрольный пример

Исходный текст:

1. privet
2. kak dela?
3. privet
4. vse horosho.
5. privet

Элемент, после которого необходимо вставить новый:

1. privet

Элемент, который необходим вставить:

1. poka

Полученный текст:

1. privet
2. poka
3. kak dela?
4. privet
5. poka
6. vse horosho.
7. privet
8. poka

# **Организация UI**

| Макет файла input.txt:   1. c\_c 2. c\_c 3. ⋮ 4. c\_c | Макет файла output.txt:   1. c\_c 2. c\_c 3. ⋮ 4. c\_c |
| --- | --- |

# Особенности системы программирования

Проект собран на g++ (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0. Для сборки используется GNU Make 4.3 Built for x86\_64-pc-linux-gnu.

# Структуры данных

| Строка | Элемент списка | Список |
| --- | --- | --- |
| class String  {  public:  char \*data = NULL;  int size = 0;    public:  String();  String(int);  bool Input(std::fstream &);  friend bool operator==(const String &, const String &);    friend std::ostream& operator<<(std::ostream &, const String &);  }; | class Node  {  public:  String data;  Node \*next = NULL;  public:  Node();  Node(String);  }; | class List  {  private:  Node \*head = NULL;  public:  void Insert(String);  void Insert(String, int);  void Delete();  void Delete(int);  int Size();  const String operator[](int);  }; |
| char \*data - указатель на массив символов строки;  int size - размер массива символов. | String data - элемент списка  Node \*next = NULL - указатель на следующий элемент списка | Node \*head = NULL - первый элемент списка |
|  |  |  |

# **Способ хранения данных**

| Тип данных | Название переменной | Описание переменной |
| --- | --- | --- |
| String | v1 | Элемент, после которого необходимо вставить новый |
| v2 | Элемент, который необходим вставить |
| List | list | Список элементов |

# Описание функций

| Тип | Название | Передаваемые переменные | Описание переменных | Описание функции |
| --- | --- | --- | --- | --- |
| List | InputFromFile | const char \*fileName | название файла | Считывание данных из файла input.txt |
| String &v1 | элемент v1 |
| String &v2 | элемент v2 |
| void | OutputToFile | const char \*fileName | название файла | Вывод данных в файл output.txt |
| List list | список |
| std::ios\_base::  openmode openmode | режим вывода |
| void | Processing | List &list | список | Обработка списка |
| String v1 | элемент v1 |
| String v2 | элемент v2 |
| bool | Input | std::fstream &fin | поток ввода | Считывание строки из потока ввода |
| bool | operator== | const String &left | левая строка | Оператор сравнения двух строк |
| const String &right | правая строка |
| std::  ostream& | operator<< | std::ostream &os | поток вывода | Вывод строки в поток вывода |
| const String &string | строка |
| void | Insert | String data | строка | Вставка строки в конец списка |
| void | Insert | String data | строка | Вставка строки после элемента |
| int number | номер элемента |
| void | Delete | - | - | Удаление списка |
| void | Delete | int number | номер элемента | Удаление элемента по номеру |
| int | Size | - | - | Размер списка |
| const String | operator[] | int number | номер элемента | Оператор индексации |

# **Представление алгоритма решения задачи**

# Текст программы

| | MakeFile | | --- | | all:  cd build/String/; g++ -c -Wall ../../lib/String/\*.cpp; cd ../../  cd build/Node/; g++ -c -Wall ../../lib/Node/\*.cpp; cd ../../  cd build/List/; g++ -c -Wall ../../lib/List/\*.cpp; cd ../../  cd build/Task/; g++ -c -Wall ../../lib/Task/\*.cpp; cd ../../  g++ -Wall build/String/\*.o build/Node/\*.o build/List/\*.o build/Task/\*.o src/Main.cpp -o build/build.exe  ./build/build.exe | | |
| --- | --- | --- | --- |
| src/   | Main.cpp | | --- | | #include "../lib/Task/Task.hpp"  int main()  {  String v1, v2;  List list = InputFromFile("files/input.txt", v1, v2);  OutputToFile("files/output.txt", list, std::ios::out);    Processing(list, v1, v2);    OutputToFile("files/output.txt", list, std::ios::app);  list.Delete();  return 0;  } |   lib/   | Header.hpp | | --- | | #pragma once  #include <iostream>  #include <fstream> |   lib/Node/   | Node.hpp | | --- | | #pragma once  #include "../String/String.hpp"  class Node  {  public:  String data;  Node \*next = NULL;  public:  Node();  Node(String);  }; | | Construction.cpp | | #include "Node.hpp"  Node::Node()  {}  Node::Node(String \_data)  {  data = \_data;  } |   lib/Tast/   | Task.hpp | | --- | | #pragma once  #include "../List/List.hpp"  static bool haveErrors = false;  List InputFromFile(const char \*, String &, String &);  void OutputToFile(const char \*, List, std::ios\_base::openmode);  void Processing(List &, String, String); | | InputFromFile.cpp | | #include "Task.hpp"  List InputFromFile(const char \*fileName, String &v1, String &v2)  {  List list;  std::fstream fin(fileName, std::ios::in);  if (!fin.is\_open())  {  haveErrors = true;  std::cout << "E1\n";  fin.close();  return list;  }  fin << std::noskipws;  char c;  int n = 0;  while (fin >> c)  n += c == '\n';  fin.close();  fin.open(fileName, std::ios::in);  if (!fin.is\_open())  {  haveErrors = true;  std::cout << "E1\n";  fin.close();  return list;  }  fin << std::noskipws;  if (n-- && !v1.Input(fin))  {  haveErrors = true;  std::cout << "E3\n";    fin.close();  return list;  }  if (n-- && !v2.Input(fin))  {  haveErrors = true;  std::cout << "E4\n";    fin.close();  return list;  }  String data[n];  while (n-- && data[n].Input(fin))  {  list.Insert(data[n]);  }  fin.close();  return list;  } | | OutputToFile.cpp | | #include "Task.hpp"  void OutputToFile(const char \*fileName, List list, std::ios\_base::openmode openmode)  {  if (haveErrors)  return;  std::fstream fout(fileName, openmode);  if (!fout.is\_open())  {  haveErrors = true;  std::cout << "E2\n";  fout.close();  return;  }  fout << " ";  for (int i = 0; i < list.Size(); i++)  fout << list[i] << "--> ";  fout << "NULL\n\n";  fout.close();  return;  } | | Processing.cpp | | #include "Task.hpp"  void Processing(List &list, String v1, String v2)  {  if (haveErrors)  return;  for (int i = 0; i < list.Size(); i++)  if (list[i] == v1)  list.Insert(v2, i++);  } |   lib/List/   | Size.cpp | | --- | | #include "List.hpp"  int List::Size()  {  int size = 0;  Node \*cur = head;  while (cur != NULL)  {  cur = cur->next;  size++;  }  return size;  } | | lib/String/   | String.hpp | | --- | | #pragma once  #include "../Header.hpp"  class String  {  public:  char \*data = NULL;  int size = 0;    public:  String();  String(int);    bool Input(std::fstream &);  friend bool operator==(const String &, const String &);    friend std::ostream& operator<<(std::ostream &, const String &);  }; | | Constructors.cpp | | #include "String.hpp"  String::String() {}  String::String(int number)  {  int t = number;  while (t)  {  size++;  t /= 10;  }  data = new char[size];  t = size;  while (number)  {  data[--t] = number%10 + '0';  number /= 10;  }  } | | Input.cpp | | #include "String.hpp"  bool String::Input(std::fstream &fin)  {  if (size != 0)  size = 0;  if (data != NULL)  delete []data;  char c;  while (fin >> c && c != '\n')  size++;  size++;  for (int i = 0; i < size; i++)  fin.unget();  data = new char[size];  if (data == NULL)  {  std::cout << "E1\n";  return false;  }  for (int i = 0; i < size; i++)  fin >> data[i];  return true;  } | | Operators.cpp | | #include "String.hpp"  bool operator==(const String &left, const String &right)  {  if (left.side == right.size)  for (int i = 0; i < left.size; i++)  if (left.data[i] != right.data[i])  return false;  return left.side == right.size;  }  std::ostream& operator<<(std::ostream &os, const String &string)  {  for (int i = 0; i < string.size; i++)  os << string.data[i];  return os;  } |   lib/List/   | List.hpp | | --- | | #pragma once  #include "../Node/Node.hpp"  class List  {  private:  Node \*head = NULL;  public:  void Insert(String);  void Insert(String, int);  void Delete();  void Delete(int);  int Size();  const String operator[](int);  }; | | Delete.cpp | | #include "List.hpp"  void List::Delete()  {  Node \*cur = head;  while (cur != NULL)  {  Node \*tmp = cur->next;  delete cur;  cur = tmp;  }  }  void List::Delete(int number)  {  Node \*cur = head;  while (number-- && cur != NULL)  cur = cur->next;  } | | Insert.cpp | | #include "List.hpp"  void List::Insert(String data)  {  if (head == NULL)  {  head = new Node(data);  return;  }  Node \*cur = head;  while(cur->next != NULL)  cur = cur->next;  cur->next = new Node(data);  }  void List::Insert(String data, int number)  {  if (0 > number || number > Size())  return;  if (head == NULL)  {  head = new Node(data);  return;  }  Node \*cur = head;  while(number--)  cur = cur->next;  Node \*tmp = cur->next;  cur->next = new Node(data);  cur->next->next = tmp;  } | | Operators.cpp | | #include "List.hpp"  const String List::operator[](const int number)  {  if (0 > number || number >= Size())  return 0;  Node \*cur = head;  int t = number;  while (t--)  cur = cur->next;    return cur->data;  } | |

# **Результаты работы программы**

Контрольный пример программа прошла успешно. Все некорректные ситуации программа обрабатывает корректно (сообщает о полученной ошибке).

# Выводы о проделанной работе

В этой лабораторной работе мы приобрели навыки работы со списком.