**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**отчет**

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Списки»**

**Вариант 7**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8302 |  | Чёрный Я.И. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Оглавление**

[1 Постановка задачи 3](#_Toc23118737)

[1.1 Реализуемые классы и методы: 3](#_Toc23118738)

[1.1.1 Node: 3](#_Toc23118739)

[1.1.2 List: 3](#_Toc23118740)

[2 Unit-тесты 4](#_Toc23118741)

[3 Пример работы 5](#_Toc23118742)

[3.1 Функция main, представляющая пример использования написанной структуры данных: 5](#_Toc23118743)

[3.2 Результат выполнения: 6](#_Toc23118744)

[4 Текст программы: 6](#_Toc23118745)

[5 Вывод 8](#_Toc23118746)

# Постановка задачи

Целью данной задачи является реализация однонаправленного списка с набором дополнительных функций, позволяющим менять элементы списка местами, удалять, вставлять новые и так далее. Помимо, непосредственно, алгоритмов, важной частью этой задачи является обработка исключений, написание юнит-тестов и внутренняя организация программы.

## Реализуемые классы и методы:

### Node:

|  |  |
| --- | --- |
| Поле | Назначение |
| Node<T>\* next | Указатель на следующий элемент |
| T data | Хранимые данные |

|  |  |  |
| --- | --- | --- |
| Функция | Назначение | Временная сложность |
| void SetNext(Node<T>\*) | Записывает указатель в поле next | O(1) |
| void SetData(T) | Записывает значение в поле data | O(1) |
| Node\* GetNext() | Возвращает указатель на сл. элемент | O(1) |
| T GetData() | Возвращает значение поля data | O(1) |

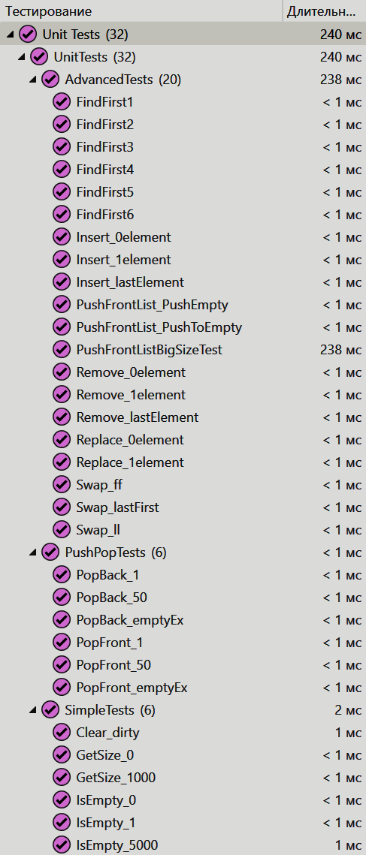
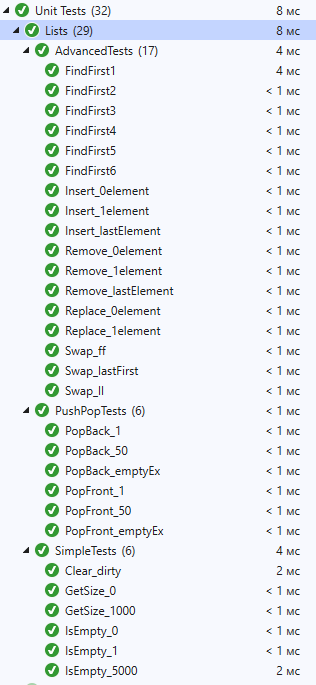
### List:

|  |  |
| --- | --- |
| Поле | Назначение |
| size\_t curElement | Номер текущего элемента |
| size\_t count | Количество элементов |
| Node<T> \*head | Указатель на первый элемент |
| Node<T> \* tail | Указатель на последний элемент |
| Node<T> \* current | Указатель на текущий элемент |
| Node<T> \* previous | Указатель на предыдущий элемент |

|  |  |  |
| --- | --- | --- |
| Функция | Назначение | Временная сложность |
| void SetHeadNode(Node<T>\*) | Замена указателя на первый эл-т | O(1) |
| void SetTailNode(Node<T>\*) | Замена указателя на последний эл-т | O(1) |
| void SetCurrentNode(Node<T>\*) | Замена указателя на текущий эл-т | O(1) |
| void SetCurrentValue(T) | Запись значения в текущий эл-т | O(1) |
| void SetPreviousNode(Node<T>\*) | Замена указателя на предыдущий элемент | O(1) |
| Node<T>\* GetHeadNode() | Возвращает указатель на первый элемент | O(1) |
| Node<T>\* GetTailNode() | Возвращает указатель на последний эл-т | O(1) |
| Node<T>\* GetCurrentNode() | Возвращает указатель на текущий эл-т | O(1) |
| T GetCurrentValue() | Возвращает значение текущего эл-та | O(1) |
| Node<T>\* GetPreviousNode() | Возвращает указатель на предыдущий эл-т | O(1) |
| Node <T>\* GetNode(size\_t) | Возвращает указатель на элемент с порядковым номером | O(N) |
| void MoveForward() | Перемещает указатель на следующий эл-т | O(1) |
| void MoveToHead() | Перемещает указатель на первый эл-т | O(1) |
| void MoveToTail() | Перемещает указатель на последний эл-т | O(1) |
| void CheckBounds(size\_t) | Проверяет, может ли быть обработан входной индекс эл-та | O(1) |
| void AddValues(initializer\_list<T>) | Добавляет в список новые эл-ты | O(N) |
| void RemoveAll(T) | Удаляет из списка все эл-ты со значением | O(N^2) |
| void RemoveFirst(T) | Удаляет из списка первый найденный элемент | O(N^2) |
| void Swap(size\_t, size\_t) | Меняет местами эл-ты списка | O(N) |
| size\_t FindFirst(List<T>&) | Возвращает индекс первого вхождения второго списка в текущий | O(N^2) |
| void PushFront(List<T>&) | Добавляет другой список в начало | O(N) |
| void PushBack(T) | Добавляет новый эл-т в конец | O(1) |
| void PushFront(T) | Добавляет новый эл-т в начало | O(1) |
| void PopBack() | Удаляет элемент с конца | O(N) |
| void PopFront() | Удаляет элемент с начала | O(N) |
| void Insert(T, size\_t) | Вставляет новый эл-т в индекс, вытесняя эл-ты вперёд | O(N) |
| T At(size\_t) | Возвращает значение эл-та по индексу | O(N) |
| void Remove(size\_t) | Удаляет эл-т по индексу | O(N) |
| size\_t GetSize() | Возвращает размер списка | O(1) |
| void Clear() | Удаляет все эл-ты списка | O(N) |
| void Replace(T, size\_t) | Заменяет элемент списка по индексу | O(N) |
| bool IsEmpty() | Возвращает значение, говорящее о том, пуст ли список | O(1) |
| void PrintToConsole() | Выводит список в консоль | O(N) |

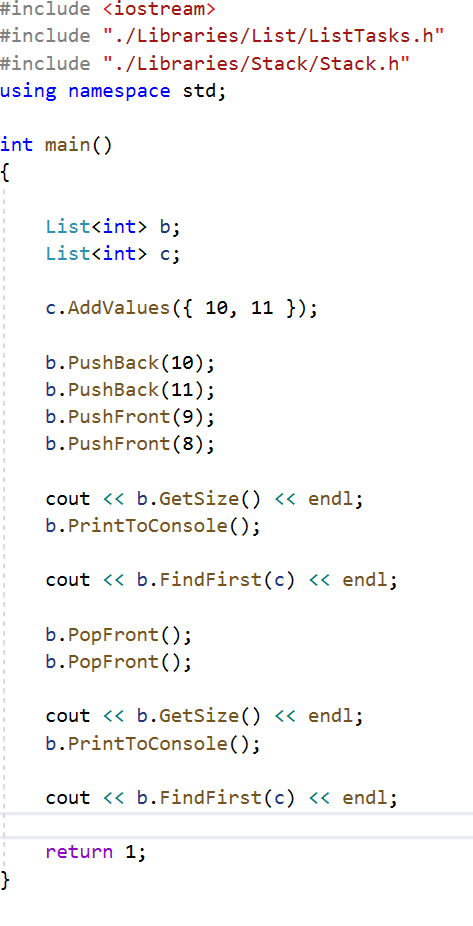
# Unit-тесты

Для проверки на работоспособность большого количества взаимосвязанных функций были реализованы юнит-тесты:

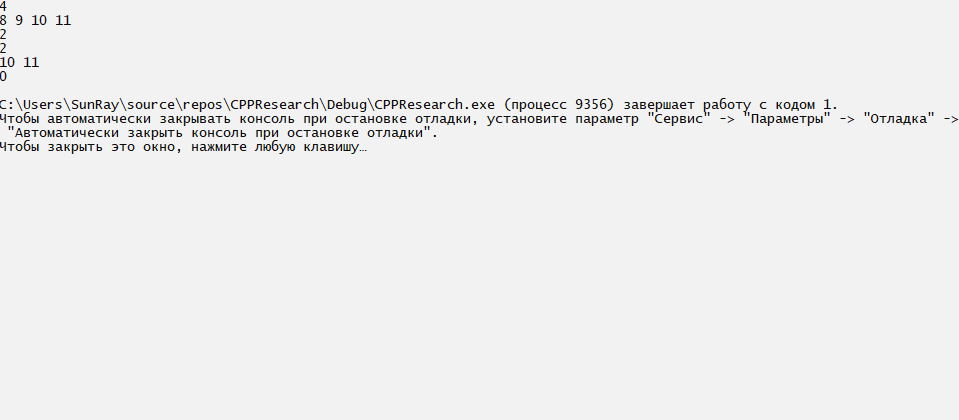


# Пример работы

## Функция main, представляющая пример использования написанной структуры данных:



## Результат выполнения:



# Текст программы:

|  |
| --- |
| #include <stdexcept>  template <class T>  class Node{  Node<T>\* next; T data;  public:  ~Node() {}  Node() : next(nullptr) {}  Node(T value) : data(value), next(nullptr) {}  void SetNext(Node<T>\* nextElement) {next = nextElement;}  void SetData(T newData){data = newData;}  Node\* GetNext() {return next;}  T GetData() {return data;}};  template <class T>  class List{  size\_t curElement; size\_t count; Node<T> \*head, \*tail, \*current, \*previous;  private:  void SetHeadNode(Node<T>\* n) { head = n; }  void SetTailNode(Node<T>\* n) { tail = n; }  void SetCurrentNode(Node<T>\* n) { current = n; }  void SetCurrentValue(T n) { current->SetData(n); }  void SetPreviousNode(Node<T>\* n) { previous = n; }  Node<T>\* GetHeadNode() { return head; }  Node<T>\* GetTailNode() { return tail; }  Node<T>\* GetCurrentNode() { return current; }  T GetCurrentValue() { return GetCurrentNode()->GetData(); }  Node<T>\* GetPreviousNode() { return previous; }  Node <T>\* GetNode(size\_t i){  if (i<0 || i > count - 1)  return nullptr;  if (curElement > i)  MoveToHead();  for (size\_t j = curElement; j < i; j++)  MoveForward();  return GetCurrentNode();}  void MoveForward(){  SetPreviousNode(GetCurrentNode());  SetCurrentNode(GetCurrentNode()->GetNext());  curElement++;}  void MoveToHead(){  SetPreviousNode(nullptr); SetCurrentNode(head); curElement = 0;}  void MoveToTail(){  MoveToHead();  while (GetCurrentNode()->GetNext() != nullptr)  MoveForward();}  void CheckBounds(size\_t index){  if (index < 0 || index > count - 1 || count == 0)  throw std::exception("Index is out of bounds");}  public:  ~List() {Clear();}  List() : count(0), curElement(0), head(nullptr), tail(nullptr), current(nullptr), previous(nullptr) {}  template <class T>  void AddValues(std::initializer\_list<T> list){  for (auto elem : list)  PushBack(elem);}  void RemoveAll(T value){  for (size\_t i = 0; i < GetSize(); i++)  if (value == At(i)){Remove(i); i--;}}  void RemoveFirst(T value){  for (size\_t i = 0; i < GetSize(); i++)  if (value == At(i)){Remove(i); return;}}  T operator[] (size\_t index) { return At(index); }  //My variant - 7  /\*+\*/void PushFront(List<T>& value)  {  for (int i = value.GetSize() - 1; i >= 0 ; i--)  PushFront(value[i]);  }  /\*+\*/void PushBack(T value){  Node<T>\* toAdd = new Node<T>(value);  if(IsEmpty())  head = current = previous = tail = toAdd;  else{  tail->SetNext(toAdd);  tail = toAdd;  MoveToHead();}  count++;}  /\*+\*/void PushFront(T value){  Node<T>\* toAdd = new Node<T>(value);  if (IsEmpty())  head = current = previous = tail = toAdd;  else{  toAdd->SetNext(head);  head = toAdd;  MoveToHead();}  count++;}  /\*+\*/void PopBack(){Remove(count - 1);}  /\*+\*/void PopFront(){Remove(0);}  /\*+\*/void Insert(T value, size\_t index){  CheckBounds(index);  MoveToHead();  Node<T>\* previousNode = GetNode(index - 1);  Node<T>\* moveNode = GetNode(index);  Node<T>\* toInsert = new Node<T>(value);  if(previousNode != nullptr)  previousNode->SetNext(toInsert);  toInsert->SetNext(moveNode);  if (moveNode == head)  head = toInsert;  count++;  MoveToHead();  }  T At(size\_t index){  CheckBounds(index);  return GetNode(index)->GetData();}  /\*+\*/void Remove(size\_t index){  CheckBounds(index);  Node<T>\* previousNode = GetNode(index - 1);  Node<T>\* toDelete = GetNode(index);  Node<T>\* nextNode = GetNode(index + 1);  if (toDelete == head)  head = nextNode;  if (toDelete == tail)  tail = previousNode;  if (previousNode != nullptr)  previousNode->SetNext(nextNode);  MoveToHead(); count--; delete toDelete;}  /\*+\*/size\_t GetSize(){return count;}  /\*+\*/void Clear(){  if (!IsEmpty()){  MoveToHead();  do{  MoveForward();  delete GetPreviousNode();  } while (GetCurrentNode() != nullptr);} count = 0;}  /\*+\*/void Replace(T value, size\_t index){  CheckBounds(index);GetNode(index)->SetData(value);}  /\*+\*/bool IsEmpty(){return count < 1;}  /\*+\*/size\_t FindFirst(List<T>& list){  size\_t firstLength = GetSize(); size\_t secondLength = list.GetSize();  if ((secondLength > firstLength) || (secondLength == 0 || firstLength == 0))  return -1;  for (size\_t i = 0; i < firstLength; i++){  for (size\_t j = 0; j < secondLength; j++){  if (firstLength - i < secondLength)  return -1;  T val1 = At(i);  T val2 = list.At(j);  if (val1 == val2){  bool success = true;  for (size\_t el = 1; el < secondLength - j; el++){  T val11 = At(i + el);  T val22 = list.At(j + el);  if (val11 != val22){  success = false;  break;}}  if (success)  return i;}  else  break;}}}  void PrintToConsole(){  MoveToHead();  for (size\_t i = 0; i < GetSize(); i++){  std::cout << GetCurrentValue() << ' '; MoveForward();}  std::cout << std::endl;}  void SelfCheck(){  MoveToHead();  for (size\_t i = 0; i < GetSize(); i++){  T val = GetCurrentValue();  MoveForward();}}}; |

# Вывод

В результате выполнения данной работы был реализован однонаправленный список, получен опыт написания юнит-тестов и оценки временной сложности реализуемых функций.