МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет

Практическая работа №5

Дисциплина Структуры и алгоритмы обработки данных

Тема. Структуры хранения линейных структур данных. Однонаправленный динамический список.

Выполнил студент Дамарад Д. В.

Фамилия И.О.

Группа ИКБО-13-21

Номер группы

Москва 2022

Вопросы

1) Внешний – как представляет данные пользователь.
   Промежуточный – как представляет данные разработчик.
   Физический – как данные размещаются ОС.
2) Тип данных определяет размер, выделяемый в памяти под объект, и операции над объектом.
3) Структура данных определяет совокупность логически связанных элементов данных, между которыми существуют отношения.
4) Данные могут располагаться последовательно в памяти, а могут располагаться в куче.
5) Линейная структура данных – структура данных, где элементы упорядочены по положению и доступ к элементу прямой.
6) Линейный список – структура данных, где элементы упорядочены по положению.
7) Стек – структура данных, основанная на принципе LIFO (последний зашел, первый вышел), где все операции применяются к последнему элементу.
8) Очередь – структура данных, основанная на принципе FIFO (первый зашел, первый вышел), где все операции применяются к первому элементу.
9) Стек отличается от линейного списка тем, что все операции применимы только к последнему элементу.
10) Стек.
11) Массив: O(n); Список: O(1)
12) Массив: O(n); Список: O(1)
13) Трюк Вирта - Удалить узел под заданным номер и предшествующему ему узлу присвоить указатель на следующий узел после удаляемого.
14) Структура узла однонаправленного списка состоит из переменной для хранения значения и из указателя на следующий элемент.
15) Node<T>* temp = head;

```
while (temp->next_ptr != nullptr)
{
        cout << temp->value << " ";
        temp = temp->next_ptr;
}
cout << temp->value;
```

16)
```
Node<T>* temp = head;
Node<T>* temp2 = temp;
while (temp->next_ptr->next_ptr != nullptr)
{
        temp = temp->next_ptr;
}
temp2 = temp->next_ptr;
temp->next_ptr = nullptr;
temp2->next_ptr = head;
head = temp2;
```

17) Лишнее действие – проверка узла на nullptr.

Новый узел вставляется в начало.

1. Условие задания:

Условие варианта: реализовать программу решения задачи варианта по использованию линейного однонаправленного списка. Тип информационной части узла – char

2. Список операций над списком, выявленный в процессе исследования дополнительных задач задания:

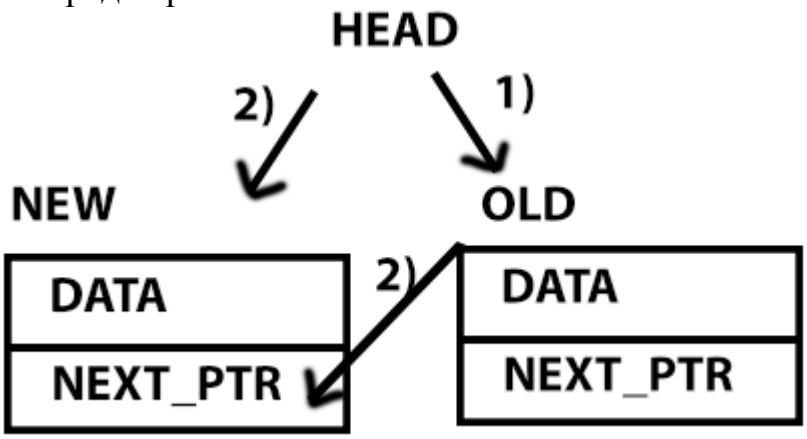
1. Вставка узла перед первым элементом.
2. Проверка списков на равенство.
3. Вставка в список №1 последнего элемента списка №2.
4. Удаление из списка №2, узлов, содержащие цифровые значения.

2.1 Структура узла:

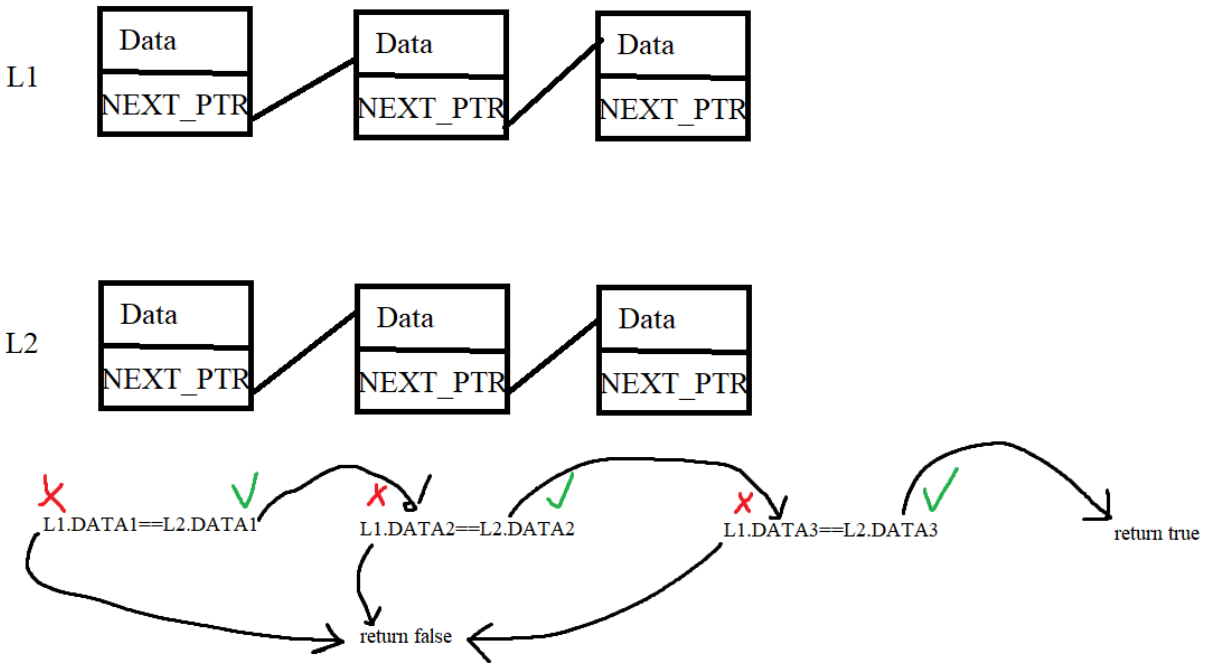
```
template<typename T>
class Node {
public:
	Node* pNext;
	T data;
	Node(T data = T(), Node* pNext = nullptr) {
		this->data = data;
		this->pNext = pNext;
	}
};
int	size;
Node<T>* head;
};
```

2.2 Изображение процесса выполнения операции:

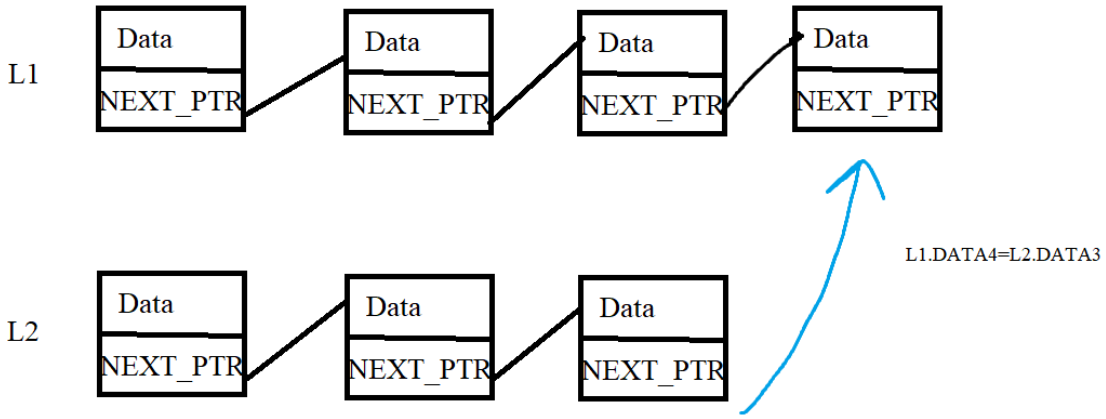
1) Вставка узла перед первым элементом



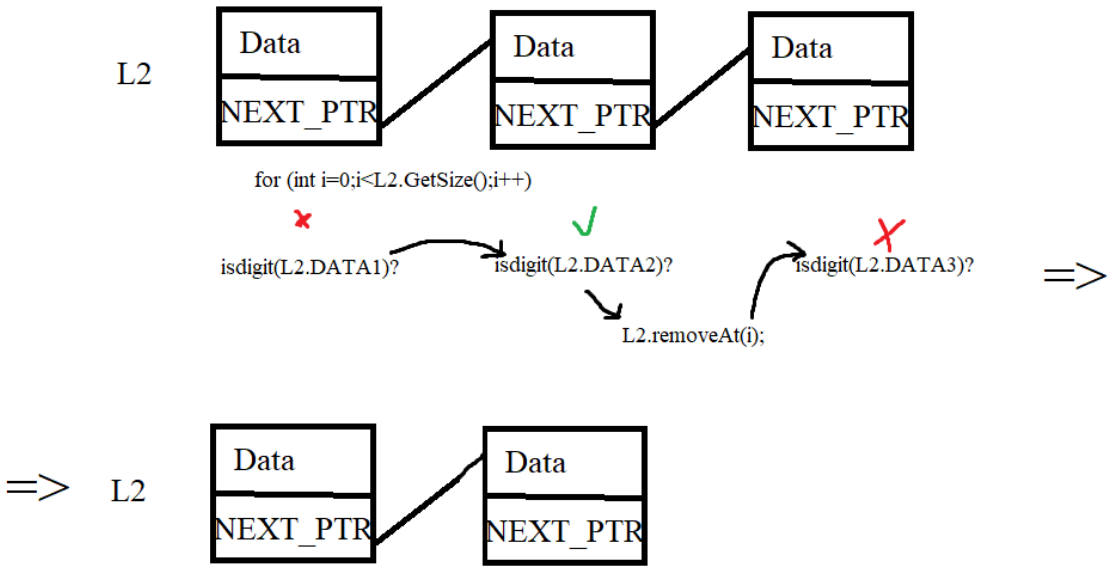
2) Проверка списков на равенство:



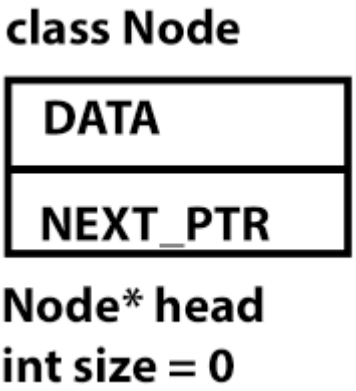
3) Вставка в список №1 последнего элемента списка №2:



4) Удаление из списка №2, узлов, содержащие цифровые значения:



2.3 Структура данных, которая используется в операциях:



2.4 Алгоритм выполнения операции:

1) Вставка узла перед первым элементом:

```
void List<T>::push_front(T data) {
    head = new Node<T>(data, head);
    size++;
}
```

2) Проверка списков на равенство:

```
template<typename T>
bool compare(List<T>& list1, List<T>& list2) {
    for (int i = 0; i < list1.GetSize(); i++) {
        if (list1[i] != list2[i]) {
            return false;
            break;
        }
    }
    return true;
}
```

3) Вставка в список №1 последнего элемента списка №2:

```
template<typename T>
void push_back_before_list2_last(List<T>& list1, List<T>& list2) {
    list1.push_back(list2[list2.GetSize() - 1]);
}
```

4) Удаление из списка №2, узлов, содержащие цифровые значения:

```
template<typename T>
 void delete_numbers(List<T>& list) {
 for (int i = 0; i < list.GetSize(); i++) {
            if (isdigit(list[i])) {
```

```
                list.removeAt(i);
            }
    }
    }
```

2.5 Таблица тестов для каждой операции:

1) Вставка узла перед первым элементом

Номер теста	Входные данные	Полученный результат	Ожидаемый результат	Пройден/ не пройден
1	L1 : Z X C NEW EL: V	V Z X C	V Z X C	Пройден

2) Проверка списков на равенство

Номер теста	Входные данные	Полученный результат	Ожидаемый результат	Пройден/ не пройден
1	L1 : Z X C L2 : A S D	false	false	Пройден
2	L1 : Z X C L2 : Z X C	true	true	Пройден

3) Вставка в список №1 последнего элемента списка №2

Номер теста	Входные данные	Полученный результат	Ожидаемый результат	Пройден/ не пройден
1	L1 : Z X C L2 : A S D	L1 : Z X C D	L1 : Z X C D	Пройден
2	L1 : 1 2 3 L2 : 4 5 6	L1 : 1 2 3 4	L1 : 1 2 3 4	Пройден

4) Удаление из списка №2, узлов, содержащие цифровые значения:

Номер теста	Входные данные	Полученный результат	Ожидаемый результат	Пройден/ не пройден
1	L2 : 1 S 2	L2 : S	L2 : S	Пройден
2	L2 : Z 1 C	L2 : Z C	L2 : Z C	Пройден

3. Код программы:

```
#include <iostream>
using namespace std;

template<typename T>
class List {
public:
  List(); //конструктор
  ~List(); //деструктор
  void pop_front(); //удаление первого элемента в списке
  void push_back(T data); //добавление элемента в конец списка
  void clear(); // очистить список
  int GetSize() { return size; } // получить количество елементов в списке
  T& operator[](const int index); // перегруженный оператор []
  void push_front(T data); //добавление элемента в начало списка
  void insert(T data, int index); //добавление элемента в список по
указанному индексу
  void removeAt(int index); //удаление элемента в списке по указанному
индексу
  void pop_back(); //удаление последнего элемента в списке
  void complete_list(List<T>& list, int size, int n); //заполнение списка
  void print_list(List<T>& list, int n); //вывод списка

private:
  template<typename T>
  class Node {
  public:
        Node* pNext;
        T data;
        Node(T data = T(), Node* pNext = nullptr) {
              this->data = data;
              this->pNext = pNext;
        }
  };
  int     size;
  Node<T>* head;
};

template<typename T>
List<T>::List() {
  size = 0;
  head = nullptr;
}
```

```
template<typename T>
List<T>::~List() {
  clear();
}

template<typename T>
void List<T>::pop_front() {
  Node<T>* temp = head;
  head = head->pNext;
  delete temp;
  size--;
}

template<typename T>
void List<T>::push_back(T data) {
  if (head == nullptr) {
        head = new Node<T>(data);
  }
  else {
        Node<T>* current = this->head;

        while (current->pNext != nullptr) {
              current = current->pNext;
        }
        current->pNext = new Node<T>(data);
  }
  size++;
}

template<typename T>
void List<T>::clear() {
  while (size) {
        pop_front();
  }
}

template<typename T>
T& List<T>::operator[](const int index) {
  int counter = 0;
  Node<T>* current = this->head;
  while (current != nullptr) {
        if (counter == index) {
```

```
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}

template<typename T>
void List<T>::push_front(T data) {
    head = new Node<T>(data, head);
    size++;
}

template<typename T>
void List<T>::insert(T data, int index){
    if (index == 0) {
        push_front(data);
    }
    else {
        Node<T>* previous = this->head;

        for (int i = 0; i < index - 1; i++) {
            previous = previous->pNext;
        }
        Node<T>* newNode = new Node<T>(data, previous->pNext);
        previous->pNext = newNode;
        size++;
    }
}

template<typename T>
void List<T>::removeAt(int index) {
    if (index == 0) {
        pop_front();
    }
    else {
        Node<T>* previous = this->head;
        for (int i = 0; i < index - 1; i++) {
            previous = previous->pNext;
        }
        Node<T>* toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;
        delete toDelete;
        size--;
```

```
  }
}

template<typename T>
void List<T>::pop_back() {
  removeAt(size - 1);
}

template<typename T>
void List<T>::complete_list(List<T>& list, int size, int n){
  T data;
  cout << "Ввод списка №" << n << ": ";
  for (int i = 0; i < size; i++) {
        cin >> data;
        list.push_back(data);
  }
}

template<typename T>
void List<T>::print_list(List<T>& list, int n) {
  cout << "Список №"<< n <<": ";
  for (int i = 0; i < list.GetSize(); i++) {
        cout << list[i] <<" ";
  }
  cout << endl;
}

template<typename T>
void push_back_before_list2_last(List<T>& list1, List<T>& list2) {
  list1.push_back(list2[list2.GetSize() - 1]);
}

template<typename T>
bool compare(List<T>& list1, List<T>& list2) {
  for (int i = 0; i < list1.GetSize(); i++) {
        if (list1[i] != list2[i]) {
              return false;
              break;
        }
  }
  return true;
}

template<typename T>
```

```
void delete_numbers(List<T>& list) {
  for (int i = 0; i < list.GetSize(); i++) {
        if (isdigit(list[i])) {
              list.removeAt(i);
        }
  }
}

int main() {
  setlocale(LC_ALL, "ru");
  int size, n=0;
  cout << "Введите размер списков: ";
  cin >> size;
  List<char> list1;
  List<char> list2;
  list1.complete_list(list1, size, 1);
  list1.print_list(list1, 1);
  list1.complete_list(list2, size, 2);
  list1.print_list(list2, 2);
  cout << "Введите номер задачи: " << endl;
  cout << "1) Проверка списков на равенство." << endl;
  cout << "2) Вставка в список №1 последнего элемента списка №2." <<
endl;
  cout << "3) Удаление из списка №2, узлов, содержащие цифровые
значения." << endl;
  cin >> n;
  switch (n) {
  case 1: {
        if (compare(list1, list2)) {
              cout << "Списки равны" << endl;
        }
        else {
              cout << "Списки не равны" << endl;
        }
        break;
  }
  case 2: {
        push_back_before_list2_last(list1, list2);
        cout << endl << "Внесены изменения в список №1" << endl;
        list1.print_list(list1, 1);
        break;
  }
  case 3: {
        delete_numbers(list2);
```

```
            cout << endl << "Внесены изменения в список №2" << endl;
            list2.print_list(list2, 2);
            break;
    }
    default: {
            break;
    }
    }
    return 0;
}
```

4. Результат тестирования программы:

1) Проверка списков на равенство:

```
Введите размер списков: 3
Ввод списка №1: Z X C
Список №1: Z X C
Ввод списка №2: Z X C
Список №2: Z X C
Введите номер задачи:
1) Проверка списков на равенство.
2) Вставка в список №1 последнего элемента списка №2.
3) Удаление из списка №2, узлов, содержащие цифровые значения.
1
Списки равны
```

```
Введите размер списков: 3
Ввод списка №1: Z X C
Список №1: Z X C
Ввод списка №2: A S D
Список №2: A S D
Введите номер задачи:
1) Проверка списков на равенство.
2) Вставка в список №1 последнего элемента списка №2.
3) Удаление из списка №2, узлов, содержащие цифровые значения.
1
Списки не равны
```

2) Вставка в список №1 последнего элемента списка №2:

```
Введите размер списков: 3
Ввод списка №1: Z X C
Список №1: Z X C
Ввод списка №2: A S D
Список №2: A S D
Введите номер задачи:
1) Проверка списков на равенство.
2) Вставка в список №1 последнего элемента списка №2.
3) Удаление из списка №2, узлов, содержащие цифровые значения.
2

Внесены изменения в список №1
Список №1: Z X C D
```

3) Удаление из списка №2, узлов, содержащие цифровые значения:

```
Введите размер списков: 3
Ввод списка №1: Z X C
Список №1: Z X C
Ввод списка №2: 1 Z 2
Список №2: 1 Z 2
Введите номер задачи:
1) Проверка списков на равенство.
2) Вставка в список №1 последнего элемента списка №2.
3) Удаление из списка №2, узлов, содержащие цифровые значения.
3

Внесены изменения в список №2
Список №2: Z
```

5. Выводы:

В ходе проделанной работы была получены навыки создания структуры односвязного списка, а также усвоен принцип его работы

6. Список информационных источников:

Отсутствуют.