|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Кафедра прикладной математики, механики, управления и программного обеспечения** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9121-09.03.04прогин | | | |
|  |  |  | | Шевелёв Р.В. | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2022 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать пакет подпрограмм для работы с упорядоченным двусвязным кольцевым (циклическим) динамическим списком, включающего в себя повторы, для которых нужно хранить количество, элементы списка – целые числа.  
Основные операции:  
1. Инициализация  
2. Освобождение памяти  
3. Добавление элемента по значению

4. Удаление элемента по значению

5. Поиск заданного по значению элемента   
6. Печать списка  
7. Объединение двух списков

**2 Описание типа + спецификация подпрограмм + тесты**

**Type** List\_Elem = ^Elem;

List\_Elem^.next – указатель на следующий элемент списка

List\_Elem^.prev – указатель на предыдущий элемент списка

data – целочисленный тип данных, поле данных элемента в списке

count – целочисленный тип данных, количество повторов данного элемента в списке

**procedure** printList(**var** head: List\_Elem);

Выводит в консоль все значения полей data с количеством повторов для каждого элемента типа List\_Elem, представленных полями count.

Входные данные: указатель на голову head списка.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Печать пустого списка | Пустой список | Операция не выполняется |
| Печать непустого списка | Список: 1(1) 3(10) 9(1) | 1(1) 3(10) 9(1) |
| Печать непустого списка | Список: 3(1) 10(4) 19(2) | 3(1) 10(4) 19(2) |

**procedure** Free(**var** head: List\_Elem);

Освобождение памяти, удаляет список начиная с указателя на head.

Входные данные: указатель на голову head списка.

Выходные данные: пустой список

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление пустого списка | Пустой список | Операция не выполняется |
| Удаление непустого списка | Список: 1(1) 3(10) 9(1) | Пустой список |
| Удаление непустого списка | Список: 3(1) 10(4) 19(2) | Пустой список |

procedure Delete\_Element(var head: List\_Elem; data: integer);

Если элемент уникальный, то функция удаляет все поля элемента списка типа List\_Elem с головой head, а также сам элемент, очищая память.

Если элемент неуникальный, то функция находит элемент списка типа List\_Elem с головой head и полем data равным data и декрементирует поле count.

Входные данные: список, значение.

Выходные данные: список

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление элемента в  пустом списке | Пустой список  Элемент: 1 | Операция не выполняется |
| Удаление уникального элемента в непустом списке, где данный элемент присутствует. | Список: 1(1) 3(10) 9(1)  Элемент: 1 | Список: 3(10) 9(1) |
| Удаление уникального элемента в непустом списке, где данный элемент отсутствует. | Список: 3(1) 10(4) 19(2)  Элемент: -1 | Список: 3(1) 10(4) 19(2) |
| Удаление повторяющегося элемента в непустом списке, где данный элемент присутствует | Список: 3(1) 10(4) 19(2)  Элемент: 10 | Список: 3(1) 10(3) 19(2) |

**procedure** Union(**var** head1: List\_Elem; **var** head2: List\_Elem; **var** head: List\_Elem);

Процедура выполняет операцию объединения двух списков с головами head1 и head2 типа List\_Elem

Входные данные: список, список.

Выходные данные: список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Объединение пустых списков | Пустой список, пустой список | Операция не выполняется |
| Объединение двух списков, один из них пустой | Список1: пустой  Список2: 1(1) 3(10) 9(1) | Операция не выполняется |
| Объединение двух непустых списков | Список1: -1(1) 3(2) 100(1)  Список2: 3(1) 10(4) 19(2) | Список3: -1(1) 3(3) 10(4) 19(2) 100(1) |

procedure Add(var head: List\_Elem; data: integer);

Если элемент не повторяется, то добавляет в нужное место списка (так как он упорядоченный) с головой head типа List\_Elem элемент типа List\_Elem со значением поля data равное data. Если элемент повторяется то, находит в списке с головой head типа List\_Elem элемент со значением переменной data типа int и инкрементирует его поле count.

Входные данные: список, значение.

Выходные данные: список

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление элемента в пустой список | Пустой список  Элемент: 1 | Список: 1(1) |
| Добавление элемента в непустой список | Список: 1(1) 3(10) 9(1)  Элемент: 10 | Список: 1(1) 3(10) 9(1) 10(1) |
| Добавление повторяющегося  элемента в непустой список | Список: 3(1) 10(4) 19(2)  Элемент: 10 | Список: 3(1) 10(5) 19(2) |

function Find\_Element(var head: List\_Elem; data: integer): boolean;

Находит в списке с головой head типа List\_Elem элемент со значением переменной data типа int и возвращает true, если элемент в списке есть, и false, если он в нём отсутствует. Входные данные: список, значение. Выходные данные: true/false.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Поиск элемента в  пустом списке | Пустой список  Элемент: 1 | false |
| Поиска элемента в списке, где присутствует заданный  элемент | Список: 1(1) 3(10) 9(1)  Элемент: 3 | true |
| Поиска элемента в списке, где отстутствует заданный  элемент | Список: 3(1) 10(4) 19(2)  Элемент: 0 | false |

**3 Текст программы**

**type**

List\_Elem = ^Elem;

Elem = **record**

prev: List\_Elem;

data: integer;

count: integer;

next: List\_Elem;

**end**;

**var**

head1, head2, head3: List\_Elem;

**procedure** printList(**var** head: List\_Elem);

**var**

elem: List\_Elem;

**begin**

**if** head <> nil **then begin**

write('...⟷');

write('[', head^.data, ']', '(', head^.count, ')', head, '⟷');

elem := head;

**while** (elem^.next <> head) **do**

**begin**

elem := elem^.next;

write('[', elem^.data, ']', '(', elem^.count, ')', elem, '⟷');

**end**;

writeln('...');

**end**;

**end**;

**procedure** Free(**var** head: List\_Elem);

**var**

current, Del: List\_Elem;

**begin**

**if** head <> nil **then begin**

current := head;

**while** (current^.next <> head) **do**

**begin**

Del := current;

current := current^.next;

dispose(Del);

head^.prev^.next := current;

current^.prev := head^.prev;

head := current;

printlist(head);

**end**;

dispose(head);

**end**;

**end**;

**procedure** Delete\_Element(**var** head: List\_Elem; data: integer);

**var**

current: List\_Elem;

**begin**

**if** head <> nil **then begin**

current := head;

**while** (current^.data <> data) **and** (current^.next <> head) **do** current := current^.next;

**if** current^.data = data **then begin**

**if** current^.count > 1 **then begin**

current^.count -= 1;

**end**

**else begin**

**if** current = head **then** head := head^.next;

current^.next^.prev := current^.prev;

current^.prev^.next := current^.next;

dispose(current);

**end**;

**end**;

**end**;

**end**;

**procedure** Union(**var** head1: List\_Elem; **var** head2: List\_Elem; **var** head: List\_Elem);

**var**

insert, cur1, cur2: List\_Elem;

**begin**

**if** (head1 <> nil) **and** (head2 <> nil) **then begin**

cur1 := head1;

cur2 := head2;

**while** (cur1^.next <> head1) **and** (cur2^.next <> head2) **do**

**begin**

**new**(insert);

insert^.data := min(cur1^.data, cur2^.data);

**if** head = nil **then begin**

head := insert;

head^.next := head;

head^.prev := head;

**end**;

**if** (cur1^.data < cur2^.data) **then begin**

insert^.count += cur1^.count;

cur1 := cur1^.next;

**end**

**else begin**

**if** (cur1^.data = cur2^.data) **then begin**

insert^.count += cur1^.count + cur2^.count;

cur1 := cur1^.next;

cur2 := cur2^.next;

**end**

**else begin**

**if** (cur1^.data > cur2^.data) **then begin**

insert^.count += cur2^.count;

cur2 := cur2^.next;

**end**;

**end**;

**end**;

**if** head <> insert **then begin**

insert^.next := head;

insert^.prev := head^.prev;

head^.prev^.next := insert;

head^.prev := insert;

**end**;

**end**;

**new**(insert);

insert^.data := min(cur1^.data, cur2^.data);

**if** (cur1^.data < cur2^.data) **then** insert^.count += cur1^.count

**else begin**

**if** (cur1^.data = cur2^.data) **then** insert^.count += cur1^.count + cur2^.count

**else if** (cur1^.data > cur2^.data) **then** insert^.count += cur2^.count

**end**;

insert^.next := head;

insert^.prev := head^.prev;

head^.prev^.next := insert;

head^.prev := insert;

**end**;

**end**;

**procedure** Add(**var** head: List\_Elem; data: integer);

**var**

current, insert: List\_Elem;

**begin**

**new**(insert);

insert^.data := data;

insert^.count += 1;

**if** (head = nil) **then begin**

head := insert;

head^.next := head;

head^.prev := head;

**end**

**else begin**

current := head;

**if** (head^.prev^.data < data) **or** (head^.data > data) **then begin**

insert^.next := head;

insert^.prev := head^.prev;

head^.prev^.next := insert;

head^.prev := insert;

**if** (head^.data > data) **then** head := insert;

**end**

**else begin**

**while** (current^.next <> head) **and** (current^.data < data) **do** current := current^.next;

**if** current^.data = data **then** current^.count += 1

**else begin**

insert^.next := current;

insert^.prev := current^.prev;

current^.prev^.next := insert;

current^.prev := insert;

**end**;

**end**;

**end**;

**end**;

**function** Find\_Element(**var** head: List\_Elem; data: integer): boolean;

**var**

elem: List\_Elem;

**begin**

**if** head <> nil **then begin**

elem := head;

**while** (elem^.next <> head) **do**

**begin**

**if** (elem^.data = data) **then** Find\_Element := True;

elem := elem^.next;

**end**;

**if** (elem^.data = data) **then** Find\_Element := True;

**end**;

**end**;

**begin**

**for var** i := 1 **to** 20 **do**

**begin**

add(head1, i);

add(head1, i);

add(head2, i);

Add(head2, -100);

Add(head2, -100);

**end**;

print(Find\_Element(head1, -49));

print(Find\_Element(head1, -4));

print(Find\_Element(head1, 1));

writeln;

Add(head1, -100);

Add(head1, -100);

Add(head1, -100);

Add(head1, -100);

printlist(head1);

writeln;

printlist(head2);

writeln;

Union(head1, head2, head3);

writeln;

PrintList(head3);

**end**.