МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Отчет по выполнению практического задания №6

Тема: Однонаправленный динамический список

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент группы ИКБО-01-20 Антонов А.Д.

Москва 2021

СОДЕРЖАНИЕ

1. Ответы на вопросы

1.1. Три уровня представления данных в программе

Существует три уровня представления данных: уровень пользователя (предметная область), логический и физический.

1.2. Что определяет тип данных?

Тип определяет возможные значения и их смысл, операции, а также способы хранения значений типа.

1.3. Что определяет структура данных?

Под структурой данных программ в общем случае понимают множество элементов данных, множество связей между ними, а также характер их организованности.

1.4. Структура данных в комльютерных технологиях

Структура данных — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

1.5. Определение линейной структуры данных

Линейные структуры — это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером.

1.6. Определение линейного списка, как структуры данных

Линейный однонаправленный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством 5

указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на NULL. Элемент, на который нет указателя, является первым (головным) элементом списка. Здесь ссылка в каждом узле указывает на следующий узел в списке. В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

1.7. Определение стека, как структуры данных

Логически стек представляет собой последовательность элементов с переменной длиной; включение и исключение элементов из последовательности происходит при этом только с одной стороны, называемой вершиной стека.

1.8. Определение очереди, как структуры данных

Очередь — это структура данных, добавление и удаление элементов в которой происходит путём операций и соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел».

1.9. Отличие стека и линейного списка

Стек – это линейный список, в котором добавление новых элементов и удаление существующих производится только с одного конца, называемого вершиной стека.

1.10. Какой из видов линейных списков лучше использовать, если введенную последовательность нужно вывести наоборот

Для решения такой задачи удобнее использовать двунаправленный список, потому что в нем есть возможность пройти список как с начала, так и с конца.

1.11. Определение сложности алгоритма вставкавми элемента в i-ую позицию

Вставку элемента в i-ую позицию можно разделить на две операции: поиск іго элемента и саму вставку. Для массива поиск элемента по индексу не составляет труда, алгоритмическая сложность составляет O(1). В случае односвязного спика придётся последовательно перебрать все элементы, пока не доберёмся до нужного элемента. Сложность будет O(n). Вставка в массив связана со сдвигом всех элементов, находящихся после точки вставки, поэтому алгоритмическаясложность этой операции O(n). В односвязном списке вставка заключается в создании нового связующего объекта и установки ссылок на него у соседнихэлементов. Сложность O(1). В сумме сложность вставки i-го элемента у массива и у списка получается одинаковая — O(n). Но поскольку операция чтения по сути быстрее операции записи, односвязный список работает быстрее.

1.12. Определение сложности алгоритма удаления элемента из i-ой позиции

Удаление элемента из i-ой позиции можно разделить на две операции: поиск i-го элемента и удаление. Для массива поиск элемента по индексу не составляет труда, алгоритмическая сложность составляет O(1). В случае односвязного спика придётся последовательно перебрать все элементы, пока не доберёмся до нужного элемента. Сложность будет O(n). Удаление из массива связана со сдвигом всех элементов, находящихся после точки вставки, поэтому алгоритмическаясложность этой операции O(n). В односвязном списке удаление заключается в переустановке ссылок соседних элементов. Сложность O(1). В сумме сложность удаление i-го элемента у массива и у списка получается одинаковая — O(n). Но поскольку7 операция чтения по сути быстрее операции записи, односвязный список работает быстрее.

1.13. Трюк Вирта при выполнении операции удаления элемента ю списка

Косвенный указатель p даёт два концептуальных преимущества: 1. Позволяет интерпретировать связный список таким образом, что указатель head становится неотъемлемой частью структуры данных. Это устраняет необходимость в специальном случае для удаления первого элемента. 2. Также позволяет оценить состояние цикла while без необходимости отпускать указатель, указывающий на target. Это позволяет изменять указатель на target и обходиться одним итератором, в отличие от prev и cur.

1.14. Определение структуры узла однонаправленного списка

template<typename T> class Node T value; Node* next;

1.15. Алгоритм вывода линейного однонаправленного списка

Далее в работе

1.16. Перемещение последнего элемента в начало списка

Далее в работе

1.17. Какое ю действий лишнее в следующем фрагменте кода? Куда вставляется новый узел?

В этом коде лишней является ветка условного оператора с проверкой на нулевой указатель т.к. при обращении по нулевому указателю произойдет ошибка программы. Код вставляет новый узел в последующий после LL узел.

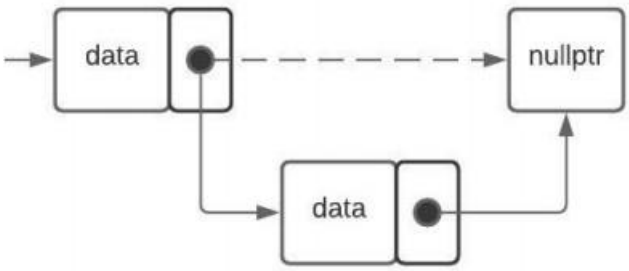
2. Отчет по разобранной программе

2.1. Постановка задачи

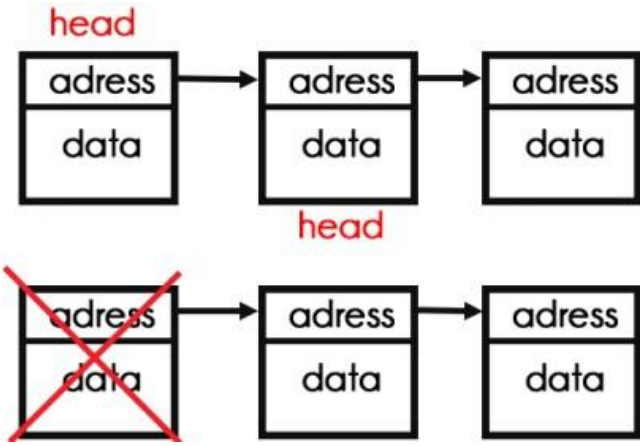
1	int	Даны два линейных однонаправленных списка L1 и L2. 1) Разработать функцию, которая формирует список L, включив в него по одному разу элементы, значения которых входят хотя бы в один из списков L1 и L2. 2) Разработать функцию, которая удаляет из списка L1 все узлы в четных позициях. 3) Разработать функцию, которая вставляет в список L2 после каждой пары узлов новый узел со значением равным сумме значений двух предыдущих узлов. Если количество узлов в исходном списке нечетное, то после последнего узла новый узел не вставлять

2.2. Определение операций над списком

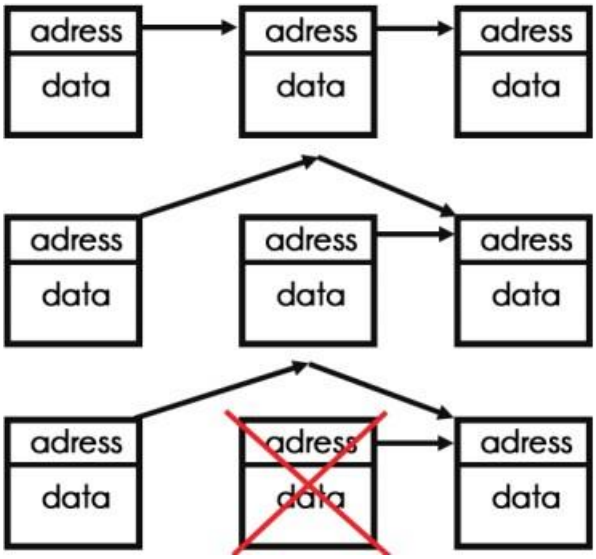
- Метод, добавляющий новый элемент в конец списка



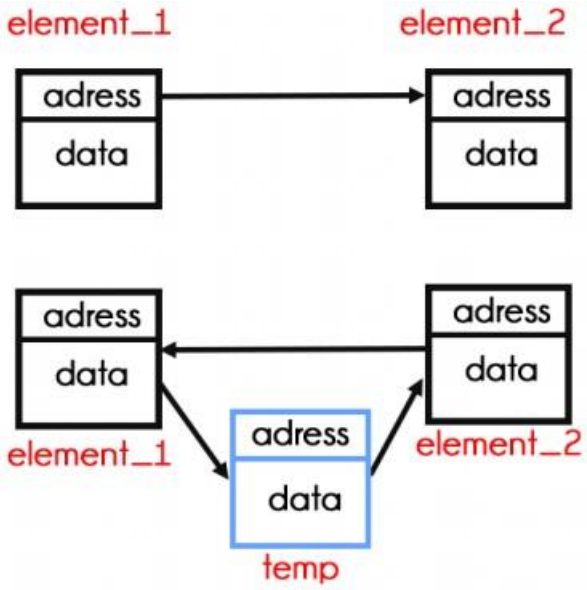
- Метод, удаляющий первый элемент в списке



- Метод, удаляющий элемент по индексу



- Метод, меняющий элементы списка местами



- Функция, реализующая заполнение списка L согласно задаче:
  - Аргументами функции являются указатели на первые элементы исходных списков L1 и L2, и результирующего списка L. В функции проходим в цикле по всем элементам списков L1 и L2. Для каждого элемента проверяем, есть ли элемент с таким значением в результирующем списке

L, и, если такого элемента нет, добавляет его в конец списка. Проверка делается с помощью дополнительной функции.

- Функция проверки имеет булевое возвращаемое значение и имеет два аргумента: целочисленное значение, на наличие которого проверяется список, и указатель на первый элемент проверяемого списка. В цикле проходим по всем элементам списка и проверяем, совпадает ли значение с аргументом. Если совпало, то возвращаем ложь, после завершения цикла возвращаем истину.
- Код функции:

```
58  // функция 1 (список со значениями двух исходных списков)
59  void createSetList(Node *head, Node *list1, Node *list2)
60  {
61      Node *node = list1;
62      int data;
63      while (node != nullptr) {
64          data = node->data;
65          if (checkIfUnique(data, head))
66              insertNode(head, data, &nextNode);
67      }
68      node = list2;
69      while (node != nullptr) {
70          data = node->data;
71          if (checkIfUnique(data, head))
72              insertNode(head, data, &nextNode);
73      }
74  }
```

```
48  bool checkIfUnique(int data, Node *head)
49  {
50      Node *node = head;
51      while (node != nullptr) {
52          if (node->data == data)
53              return false;
54      }
55      return true;
56  }
```

- Тестирование:

```
Enter L1: 3 7 4 7 9
Enter L2: 2 3 4
L: 3 7 4 9 2
```

```
Enter L1: 1 2 0 4 5
Enter L2: 0 2 1
L: 1 2 0 4 5
```

```
Enter L1: 33 73 85 90
Enter L2: 21 54
L: 33 73 85 90 21 54
```

- Функция, выполняющая удаление четных позиций из списка L1
  - Аргументом функции является указатель на первый элемент списка L1. В цикле присваиваем полю, хранящему указатель на следующий узел, указатель на узел вперед, и далее в следующей итерации цикла делаем то же самое для этого следующего узла.
  - Код функции:

```
76      // функция 2 (удаление четных элементов)
77      void deleteEvenNodes(Node *head) {
78          Node *node = head;
79          while (node != nullptr)
80          {
81              node->nextNode = node->nextNode->nextNode;
82              node = node->nextNode;
83          }
84      }
```

  - Тестирование:

```
Enter L1: 1 2 3 4 5 6 7 8 9
L1: 1 3 5 7 9
```

```
Enter L1: 2 6 3 6 4 6 9 4
L1: 2 3 4 9
```

- Функция, вставляющая в список L2 новый элемент после каждых двух
  - Аргумент – указатель на первый элемент списка L2. В цикле проходим по двум элементам списка за итерацию, прибавляем их значения в переменную, обнуляемую в начале итерации, и вставляем после второго элемента новый элемент со значением, равным значению используемой переменной. Если встречаем нулевой указатель, то завершаем цикл, не добавляя очередного нового элемента.
  - Код:

```
86  // функция 3 (вставка элемента после каждых двух элементов)
87  void insertAfterTwo(Node *head) {
88      Node *node = head;
89      int sum;
90      while (node != nullptr)
91      {
92          sum += node->data;
93          node = node->nextNode;
94          sum += node->data;
95          node = node->nextNode;
96          if (node != nullptr)
97              insertBetweenNode(node, sum, head);
98      }
99  }
```

◦ Тестирование:

```
Enter L2: 1 2 3 4 5 6 7 8 9
L2: 1 2 3 3 4 7 5 6 11 7 8 15 9
```

```
Enter L2: 12 32 65 36 18 47
L2: 12 32 44 65 36 101 18 47 65
```

2.3. Используемая структура данных

```
3   // структура элемента списка
4   struct Node {
5       int data;
6       Node *nextNode;
7
8       Node() {};
9
10      Node (int data, Node *nextNode = nullptr):
11              data(data),
12              nextNode(nextNode) {}
13  };
```

2.4. Код программы

```
// функция вывода списка
void showList(Node *head) {
    Node *node = head;
    while (node != nullptr) {
        std::cout << node->data;
        node = node->nextNode;
    }
}

// функция вставки элемента в край списка
void insertNode(Node *previousNode, int data, Node **node) {
    *node = new Node(data);
    previousNode->nextNode = *node;
}

// функция вставки элемента
void insertBetweenNode(Node *previousNode, int data, Node **node) {
    *node = new Node(data, previousNode->nextNode);
    previousNode->nextNode = *node;
}
```

```
// функция создания списка
void createList(int length, int *dataArr, Node *head) {
    Node *node = nullptr;
    insertNode(head,  data: dataArr[1], &node);
    Node *previousNode = node;
    for (int i = 2; i < length; ++i) {
        Node *nextNode = nullptr;
        insertNode(previousNode, dataArr[i], &nextNode);
        previousNode = nextNode;
    }
}
```

```
101  int main() {
102      int l1, l2;
103      std::cout << "Length of list 1 = ";
104      std::cin >> l1;
105      std::cout << "Length of list 2 = ";
106      std::cin >> l2;
107
108      std::cout << "Data of nodes for list 1:" << std::endl;
109      int *dataArr1 = new int[l1];
110      for (int i = 0; i < l1; ++i)
111          std::cin >> dataArr1[i];
112      std::cout << "Data of nodes for list 2:" << std::endl;
113      int *dataArr2 = new int[l2];
114      for (int i = 0; i < l2; ++i)
115          std::cin >> dataArr2[i];
116
117      Node *list1 = new Node( data: dataArr1[0]);
118      Node *list2 = new Node( data: dataArr2[0]);
119      createList(l1, dataArr1, list1);
120      createList(l2, dataArr2, list2);
121
122      /*
123      showList(head);
124      isSameNode(head);
125      insertEvenNode(head);
126      showList(head);
127      deleteMax(&head);
128      showList(head);
129      */
130  }
```

ВЫВОДЫ

В ходе данной практической работы разработали структуру данных – однонаправленный динамический список, получили знания и практическиенавыки управления динамическим однонаправленным списком. Разработалиобязательные функции по управлению списком и функции согласно варианту индивидуального задания. Провели тестирование функций, результат тестирования подтвердил правильность работы функций.

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1) Теоретический материал по структурам и алгоритмам обработки данных.
2) Инвариант цикла https://ru.wikipedia.org/wiki/Инвариант_цикла
3) Лекционный материал по структуре и алгоритмам обработки данных Гданского Н.И.