***Лабораторная работа 6.***

**Организация и алгоритмы обработки линейных списков**

***Определение.*** *Линейный список — конечное и упорядоченное множество данных, для которого определены операции добавления и удаления элементов в произвольной точке множества с сохранением упорядоченности.*

Можно сказать, что линейный список представляет собой обобщение структур стеков и очередей. В отличие от элементов этих структур данных любой элемент линейного списка доступен без извлечения других элементов.

Линейный список можно определить как перечисление элементов .

По Д. Кнуту понятие «линейный список» отличается от понятия «список», представляющего собой древовидную структуру.

Примеры.

1. Очередь обработки прерываний с учётом различных приоритетов представляет собой линейный список.
2. В трансляторе для синтаксического анализа текста программы используются линейные списки (список идентификаторов в описании переменных; списки параметров переменных; списки полей в записи).

**1.1 Функциональная спецификация линейного списка**

Считая, что базовый тип элементов Т, можно определить наименование типа «линейный список» как ЛСТ . Тогда перечень допустимых операций будет иметь вид:

1. Создать\_список: ( ) ЛСТ.

2. Список\_пуст: ЛСТ ЛОГ.

3. Следующий: ТЛСТ  Т.

4. Добавление: ТТЛСТ ЛСТ.

5. Удаление: ТЛСТ ЛСТ.

6. Первый: ЛСТ Т.

Из этого перечня следует, что при создании линейного списка он должен быть пустым; проверка линейного списка на пустоту возвращает логическое значение. При добавлении требуется указывать не только добавляемый элемент, но и место включения нового элемента в списке («косой крест» означает операцию декартова произведения). Здесь возможны варианты: «перед каким» или «после какого» требуется вставить элемент. При удалении может быть указан удаляемый элемент или место удаления в списке.

Перечень операций функциональной спецификации следует дополнить перечислением свойств введённых операций. Свойства представляют собой проверки, которые должны выполняться при физической реализации. Количество таких проверок может быть различным. Оно определяется классом задач, где предполагается использование структуры данных.

**1.2 Физическая реализация линейного списка**

Обычно рассматривают следующие варианты физического представления:

1. **Сплошное представление без использования указателей.** Здесь имеется в виду отображение линейного списка на одномерный массив. Очевидным недостатком такого представления является необходимость перемещения части элементов списка при выполнении операций добавления и удаления.
2. **Сплошное представление с использованием указателей**. При таком представлении не требуется производить перенос элементов списка внутри массива размещения. За такое удобство требуется выделить память под дополнительный массив, где размещаются «указатели», в роли которых выступают индексы массива данных.
3. **Классическое цепное представление с использованием указателей**. В этом случае элементы списка соединяются в цепочку в области динамического размещения данных ( в «куче» ) с помощью указателей, включаемых в элементы цепочки. При этом усложняется процесс программирования, требуется память под указатели, однако не нужно заранее выделять память под массивы размещения.
   1. **Сплошное представление с использованием указателей**

Для такого представления надо описать два массива: для размещения данных (Данные) и для размещения указателей (Указатели). Кроме того, необходимы переменные, которые будут играть роль точки входа в структуру (Адр\_вх) и указателя на вершину стека свободных элементов (Стек). Если индексация массива начинается с нуля, то признаком окончания цепочки может служить значение нуль в массиве указателей.

Пример.

A B C **|**

Свободные элементы: 1 3 4 6 8 9 10.

|  |  |
| --- | --- |
| **10** |  |
| **9** |  |
| **8** |  |
| **7** | **A** |
| **6** |  |
| **5** | **C** |
| **4** |  |
| **3** |  |
| **2** | **B** |
| **1** |  |

|  |  |
| --- | --- |
| **10** | **0** |
| **9** | **10** |
| **8** | **9** |
| **7** | **2** |
| **6** | **8** |
| **5** | **0** |
| **4** | **6** |
| **3** | **4** |
| **2** | **5** |
| **1** | **3** |

Адр\_вх=7

Стек=1

Данные Указатели

Здесь массивы состоят из 10 элементов.

Стек свободных элементов не требует для своего размещения дополнительной памяти: он помещается в массиве указателей.

Вершина стека свободных элементов содержит индекс элемента массива данных, где будет помещён добавляемый элемент.

Если добавляемый элемент должен располагаться в начале списка, то начальное и конечное положение в массивах из шести элементов будет выглядеть следующим образом:

Исходное положение

|  |  |
| --- | --- |
| **6** |  |
| **5** |  |
| **4** | **A** |
| **3** |  |
| **2** |  |
| **1** | **B** |

|  |  |
| --- | --- |
| **6** | **0** |
| **5** | **6** |
| **4** | **1** |
| **3** | **5** |
| **2** | **3** |
| **1** | **0** |

Адр\_вх = 4

Стек = 2

Данные Указатели

Результирующее положение

|  |  |
| --- | --- |
| **6** |  |
| **5** |  |
| **4** | **A** |
| **3** |  |
| **2** | **X** |
| **1** | **B** |

|  |  |
| --- | --- |
| **6** | **0** |
| **5** | **6** |
| **4** | **1** |
| **3** | **5** |
| **2** | **4** |
| **1** | **0** |

Адр\_вх = 2

Стек = 3

Данные Указатели

Очевидно, были выполнены операции:

**Данные [Стек] = X;**

**t = Указатели [Стек];**

**Адр\_вх = Стек;**

**Стек = t;**

При добавлении в тело списка или в конец списка следует указать «текущий» элемент (устанавливается на элемент, за которым следует добавить новый элемент списка). Тогда фрагмент алгоритма имеет вид:

**Данные [Стек] = X;**

**t = Указатели [Стек];**

**Указатели [Стек] = Указатели [Текущий];**

**Указатели [Текущий] = Стек;**

**Стек = t;**

Аналогично записываются алгоритмы удаления элемента из головы списка и из хвоста.

*Замечание.* Следует проверять, не переполнится ли массив при добавлении элемента. При удалении элемента линейного списка надо убедится, что список не пуст.

* 1. **Цепное представление с использованием указателей**

Под цепным представлением понимают представление в динамически распределяемой области памяти (в «куче») в виде цепочки элементов, связанных с помощью указателей. В качестве точки входа в цепочку применяют указатель на голову. Описание типа соответствующего указателя на Турбо Паскале может быть дано в форме:

Type UKAZ=^ES;

ES=record

info: integer;

next: UKAZ {указатель на следующий элемент цепочки}

end;

Var r, q, H, NE: UKAZ; value: integer;]

{H – первый элемент, q – текущий}

Здесь указатель включён в структуру данных. Поле записи info имеет тип integer. Это означает, что в рассматриваемом примере тип данных, образующих линейный список, предполагается целочисленным. В случае другого типа данных поле info следует изменить.

Далее рассматривается вариант реализации операций с линейным списком при цепном представлении. При этом использованы процедуры и функция (реализация на языке Турбо Паскаль).

*1. Создание линейного списка (H- указатель на голову)*

Procedure Sozd\_Sp;

BEGIN

H := nil

END;

*Замечание.* На практике часто операцию создания линейного списка совмещают с внесением в него начальных данных.

*2. Добавление элемента (Добавляемый элемент – value, вставляется после элемента, на который указывает q).*

Procedure Dobav\_Sp;

BEGIN

New(NE);

NE^.info:= value;

NE^.next:=q^.next;

q^next:=NE;

END;

*Замечание*. Не предусмотрена возможность добавления перед началом списка.

*3. Удаление элемента, следующего за тем элементом, на который указывает q*

Procedure Udal\_Sp;

BEGIN

r := q^.next; {для сохранения доступа к фрагменту памяти, занимаемому удаляемым элементом}

q^.next := q^.next.^next; {удаление из цепочки}

r^.next:=nil; {завершение обхода}

dispose(r) {возврат памяти в «кучу»}

END;

Операцию удаления можно наглядно представить в форме:

*А). Исходное состояние*

H

q удаляемый

текущий

r

*B). Результирующее состояние*

H

q

r

*Замечание*. Не предусмотрена возможность удаления первого элемента списка (у него нет предшествующего).

*4. Добавление value в начало линейного списка*

procedure Dob\_HSp;

BEGIN

new(NE);

NE^.info:= value;

NE^.next:= H;

H:= NE

END;

***ЗАДАНИЕ***

**1). Разработать программу, демонстрирующую работу с линейным списком при сплошном и цепном представлении.**

**2). При сплошном представлении без использования указателей спроектировать счётчики перемещаемых элементов списка в массиве.**

**2). Предусмотреть реализацию операций функциональной спецификации в виде подпрограмм.**

**3). Добавить подпрограммы вывода всего списка, а также полного удаления списка.**

**4). Предусмотреть меню выбора варианта представления линейного списка, а также выбора операций с возможностью продолжения работы.**