**Лекция 10. Динамические структуры данных. линейные списки**

**линейные списки**

Для работы с динамическими структурами данных используются указатели. Указатели представляют собой специальный тип данных. Они принимают значения, равные адресам размещения в оперативной памяти соответствующих динамических переменных.

Списком называется структура данных, каждый элемент которой с помощью указателя связывается с последующим элементом. На самый элемент (голову списка) есть отдельный указатель.

Из определения следует, что каждый элемент списка содержит поле данных (оно может иметь сложную структуру) и поле ссылки на следующий элемент. После ссылки последнего элемента должно содержать пустой указатель (nil).

Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа.

**Однонаправленный (односвязный) список -** это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка (рис. 1). В последнем элементе указатель на следующий элемент равен NULL.



Рис. 1. Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

struct имья\_типу {информационное поле; адресное поле; };

где информационное поле - это поле каждого, ранее объявленного или стандартного, типа;

адресное поле - это указатель на объект того же типа, что и обусловлена ​​структура, в него записывается адрес следующего элемента списка.

К примеру:

struct Node {

int key; // информационное поле

Node \* next; // адресное поле

};

Информационных полей может быть несколько.

К примеру:

struct point {

char \* name; // информационное поле

int age; // информационное поле

point \* next; // адресное поле

};

Каждый элемент списка содержит ключ, идентифицирующий этот элемент. Ключ обычно бывает или целым числом, или строкой.

Основными операциями, осуществляемыми с однонаправленными спискам, являются:

• создание списка;

• печать (просмотр) списка;

• вставка элемента в список;

• удаление элемента из списка;

• поиск элемента в списке

• проверка пустоты списка;

• удаление списка.

Особое внимание следует обратить на то, что при выполнении любых операций с линейным однонаправленным списком необходимо обеспечивать позиционирование какого-либо указателя на первый элемент. В противном случае часть или весь список будет недоступен.

Рассмотрим подробнее каждую из приведенных операций.

Число элементов связанного списка может расти или уменьшаться в зависимости от того, сколько данных мы хотим сохранять в нем. Чтобы добавить новый элемент в список, необходимо:

1. Получить память для него;

2. Поместить туда информацию;

3. Добавить элемент в конец списка (или начало).

Элемент списка состоит из разнотипных частей (сохранена информация и указатель), и его естественно представить записи. Перед описанием самой записи описывают указатель на нее:

Type {описание списка из целых чисел}

PList = ^ TList;

TList = record

Inf: Integer;

Next: PList;

end;

примеры

**Создание списка.**

Задание. Сформировать список, содержащий целые числа 3, 5, 1, 9.

Определим запись типа TList с полями, содержащими характеристики данных - значение очередного элемента и адреса следующего за ним элемента

PList = ^ TList;

TList = record

Data: Integer;

Next: PList;

end;

Чтобы список существовал, надо определить указатель на его начало. Опишем переменные.

Var

Head, x: PList;

Создадим первый элемент: New (Head) {Выделяем место в памяти для переменной Head} Head ^ .Next = nil; {Указатель на следующий элемент пустой (такого элемента нет)} Head ^ .Data = 3; {Заполняем информационное поле первого элемента}



Продолжим формирование списка, для этого нужно добавить элемент в конец списка.

Введем вспомогательную переменную указательного типа, будет хранить адрес последнего элемента списка: x = Head; {Сейчас последний элемент списка совпадает с его началом}

****

New (x ^ .Next) {Выделим области памяти для следующего (2-го) элементы и поместим его адрес в адресную часть предыдущие (1-го) элемента}



x = x ^ .Next; {Переменная x принимает значение адреса выделенной области. Таким образом осуществляется переход к следующему (второму) элемента списка}



x ^ .Data = 5; {Значение этого элемента} x ^ .Next = nil; {Следующего значения не}



Другие числа заносятся аналогично: New (x ^ .Next) {Выделим области памяти для следующего элемента} x = x ^ .Next; {Переход к следующему (3-м) элемента списка} x ^ .Data: = 1; {Значение этого элемента} x ^ .Next = nil; {Следующего значения не} New (x ^ .Next) {Выделим области памяти для следующего элемента} x = x ^ .Next; {Переход к следующему (4-й) элемента списка} x ^ .Data = 9; {Значение этого элемента} x ^ .Next = nil; {Следующего значения не}

Замечания. Как видно из примера, отличным является только создание первого (Head) элемента - главы списка. Все остальные действия полностью аналогичны и их естественно выполнять в цикле.

Присоединение нового элемента к голове списка производится аналогично ... ............... ...... New (x) {Ввод значения элемента x ^ .Data = ...} x ^ .Next = Head; Head = x; ... ............... ......

В этом случае последний введенный элемент окажется в списке первым, а первый - последним.

**Просмотр списка.**

Просмотр элементов списка осуществляется последовательно, начиная с его начала. Указатель List последовательно ссылается на первый, второй и т.д.

элементы списка до тех пор, пока весь список не будет пройден. При этом с каждым элементом списка выполняется некоторая операция- например, печать элемента. Начальное значение List - адрес первого элемента списка (Head). Digit - значение элемента удаляется.

List = Head;

While List ^ .Next <> nil do

begin

WriteLn (List ^ .Data)

List = List ^ .Next; {Переход к следующему элементу; аналог для массива i = i + 1}

end;

**Удаление элемента из списка.**

При удалении элемента из списка необходимо различать три случая: 1. Удаление элемента из начала списка.

2. Удаление элемента из середины списка.

3. Удаление с конца списка.

Удаление элемента из начала списка.

List = Head; {Запомним адрес первого элемента списка}

Head = Head ^ .List; {Теперь Head указывает на второй элемент списка}

Dispose (List) {Освободим память, занятую переменной List ^}

Удаление элемента из середины списка.

Для этого нужно знать адреса элемента удаляемого и элемента, находящегося в списке перед ним.

List = Head;

While (List <> nil) and (List ^ .Data <> Digit) do

begin

x = List;

List = List ^ .Next;

end;

x ^ .Next = List ^ .Next;

Dispose (List)

Удаление с конца списка.

Оно производится, когда указатель х показывает на предпоследний элемент списка, а List - на последний.

List = Head; x = Head;

While List ^ .Next <> nil do

begin

x = List;

List = List ^ .Next;

end;

x ^ .Next = nil;

Dispose (List)

Рис. 2. Вставка нового узла в односвязный список

****

Рис. 3. Удаление узла из односвязного списка

