Основы алгоритмизации и программирования

Лекция 15

Динамические структуры данных

Линейные списки

Некоторые задачи исключают использование структур данных фиксированного размера и требуют введения структур, способных увеличивать или уменьшать свой размер уже в процессе работы программы. Основу таких структур составляют **динамические переменные**.

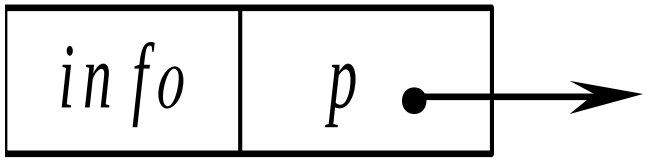
Динамическая переменная хранится в некоторой области **ОП**, обращение к которой производится через переменную-указатель

Как правило, динамические переменные организуются в списковые структуры данных, элементы которых имеют тип ***struct***. Для адресации элементов в структуру включается указатель (адресное поле) на область размещения следующего элемента

Такой список называют ***однонаправленным*** *(односвязным).* Если добавить в каждый элемент ссылку на предыдущий, получится ***двунаправленный*** *список (двусвязный),* если последний элемент связать указателем с первым, получится ***кольцевой*** *список.*

Линейные списки

Пусть необходимо создать линейный список, содержащий целые числа, тогда каждый элемент списка должен иметь информационную (***info***) часть, в которой будут находиться данные, и адресную часть (***p***), в которой будут размещаться указатели связей, т.е. элемент такого списка имеет вид



шаблон структуры будет иметь вид

```
struct Spis {
        int info;
        Spis *p;
} ;
```

Каждый элемент списка содержит ***ключ,*** идентифицирующий этот элемент. Ключ обычно бывает либо целым числом, либо строкой и является частью поля данных. В качестве ключа в процессе работы со списком могут выступать разные части поля данных.

Линейные списки

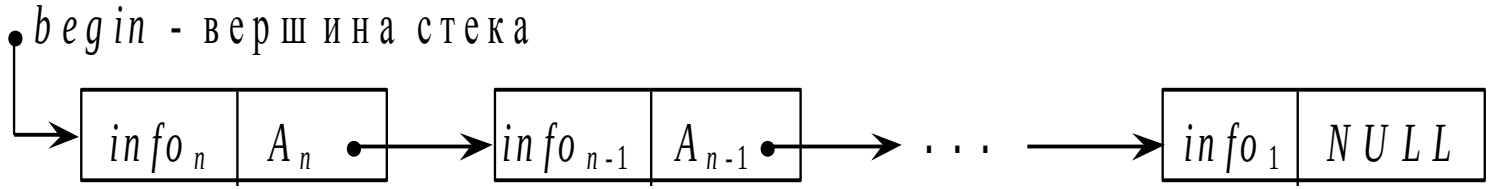
Например, если создается линейный список из записей, содержащих фамилию, год рождения, стаж работы и пол, любая часть записи может выступать в качестве ключа. При упорядочивании такого списка по алфавиту ключом будет являться фамилия. Как правило, ключи должны быть уникальными, но могут и совпадать.

Над списками можно выполнять следующие ***операции***

- начальное формирование списка (создание первого элемента)
- вставка элемента в заданное место списка (до или после элемента с заданным ключом)
- добавление элемента в список
- обработка (чтение, удаление и т.п.) элемента с заданным ключом
- упорядочивание списка по ключу

Структура данных СТЕК

Стек – упорядоченный набор данных, в котором размещение новых элементов и удаление существующих производится только с одного его конца, который называют вершиной стека, т.е. **стек** – это список с одной точкой доступа к его элементам.



Стек – структура типа ***LIFO*** (*Last In, First Out*) – последним вошел, первым выйдет. Стек получил свое название из-за схожести с оружейным магазином с патронами (обойма): когда в стек добавляется новый элемент, то прежний проталкивается вниз и временно становится недоступным. Когда же верхний элемент удаляется из стека, следующий за ним поднимается вверх и становится опять доступным.

Структура данных СТЕК

Максимальное число элементов стека ограничивается, т.е. по мере вталкивания в стек новых элементов память под него должна динамически запрашиваться и освобождаться также динамически при удалении элемента из стека. Таким образом, **стек** – динамическая структура данных, состоящая из переменного числа элементов одинакового типа.

Состояние стека рассматривается только по отношению к его вершине, а не по отношению к количеству его элементов, т.е. только вершина стека характеризует его состояние

Операции, выполняемые над стеком, имеют специальные названия:
push – добавление элемента в стек (вталкивание);
pop – выталкивание (извлечение) элемента из стека, верхний элемент стека удаляется (не может применяться к пустому стеку).

Кроме этих обязательных операций часто нужно прочитать значение элемента в вершине стека, не извлекая его оттуда. Такая операция получила название ***peek***.

Структура данных ОЧЕРЕДЬ

Очередь – упорядоченный набор данных (структура данных), в котором в отличие от стека извлечение данных происходит из начала цепочки, а добавление данных – в конец этой цепочки.

Очередь также называют структурой данных, организованной по принципу ***FIFO*** (*First In, First Out*) – первый вошел (первый созданный элемент очереди), первый вышел.

В языке **Си** работа с очередью, как и со стеком, реализуется при помощи структур, указателей на структуры и операций динамического выделения и освобождения памяти.

Пример очереди – некоторый механизм обслуживания, который может выполнять заказы только последовательно один за другим. Если при поступлении нового заказа данное устройство свободно, оно немедленно приступит к выполнению этого заказа, если же оно выполняет какой-то ранее полученный заказ, то новый заказ поступает в конец очереди из других ранее пришедших заказов. Когда устройство освобождается, оно приступает к выполнению заказа из начала очереди, т.е. этот заказ удаляется из очереди и первым в ней становится следующий за ним заказ.

Структура данных ОЧЕРЕДЬ

При работе с очередью обычно помимо текущего указателя используют еще два указателя, первый указатель устанавливается на начало очереди, а второй – на ее конец.

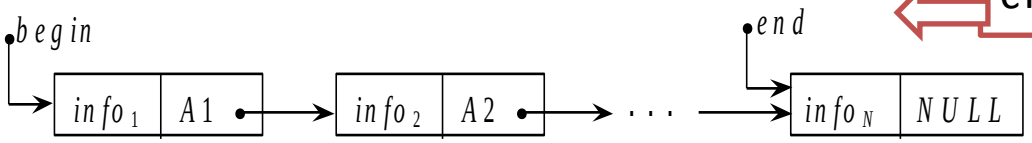
Шаблон элемента структуры, информационной частью которого является целое число, может иметь следующий вид:

```
struct Spis {
        int info;
        Spis *Next;
    };
```

При организации очереди обычно используют два указателя

Spis *begin, *end;

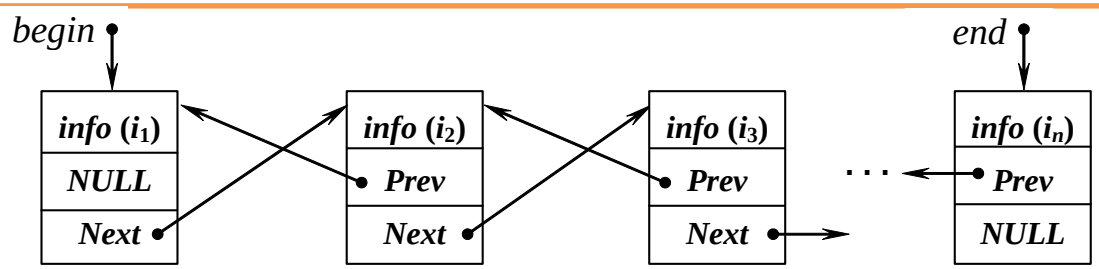
где ***begin*** и ***end*** – указатели на начало и конец очереди соответственно, т.е. при создании очереди мы организуем структуру данных следующего вида:



Каждый элемент имеет информационную *info* и адресную *Next* (*A*1, *A*2, ...) части.
Основные операции с очередью следующие:
– формирование очереди;
– добавление нового элемента в конец очереди;
– удаление элемента из начала очереди.

Двунаправленный линейный список

Более универсальным является **двунаправленный (двухсвязный)** список, в каждый элемент которого кроме указателя на следующий элемент включен и указатель на предыдущий. Для обработки такого списка обычно аналогично очереди используются два указателя – на первый и последний элементы.



Введем структуру, в которой (для простоты, как и раньше) информационной частью *info* будут целые числа, а адресная часть состоит из двух указателей на предыдущий (*Prev*) и следующий (*Next*) элементы:

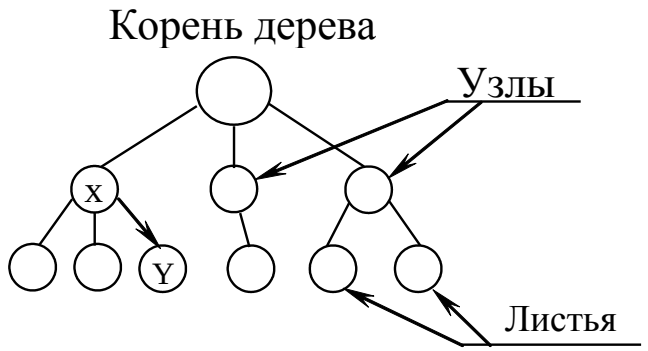
```
struct Spis   {
                    int info;
                    Spis  *Prev, *Next;
          } ;
```

Для работы со списком декларируем Spis **begin*, **end*; – указатели на начало и конец списка соответственно.

Нелинейные структуры данных

Введение в динамическую переменную двух и более полей-указателей позволяет получить нелинейные структуры данных. Наиболее распространенными являются структуры с иерархическим представлением, которые хорошо изображаются следующим образом

Дерево состоит из элементов, называемых ***узлами*** (вершинами). Узлы соединены между собой направленными дугами. В случае $X \rightarrow Y$ вершина X называется *родителем*, а Y – *сыном* (*дочерью*).

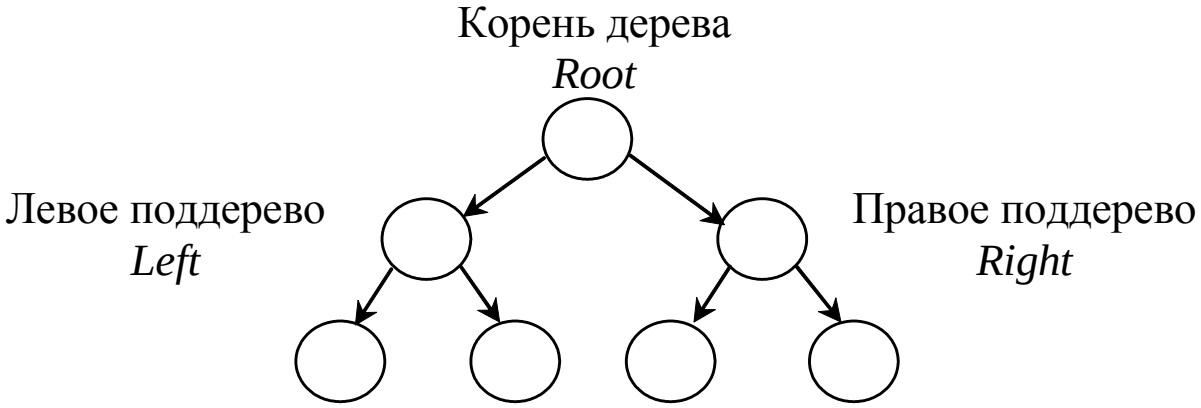


Дерево имеет единственный узел, не имеющий родителей (ссылок на этот узел), который называется ***корнем***. Любой другой узел имеет ровно одного родителя, т.е. на каждый узел дерева имеется ровно одна ссылка.
Узел, не имеющий сыновей, называется ***листом***.

Внутренний узел – это узел, не являющийся ни листом, ни корнем. ***Порядок узла*** равен количеству его узлов-сыновей. ***Степень дерева*** – максимальный порядок его узлов. ***Высота (глубина) узла*** равна числу его родителей плюс один. ***Высота дерева*** – это наибольшая высота его узлов.

Бинарные деревья

Бинарное дерево – это динамическая структура данных, в которой каждый узел-родитель содержит, кроме данных, не более двух сыновей (левый и правый).



Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева – больше, оно называется деревом поиска. Одинаковые ключи здесь не допускаются.

Представление динамических данных в виде древовидных структур оказывается довольно удобным и эффективным для решения задач быстрого поиска информации.

Дерево по своей организации является рекурсивной структурой данных, поскольку каждое его поддерево также является деревом. В связи с этим действия с такими структурами чаще всего описываются с помощью рекурсивных алгоритмов.