
Работа со списками

А. Г. Фенстер, <http://info.fenster.name>

6 февраля 2009 г.

Структура списка

Связанный список — один из способов хранения информации, при котором в программе хранится указатель на первый элемент списка, первый элемент хранит в себе указатель на следующий элемент, и так далее. У последнего элемента списка указатель на следующий элемент хранит нулевое значение. Переменную, хранящую указатель на первый элемент списка, мы назовём `head` (голова списка).

На рис. 1 схематично показан связанный список из трёх элементов. Перечёркнутым прямоугольником обозначен нулевой указатель. Тип, соответствующий элементу списка, можно описать следующим образом:

```
struct item {  
    int data;  
    struct item *next;  
};  
struct item *head;
```

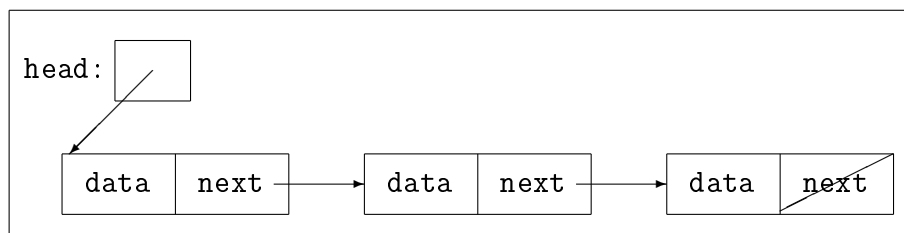


Рис. 1. Связанный список

Вставка, удаление и перебор элементов

Рассмотрим процедуру добавления элемента в начало списка. Для создания нового элемента необходимо воспользоваться стандартными средствами выделения динамической памяти: функцией `malloc`. Алгоритм добавления элемента в начало списка будет выглядеть следующим образом:

1. Выделить память под новый элемент списка. Пусть переменная `p` указывает на выделенную память.
2. Установить необходимое значение поля `data`.
3. Присвоить полю `next` значение элемента `head`.
4. Присвоить переменной `head` значение `p`.

Ниже приведён код, реализующий этот алгоритм:

```
struct item *p;
p = malloc(sizeof(struct item));
if (!p)
{
    /* ошибка: невозможно выделить память */
}
p->data = ...;
p->next = head;
head = p;
```

Чтобы удалить первый элемент списка, необходимо сначала запомнить следующий за ним элемент, затем освободить выделенную под первый элемент память при помощи стандартной функции `free` и присвоить переменной `head` адрес второго элемента списка (который теперь станет первым):

1. В переменную `p` сохранить значение поля `next` элемента, на который указывает `head`.
2. Освободить память, на которую указывает `head`.

3. Присвоить переменной `head` значение `p`.

То же самое на языке C:

```
struct item *p;
p = head->next;
free(head);
head = p;
```

Для доступа к элементу списка необходимо перебирать их подряд, начиная от `head`, пока не будет найден нужный элемент. Это удобно делать при помощи такого цикла `for`:

```
struct item *p;
for (p = head; p; p = p->next)
{
    /* обработать элемент, на который указывает p */
}
```

Имея указатель на некоторый элемент, можно вставить элемент после него или удалить следующий за ним элемент, выполнив последовательность действий, похожую на алгоритмы вставки в начало и удаления первого элемента.

Чтобы добавить элемент в конец списка, необходимо сначала перебрать все его элементы в поисках последнего. Для ускорения этой операции помимо `head` часто хранят и указатель на последний элемент списка (`tail`).

Временные оценки

В таблице для различных задач с массивами и списками приведено время работы соответствующих алгоритмов. Через $O(1)$ обозначено константное время работы — алгоритм, не зависящий от N (количества элементов в массиве или списке). При оценке времени работы алгоритмов для списка считаем, что хранятся указатели на голову и на хвост списка.

Работа со списками

| | Массив | Связанный список |
|---------------------------------------|--------|------------------|
| Вставка в начало | $O(N)$ | $O(1)$ |
| Вставка в конец | $O(1)$ | $O(1)$ |
| Вставка нового элемента после данного | $O(N)$ | $O(1)$ |
| Поиск элемента с данным значением | $O(N)$ | $O(N)$ |
| Поиск элемента по номеру | $O(1)$ | $O(N)$ |
| Удаление элемента | $O(N)$ | $O(1)$ |