В односвязном списке каждый элемент информации содержит ссылку на следующий элемент списка. Каждый элемент данных обычно представляет собой структуру, которая состоит из информационных полей и указателя связи. Концептуально односвязный список выглядит так, как показано на рис. 22.2.

|  |
| --- |
| +---------+ +---------+ +---------+  | данные | | данные | | данные |  +---------+ +---------+ +---------+  |указатель|--->|указатель|--->| 0 |  +---------+ +---------+ +---------+ |
| *Рис. 22.2 Односвязный список* |

Существует два основных способа построения односвязного списка. Первый способ — помещать новые элементы в конец списка[[1]](http://www.chitay.org/c/22/2205.htm" \l "11). Второй — вставлять элементы в определенные позиции списка, например, в порядке возрастания. От способа построения списка зависит алгоритм функции добавления элемента. Давайте начнем с более простого способа создания списка путем помещения элементов в конец.

Как правило, элементы связанного списка являются структурами, так как, помимо данных, они содержат ссылку на следующий элемент. Поэтому необходимо определить структуру, которая будет использоваться в последующих примерах. Поскольку списки рассылки обычно хранятся в связанных списках, хорошим выбором будет структура, описывающая почтовый адрес. Ее описание показано ниже:

struct address {

char name[40];

char street[40];

char city[20];

char state[3];

char zip[11];

struct address \*next; /\* ссылка на следующий адрес \*/

} info;

Приведенная ниже функция slstore() создает односвязный список путем помещения каждого очередного элемента в конец списка. В качестве параметров ей передаются указатель на структуру типа address, содержащую новую запись, и указатель на последний элемент списка. Если список пуст, указатель на последний элемент должен быть равен нулю.

void slstore(struct address \*i,

struct address \*\*last)

{

if(!\*last) \*last = i; /\* первый элемент в списке \*/

else (\*last)->next = i;

i->next = NULL;

\*last = i;

}

Несмотря на то, что созданный с помощью функции slstore() список можно отсортировать отдельной операцией уже после его создания, легче сразу создавать упорядоченный список, вставляя каждый новый элемент в нужное место в последовательности. Кроме того, если список уже отсортирован, имеет смысл поддерживать его упорядоченность, вставляя новые элементы в соответствующие позиции. Для вставки элемента таким способом требуется последовательно просматривать список до тех пор, пока не будет найдено место нового элемента, затем вставить в найденную позицию новую запись и переустановить ссылки.

При вставке элемента в односвязный список может возникнуть одна из трех ситуаций: элемент становится первым, элемент вставляется между двумя другими, элемент становится последним. На рис. 22.3 показана схема изменения ссылок в каждом случае.

|  |
| --- |
| **Вставка в начало списка**  +----+ п +----+  |new | р |new |  +----+ е +----+  | | в .------------| |  +----+ р | +----+  а |  +----+ +----+ +----+ щ в | +----+ +----+ +----+  |info| |info| |info| а | |info| |info| |info|  \/\/\->+----+ +----+ +----+ е | +----+ +----+ +----+  | |--->| |--->| 0 | т '->| |--->| |--->| 0 |  +----+ +----+ +----+ с +----+ +----+ +----+  я  **Вставка в середину списка**  +----+ п +----+  |new | р |new |  +----+ е +----+  | | в .---------->| |  +----+ р | .--+----+  а | |  +----+ +----+ +----+ щ в | +----+ | +----+ +----+  |info| |info| |info| а | |info| | |info| .->|info|  \/\/\->+----+ +----+ +----+ е \/\/\->+----+ | +----+ | +----+  | |--->| |--->| 0 | т '-| | '->| |-' | 0 |  +----+ +----+ +----+ с +----+ +----+ +----+  я  **Вставка в конец списка**  +----+ п +----+  |new | р |new |<----------.  +----+ е +----+ |  | | в | 0 | |  +----+ р +----+ |  а |  +----+ +----+ +----+ щ в +----+ +----+ +----+ |  |info| |info| |info| а |info| .->|info| |info| |  \/\/\->+----+ +----+ +----+ е \/\/\->+----+ | +----+ +----+ |  | |--->| |--->| 0 | т | |-' | |--->| |-'  +----+ +----+ +----+ с +----+ +----+ +----+  я |
| *Рис. 22.3. Вставка элемента new в односвязный список (в котором info - поле данных)* |

Следует помнить, что при вставке элемента в начало (первую позицию) списка необходимо также изменить адрес входа в список где-то в другом месте программы. Чтобы избежать этих сложностей, можно в качестве первого элемента списка хранить служебный *сторожевой* элемент[[2]](http://www.chitay.org/c/22/2205.htm" \l "22). В случае упорядоченного списка необходимо выбрать некоторое специальное значение, которое всегда будет первым в списке, чтобы начальный элемент оставался неизменным. Недостатком данного метода является довольно большой расход памяти на хранение сторожевого элемента, но обычно это не столь важно.

Следующая функция, sls\_store(), вставляет структуры типа address в список рассылки, упорядочивая его по возрастанию значений в поле name. Функция принимает указатели на указатели на первый и последний элементы списка, а также указатель на вставляемый элемент. Поскольку первый или последний элементы списка могут измениться, функция sls\_store() при необходимости автоматически обновляет указатели на начало и конец списка. При первом вызове данной функции указатели first и last должны быть равны нулю.

/\* Вставка в упорядоченный односвязный список. \*/

void sls\_store(struct address \*i, /\* новый элемент \*/

struct address \*\*start, /\* начало списка \*/

struct address \*\*last) /\* конец списка \*/

{

struct address \*old, \*p;

p = \*start;

if(!\*last) { /\* первый элемент в списке \*/

i->next = NULL;

\*last = i;

\*start = i;

return;

}

old = NULL;

while(p) {

if(strcmp(p->name, i->name)<0) {

old = p;

p = p->next;

}

else {

if(old) { /\* вставка в середину \*/

old->next = i;

i->next = p;

return;

}

i->next = p; /\* вставка в начало \*/

\*start = i;

return;

}

}

(\*last)->next = i; /\* вставка в конец \*/

i->next = NULL;

\*last = i;

}

Последовательный перебор элементов связанного списка осуществляется очень просто: начать с начала и следовать указателям. Обычно фрагмент кода перебора настолько мал, что его вставляют в другую процедуру — например, функцию поиска, удаления или отображения содержимого. Так, приведенная ниже функция выводит на экран все имена из списка рассылки:

void display(struct address \*start)

{

while(start) {

printf("%s\n", start->name);

start = start->next;

}

}

При вызове функции display() параметр start должен быть указателем на первую структуру в списке. После этого функция переходит к следующему элементу, на который указывает указатель в поле next. Процесс прекращается, когда next равно нулю.

Для получения элемента из списка нужно просто пройти по цепочке ссылок. Ниже приведен пример функции поиска по содержимому поля name:

struct address \*search(struct address \*start, char \*n)

{

while(start) {

if(!strcmp(n, start->name)) return start;

start = start->next;

}

return NULL; /\* подходящий элемент не найден \*/

}

Поскольку функция search() возвращает указатель на элемент списка, содержащий искомое имя, возвращаемый тип описан как указатель на структуру address. При отсутствии в списке подходящих элементов возвращается нуль (NULL).

Удаление элемента из односвязного списка выполняется просто. Так же, как и при вставке, возможны три случая: удаление первого элемента, удаление элемента в середине, удаление последнего элемента. На рис. 22.4 показаны все три операции.

|  |
| --- |
| **Удаление первого элемента списка**    +------+ +------+ +------+  |данные| |данные| |данные|  \/\/\->+------+ +------+ +------+  | |--->| |--->| 0 |  +------+ +------+ +------+  превращается в  +------+ +------+ +------+  |удален| \/\/\->|данные| .->|данные|  +------+ +------+ | +------+  | 0 | | |-' | 0 |  +------+ +------+ +------+  **Удаление среднего элемента списка**    +------+ +------+ +------+  |данные| |данные| |данные|  \/\/\->+------+ +------+ +------+  | |--->| |--->| 0 |  +------+ +------+ +------+  превращается в  +------+ +------+ +------+  |данные| |удален| |данные|  \/\/\->+------+ +------+ +------+  | | | 0 | .->| 0 |  +------+ +------+ | +------+  \\_\_\_\_\_\_\_\_\_\_\_\_\_\_|  **Удаление последнего элемента списка**    +------+ +------+ +------+  |данные| |данные| |данные|  \/\/\->+------+ +------+ +------+  | |--->| |--->| 0 |  +------+ +------+ +------+  превращается в  +------+ +------+ +------+  |данные| |данные| |удален|  \/\/\->+------+ +------+ +------+  | |--->| 0 | | 0 |  +------+ +------+ +------+ |
| *Рис. 22.4. Удаление элемента из односвязного списка* |

Ниже приведена функция, удаляющая заданный элемент из списка структур address.

void sldelete(

struct address \*p, /\* предыдущий элемент \*/

struct address \*i, /\* удаляемый элемент \*/

struct address \*\*start, /\* начало списка \*/

struct address \*\*last) /\* конец списка \*/

{

if(p) p->next = i->next;

else \*start = i->next;

if(i==\*last && p) \*last = p;

}

Функции sldelete() необходимо передавать указатели на удаляемый элемент, предшествующий удаляемому, а также на первый и последний элементы. При удалении первого элемента указатель на предшествующий элемент должен быть равен нулю (NULL). Данная функция автоматически обновляет указатели start и last, если один из них ссылается на удаляемый элемент.

У односвязных списков есть один большой недостаток: односвязный список невозможно прочитать в обратном направлении. По этой причине обычно применяются двусвязные списки.