Двусвязный список состоит из элементов данных, каждый из которых содержит ссылки как на следующий, так и на предыдущий элементы. На рис. 22.5 показана организация ссылок в двусвязном списке.

|  |
| --- |
| +-------+ +-------+ +-------+  |данные | .->|данные | .->|данные |  +---+---+ | +---+---+ | +---+---+  | 0 | |-' | | |-' | | 0 |  | | |<---| | |<---| | |  +---+---+ +---+---+ +---+---+ |
| *Рис. 22.5. Двусвязные списки* |

Наличие двух ссылок вместо одной предоставляет несколько преимуществ. Вероятно, наиболее важное из них состоит в том, что перемещение по списку возможно в обоих направлениях. Это упрощает работу со списком, в частности, вставку и удаление. Помимо этого, пользователь может просматривать список в любом направлении. Еще одно преимущество имеет значение только при некоторых сбоях. Поскольку весь список можно пройти не только по прямым, но и по обратным ссылкам, то в случае, если какая-то из ссылок станет неверной, целостность списка можно восстановить по другой ссылке.

При вставке нового элемента в двусвязный список могут быть три случая: элемент вставляется в начало, в середину и в конец списка. Эти операции показаны на рис. 22.6.

|  |
| --- |
| **Вставка элемента в начало списка**  +-----+ +-----+  | new | \/\/\->| new |  +--+--+ п +--+--+  | | | р .----------->|0 | |  | | | е | | | |  +--+--+ в | +--+|-+  р | \_\_\_\_\_|  а | |  +-----+ +-----+ +-----+ щ в | +-----+ | +-----+ +-----+  |info | |info | |info | а \/\/\->|info |<-' |info | |info |  \/\/\->+--+--+ +--+--+ +--+--+ т | +--+--+ +--+--+ +--+--+  |0 | |--->| | |--->| |0 | с | | | |--->| | |--->| |0 |  | | |<---| | |<---| | | я '-| | |<---| | |<---| | |  +--+--+ +--+--+ +--+--+ +--+--+ +--+--+ +--+--+  **Вставка элемента в середину списка**  +-----+ +-----+  | new | | new |  +--+--+ п +--+--+  | | | р .---------| | |  | | | е | .--->| | |  +--+--+ в | | +--+A|+  р | | \_\_\_\_\_||  а | | | |  +-----+ +-----+ +-----+ щ в +--V--+ | | +--+-V+ +-----+  |info | |info | |info | а \/\/\->|info | | | |info | |info |  \/\/\->+--+--+ +--+--+ +--+--+ т +--+--+ | | +--+--+ +--+--+  |0 | |--->| | |--->| |0 | с |0 | |-' '-| | |--->| |0 |  | | |<---| | |<---| | | я | | | | | |<---| | |  +--+--+ +--+--+ +--+--+ +--+--+ +--+--+ +--+--+  **Вставка элемента в конец списка**  +-----+ +-----+  | new | | new |  +--+--+ п +--+--+  | | | р | |0 |  | | | е | | |<-----------.  +--+--+ в +|-+--+ |  р |\_\_\_\_\_\_\_\_\_\_\_\_ |  а | |  +-----+ +-----+ +-----+ щ в +-----+ +-----+ +--V--+ |  |info | |info | |info | а \/\/\->|info | |info | |info | |  \/\/\->+--+--+ +--+--+ +--+--+ т +--+--+ +--+--+ +--+--+ |  |0 | |--->| | |--->| |0 | с |0 | |--->| | |--->| | |-'  | | |<---| | |<---| | | я | | |<---| | |<---| | |  +--+--+ +--+--+ +--+--+ +--+--+ +--+--+ +--+--+ |
| *Рис. 22.6. Операции с двусвязными списками (Здесь new - вставляемый элемент, а info - поле данных)* |

Построение двусвязного списка выполняется аналогично построению односвязного за исключением того, что необходимо установить две ссылки. Поэтому в структуре данных должны быть описаны два указателя связи. Возвращаясь к примеру списка рассылки, для двусвязного списка структуру address можно модифицировать следующим образом:

struct address {

char name[40];

char street[40] ;

char city[20];

char state[3];

char zip[11];

struct address \*next;

struct address \*prior;

} info;

Следующая функция, dlstore(), создает двусвязный список, используя структуру address в качестве базового типа данных:

void dlstore(struct address \*i, struct address \*\*last)

{

if(!\*last) \*last = i; /\* вставка первого элемента \*/

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

i->next = NULL;

i->prior = \*last;

\*last = i;

}

Функция dlstore() помещает новые записи в конец списка. В качестве параметров ей необходимо передавать указатель на сохраняемые данные; а также указатель на конец списка, который при первом вызове должен быть равен нулю (NULL).

Подобно односвязным, двусвязные списки можно создавать с помощью функции, которая будет помещать элементы в определенные позиции, а не только в конец списка. Показанная ниже функция dls\_store() создает список, упорядочивая его по возрастанию имен:

/\* Создание упорядоченного двусвязного списка. \*/

void dls\_store(

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

struct address \*\*start, /\* первый элемент в списке \*/

struct address \*\*last /\* последний элемент в списке \*/

)

{

struct address \*old, \*p;

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

i->next = NULL;

i->prior = NULL;

\*last = i;

\*start = i;

return;

}

p = \*start; /\* начать с начала списка \*/

old = NULL;

while(p) {

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

old = p;

p = p->next;

}

else {

if(p->prior) {

p->prior->next = i;

i->next = p;

i->prior = p->prior;

p->prior = i;

return;

}

i->next = p; /\* новый первый элемент \*/

i->prior = NULL;

p->prior = i;

\*start = i;

return;

}

}

old->next = i; /\* вставка в конец \*/

i->next = NULL;

i->prior = old;

\*last = i;

}

Поскольку первый и последний элементы списка могут меняться, функция dls\_store() автоматически обновляет указатели на начало и конец списка посредством параметров start и last. При вызове функции необходимо передавать указатель на сохраняемые данные и указатели на указатели на первый и последний элементы списка. В первый раз параметры start и last должны быть равны нулю (NULL).

Как и в односвязных списках, для получения элемента данных двусвязного списка необходимо переходить по ссылкам до тех пор, пока не будет найден искомый элемент.

При удалении элемента двусвязного списка могут возникнуть три случая: удаление первого элемента, удаление элемента из середины и удаление последнего элемента. На рис. 22.7 показано, как при этом изменяются ссылки. Показанная ниже функция dldelete() удаляет элемент двусвязного списка:

void dldelete(

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

struct address \*\*start, /\* первый элемент \*/

struct address \*\*last) /\* последний элемент \*/

{

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

else { /\* new first item \*/

\*start = i->next;

if(start) start->prior = NULL;

}

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

else /\* удаление последнего элемента \*/

\*last = i->prior;

}

Поскольку первый или последний элементы списка могут быть удалены, функция dldelete() автоматически обновляет указатели на начало и конец списка посредством параметров start и last. При вызове ей передаются указатель на удаляемый элемент и указатели на указатели на начало и конец списка.

|  |
| --- |
| **Удаление первого элемента списка**    +-------+ +-------+ +-------+  \/\/\->|данные | |данные | |данные |  +---+---+ +---+---+ +---+---+  | 0 | |--->| | |--->| | 0 |  +---+-A-+ +-|-+-A-+ +-|-+---+  |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_|    превращается в  +-------+ +-------+ +-------+  |удален | \/\/\->|данные | |данные |  +---+---+ +---+---+ +---+---+  | 0 | 0 | | 0 | |--->| | 0 |  +---+---+ +---+-A-+ +-|-+---+  |\_\_\_\_\_\_\_\_|  **Удаление элемента из середины списка**    +-------+ +-------+ +-------+  \/\/\->|данные | |данные | |данные |  +---+---+ +---+---+ +---+---+  | 0 | |--->| | |--->| | 0 |  +---+-A-+ +-|-+-A-+ +-|-+---+  |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_|    превращается в  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  | |  +-------+ | +-------+ +---V---+  \/\/\->|данные | | |удален | |данные |  +---+---+ | +---+---+ +---+---+  | 0 | |-' | 0 | 0 |--->| | 0 |  +---+-A-+ +---+---+ +-|-+---+  |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|  **Удаление первого элемента списка**    +-------+ +-------+ +-------+  \/\/\->|данные | |данные | |данные |  +---+---+ +---+---+ +---+---+  | 0 | |--->| | |--->| | 0 |  +---+-A-+ +-|-+-A-+ +-|-+---+  |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_|  превращается в  +-------+ +-------+ +-------+  \/\/\->|данные | |данные | |удален |  +---+---+ +---+---+ +---+---+  | 0 | |--->| | 0 |--->| 0 | 0 |  +---+-A-+ +-|-+---+ +---+---+  |\_\_\_\_\_\_\_\_| |
| *Рис. 22.7. Удаление элемента двусвязного списка* |