Лабораторная работа № 1

Тема: *Реализация линейной структуры данных «Список» и основные алгоритмы обработки.*

Цель: *ознакомиться с основами линейной структуры данных «Список», изучить основные алгоритмы обработки ЛСД «Список», научиться применять полученные знания на практике.*

1 Краткая теория

1.1 СВЯЗНЫЕ СПИСКИ

Связный список является простейшим типом данных динамической структуры, состоящей из элементов (***узлов***). Каждый узел включает в себя в классическом варианте два поля:

- данные (в качестве данных может выступать переменная, объект класса или структуры и т. д.)
- указатель на следующий узел в списке.

Элементы связанного списка можно помещать и исключать произвольным образом.

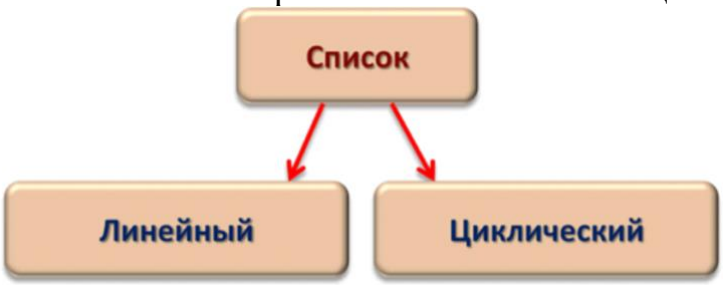
Доступ к списку осуществляется через указатель, который содержит адрес первого элемента списка, называемый ***корнем списка***.

Классификация списков

По количеству полей указателей различают однонаправленный (односвязный) и двунаправленный (двусвязный) списки.

Связный список, содержащий только один указатель на следующий элемент, называется ***односвязным***. Связный список, содержащий два поля указателя – на следующий элемент и на предыдущий, называется ***двусвязным***.

По способу связи элементов различают линейные и циклические списки.



Связный список, в котором, последний элемент указывает на NULL, называется ***линейным***. Связный список, в котором последний элемент связан с первым, называется ***циклическим***.

Виды списков

Таким образом, различают 4 основных вида списков.

- ***Односвязный линейный список*** (ОЛС). Каждый узел ОЛС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

- ***Односвязный циклический список*** (ОЦС). Каждый узел ОЦС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит адрес первого узла (корня списка).

- ***Двусвязный линейный список*** (ДЛС). Каждый узел ДЛС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит нулевое значение (указывает на NULL). Поле указателя на предыдущий узел первого узла (корня списка) также содержит нулевое значение (указывает на NULL).

- ***Двусвязный циклический список*** (ДЦС). Каждый узел ДЦС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит адрес первого узла (корня списка). Поле указателя на предыдущий узел первого узла (корня списка) содержит адрес последнего узла.

Сравнение массивов и связных списков

Массив	Список
Выделение памяти осуществляется единовременно под весь массив до начала его использования	Выделение памяти осуществляется по мере ввода новых элементов
При удалении/добавлении элемента требуется копирование всех последующих элементов для осуществления их сдвига	Удаление/добавление элемента осуществляется переустановкой указателей, при этом сами данные не копируются
Для хранения элемента требуется объем памяти, необходимый только для хранения данных этого элемента	Для хранения элемента требуется объем памяти, достаточный для хранения данных этого элемента и указателей (1 или 2) на другие элементы списка
Доступ к элементам может осуществляться в произвольном порядке	Возможен только последовательный доступ к элементам

1.2 ОДНОСВЯЗНЫЙ ЛИНЕЙНЫЙ СПИСОК

Каждый узел однонаправленного (односвязного) линейного списка (ОЛС) содержит одно поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

Узел ОЛС можно представить в виде структуры

```
struct list
{
  int field; // поле данных
  struct list *ptr; // указатель на следующий элемент
};
```

Основные действия, производимые над элементами ОЛС:

- Инициализация списка
- Добавление узла в список
- Удаление узла из списка
- Удаление корня списка
- Вывод элементов списка
- Взаимообмен двух узлов списка

Инициализация ОЛС

Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит нулевое значение.


```
1   struct list * init(int a) // a- значение первого узла
2   {
3     struct list *lst;
4     // выделение памяти под корень списка
5     lst = (struct list*)malloc(sizeof(struct list));
6     lst->field = a;
7     lst->ptr = NULL; // это последний узел списка
8     return(lst);
9   }
```

Добавление узла в ОЛС

Функция добавления узла в список принимает два аргумента:

- Указатель на узел, после которого происходит добавление
- Данные для добавляемого узла.

Процедуру добавления узла можно отобразить следующей схемой:

Добавление узла в ОЛС включает в себя следующие этапы:

- создание добавляемого узла и заполнение его поля данных;
- переустановка указателя узла, предшествующего добавляемому, на добавляемый узел;
- установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).

Таким образом, функция добавления узла в ОЛС имеет вид:

```
1   struct list * addelem(list *lst, int number)
2   {
3     struct list *temp, *p;
4     temp = (struct list*)malloc(sizeof(list));
5     p = lst->ptr; // сохранение указателя на следующий узел
6     lst->ptr = temp; // предыдущий узел указывает на создаваемый
7     temp->field = number; // сохранение поля данных добавляемого узла
8     temp->ptr = p; // созданный узел указывает на следующий элемент
9     return(temp);
10  }
```

Возвращаемым значением функции является адрес добавленного узла.

Удаление узла ОЛС

В качестве аргументов функции удаления элемента ОЛС передаются указатель на удаляемый узел, а также указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым.

Удаление узла может быть представлено следующей схемой:

Удаление узла ОЛС включает в себя следующие этапы:

- установка указателя предыдущего узла на узел, следующий за удаляемым;
- освобождение памяти удаляемого узла.

```
1   struct list * deletelem(list *lst, list *root)
2   {
3     struct list *temp;
4     temp = root;
5     while (temp->ptr != lst) // просматриваем список начиная с корня
6     { // пока не найдем узел, предшествующий lst
7       temp = temp->ptr;
8     }
9     temp->ptr = lst->ptr; // переставляем указатель
10    free(lst); // освобождаем память удаляемого узла
11    return(temp);
12  }
```

Удаление корня списка

Функция удаления корня списка в качестве аргумента получает указатель на текущий корень списка. Возвращаемым значением будет новый корень списка — тот узел, на который указывает удаляемый корень.

```
1   struct list * deletehead(list *root)
2   {
3     struct list *temp;
4     temp = root->ptr;
5     free(root); // освобождение памяти текущего корня
6     return(temp); // новый корень списка
7   }
```

Вывод элементов списка

В качестве аргумента в функцию вывода элементов передается указатель на корень списка. Функция осуществляет последовательный обход всех узлов с выводом их значений.

```
1   void listprint(list *lst)
2   {
3     struct list *p;
4     p = lst;
5     do {
6       printf("%d ", p->field); // вывод значения элемента p
7       p = p->ptr; // переход к следующему узлу
8     } while (p != NULL);
9   }
```

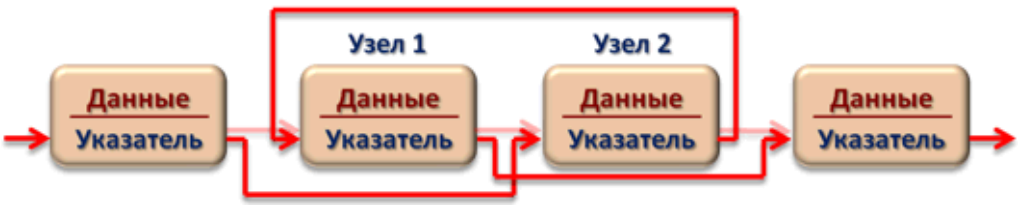
Взаимообмен узлов ОЛС

В качестве аргументов функция взаимообмена ОЛС принимает два указателя на обмениваемые узлы, а также указатель на корень списка. Функция возвращает адрес корневого элемента списка.

Взаимообмен узлов списка осуществляется путем переустановки указателей. Для этого необходимо определить предшествующий и последующий узлы для каждого заменяемого. При этом возможны две ситуации:

- заменяемые узлы являются соседями;
- заменяемые узлы не являются соседями, то есть между ними имеется хотя бы один элемент.

При замене соседних узлов переустановка указателей выглядит следующим образом:



При замене узлов, не являющихся соседними переустановка указателей выглядит следующим образом:

При переустановке указателей необходима также проверка, является ли какой-либо из заменяемых узлов корнем списка, поскольку в этом случае не существует узла, предшествующего корневому.

Функция взаимообмена узлов списка выглядит следующим образом:

```
struct list * swap(struct list *lst1, struct list *lst2, struct list *head)
{
  // Возвращает новый корень списка
  struct list *prev1, *prev2, *next1, *next2;
  prev1 = head;
  prev2 = head;
  if (prev1 == lst1)
    prev1 = NULL;
  else
    while (prev1->ptr != lst1) // поиск узла предшествующего lst1
      prev1 = prev1->ptr;
  if (prev2 == lst2)
    prev2 = NULL;
  else
    while (prev2->ptr != lst2) // поиск узла предшествующего lst2
      prev2 = prev2->ptr;
  next1 = lst1->ptr;  // узел следующий за lst1
  next2 = lst2->ptr;  // узел следующий за lst2
  if (lst2 == next1)
  {                       // обмениваются соседние узлы
    lst2->ptr = lst1;
    lst1->ptr = next2;
    if (lst1 != head)
      prev1->ptr = lst2;
  }
  else
    if (lst1 == next2)
    {
      // обмениваются соседние узлы
      lst1->ptr = lst2;
      lst2->ptr = next1;
      if (lst2 != head)
        prev2->ptr = lst2;
    }
    else
    {
      // обмениваются отстоящие узлы
      if (lst1 != head)
        prev1->ptr = lst2;
      lst2->ptr = next1;
      if (lst2 != head)
        prev2->ptr = lst1;
      lst1->ptr = next2;
    }
  if (lst1 == head)
    return(lst2);
  if (lst2 == head)
    return(lst1);
  return(head);
}
```

1.3 ДВУСВЯЗНЫЙ ЛИНЕЙНЫЙ СПИСОК

Каждый узел двунаправленного (двусвязного) линейного списка (ДЛС) содержит два поля указателей — на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель на следующий узел последнего узла также содержит нулевое значение.

Узел ДЛС можно представить в виде структуры:

```
struct list
{
  int field; // поле данных
  struct list *next; // указатель на следующий элемент
  struct list *prev; // указатель на предыдущий элемент
};
```

Основные действия, производимые над узлами ДЛС:

- Инициализация списка
- Добавление узла в список
- Удаление узла из списка
- Удаление корня списка
- Вывод элементов списка
- Вывод элементов списка в обратном порядке
- Взаимообмен двух узлов списка

Инициализация ДЛС

Инициализация списка предназначена для создания корневого узла списка, у которого поля указателей на следующий и предыдущий узлы содержат нулевое значение.


```
1   struct list * init(int a)  // a- значение первого узла
2   {
3     struct list *lst;
4     // выделение памяти под корень списка
5     lst = (struct list*)malloc(sizeof(struct list));
6     lst->field = a;
7     lst->next = NULL; // указатель на следующий узел
8     lst->prev = NULL; // указатель на предыдущий узел
9     return(lst);
10  }
```

Добавление узла в ДЛС

Функция добавления узла в список принимает два аргумента:

- Указатель на узел, после которого происходит добавление
- Данные для добавляемого узла.

Процедуру добавления узла можно отобразить следующей схемой:

Добавление узла в ДЛС включает в себя следующие этапы:

- создание узла добавляемого элемента и заполнение его поля данных;
- переустановка указателя «следующий» узла, предшествующего добавляемому, на добавляемый узел;
- переустановка указателя «предыдущий» узла, следующего за добавляемым, на добавляемый узел;
- установка указателя «следующий» добавляемого узла на следующий узел (тот, на который указывал предшествующий узел);
- установка указателя «предыдущий» добавляемого узла на узел, предшествующий добавляемому (узел, переданный в функцию).

Таким образом, функция добавления узла в ДЛС имеет вид:

```
1   struct list * addelem(list *lst, int number)
2   {
3     struct list *temp, *p;
4     temp = (struct list*)malloc(sizeof(list));
5     p = lst->next; // сохранение указателя на следующий узел
6     lst->next = temp; // предыдущий узел указывает на создаваемый
7     temp->field = number; // сохранение поля данных добавляемого узла
8     temp->next = p; // созданный узел указывает на следующий узел
9     temp->prev = lst; // созданный узел указывает на предыдущий узел
10    if (p != NULL)
11      p->prev = temp;
12    return(temp);
13  }
```

Возвращаемым значением функции является адрес добавленного узла.

Удаление узла ДЛС

В качестве аргументов функции удаления узла ДЛС передается указатель на удаляемый узел. Поскольку узел списка имеет поле указателя на предыдущий узел, нет необходимости передавать указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым.

Удаление узла может быть представлено следующей схемой:

Удаление узла ДЛС включает в себя следующие этапы:

- установка указателя «следующий» предыдущего узла на узел, следующий за удаляемым;
- установка указателя «предыдущий» следующего узла на узел, предшествующий удаляемому;
- освобождение памяти удаляемого узла.

```
1   struct list * deletelem(list *lst)
2   {
3     struct list *prev, *next;
4     prev = lst->prev; // узел, предшествующий lst
5     next = lst->next; // узел, следующий за lst
6     if (prev != NULL)
7       prev->next = lst->next; // переставляем указатель
8     if (next != NULL)
9       next->prev = lst->prev; // переставляем указатель
10    free(lst); // освобождаем память удаляемого элемента
11    return(prev);
12  }
```

Удаление корня ДЛС

Функция удаления корня ДЛС в качестве аргумента получает указатель на текущий корень списка. Возвращаемым значением будет новый корень списка — тот узел, на который указывает удаляемый корень.

```
1   struct list * deletehead(list *root)
2   {
3     struct list *temp;
4     temp = root->next;
5     temp->prev = NULL;
6     free(root);   // освобождение памяти текущего корня
7     return(temp); // новый корень списка
8   }
```

Вывод элементов ДЛС

Функция вывода элементов ДЛС аналогична функции для ОЛС. В качестве аргумента в функцию вывода элементов передается указатель на корень списка.

Функция осуществляет последовательный обход всех узлов с выводом их значений.

```c
void listprint(list *lst)
{
  struct list *p;
  p = lst;
  do {
    printf("%d ", p->field); // вывод значения элемента p
    p = p->next; // переход к следующему узлу
  } while (p != NULL); // условие окончания обхода
}
```

Для ДЛС также можно вызвать функцию вывода элементов списка в обратном порядке.

Вывод элементов ДЛС в обратном порядке

Функция вывода элементов ДЛС в обратном порядке принимает в качестве аргумента указатель на корень списка. Функция перемещает указатель на последний узел списка и осуществляет последовательный обход всех узлов с выводом их значений в обратном порядке.

```c
void listprintr(list *lst)
{
  struct list *p;
  p = lst;
  while (p->next != NULL)
    p = p->next;  // переход к концу списка
  do {
    printf("%d ", p->field); // вывод значения элемента p
    p = p->prev; // переход к предыдущему узлу
  } while (p != NULL); // условие окончания обхода
}
```

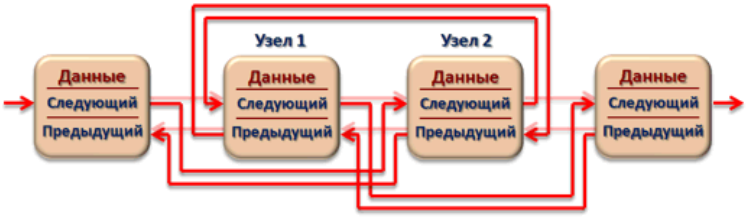
Взаимообмен узлов ДЛС

В качестве аргументов функция взаимообмена узлов ДЛС принимает два указателя на обмениваемые узлы, а также указатель на корень списка. Функция возвращает адрес корневого узла списка.

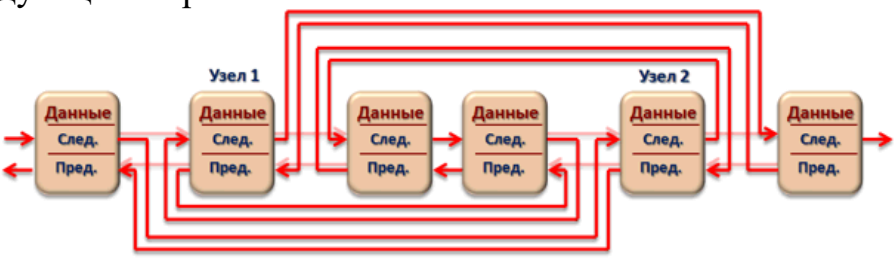
Взаимообмен узлов списка осуществляется путем переустановки указателей. Для этого необходимо определить предшествующий и последующий узлы для каждого заменяемого. При этом возможны две ситуации:

- заменяемые узлы являются соседями;
- заменяемые узлы не являются соседями, то есть между ними имеется хотя бы один узел.

При замене соседних узлов переустановка указателей выглядит следующим образом:



При замене узлов, не являющихся соседними переустановка указателей выглядит следующим образом:



Функция взаимообмена узлов списка выглядит следующим образом:

```
1   struct list * swap(struct list *lst1, struct list *lst2, struct list *head)
2   {
3     // Возвращает новый корень списка
4     struct list *prev1, *prev2, *next1, *next2;
5     prev1 = lst1->prev;  // узел предшествующий lst1
6     prev2 = lst2->prev;  // узел предшествующий lst2
7     next1 = lst1->next; // узел следующий за lst1
8     next2 = lst2->next; // узел следующий за lst2
9     if (lst2 == next1)  // обмениваются соседние узлы
10    {
11      lst2->next = lst1;
12      lst2->prev = prev1;
13      lst1->next = next2;
14      lst1->prev = lst2;
15      if(next2 != NULL)
16        next2->prev = lst1;
17      if (lst1 != head)
18        prev1->next = lst2;
19    }
20    else if (lst1 == next2)  // обмениваются соседние узлы
21    {
22      lst1->next = lst2;
23      lst1->prev = prev2;
24      lst2->next = next1;
25      lst2->prev = lst1;
26      if(next1 != NULL)
27        next1->prev = lst2;
28      if (lst2 != head)
```

```
29          prev2->next = lst1;
30        }
31        else  // обмениваются отстоящие узлы
32        {
33          if (lst1 != head)  // указатель prev можно установить только для элемента,
34            prev1->next = lst2; // не являющегося корневым
35          lst2->next = next1;
36          if (lst2 != head)
37            prev2->next = lst1;
38          lst1->next = next2;
39          lst2->prev = prev1;
40          if (next2 != NULL) // указатель next можно установить только для элемента,
41            next2->prev = lst1; // не являющегося последним
42          lst1->prev = prev2;
43          if (next1 != NULL)
44            next1->prev = lst2;
45        }
46        if (lst1 == head)
47          return(lst2);
48        if (lst2 == head)
49          return(lst1);
50        return(head);
51      }
```

! Дополнительный материал !

1. Списки в C++:
https://otus.ru/nest/post/911/
2. Связный список C#:
https://shwanoff.ru/linked-list/

2 Задания

Обязательное: *Создать двусвязный список с числами в диапазоне от* –50 *до* +50. *После создания списка выполнить индивидуальное задание и вывести результат. В конце работы все списки должны быть удалены*.

1. Найти минимальный элемент и сделать его первым.
2. Создать два списка. Первый должен содержать только положительные, а второй – только отрицательные числа.
3. Удалить из списка все элементы, находящиеся между его максимальным и минимальным значениями.
4. Переместить во второй список все элементы, находящиеся после элемента с максимальным значением.
5. Сместить по кольцу элементы списка на заданное число позиций.
6. Удалить все отрицательные элементы списка.
7. Из элементов, расположенных между максимальным и минимальным элементами, создать кольцо. Остальные элементы должны остаться в первом списке.
8. Удалить первый и последний элементы списка.

9. Удалить по одному элементу справа и слева от минимального элемента. Если элементы справа или слева отсутствуют, то удалить сам минимальный элемент.

10. Удалить элементы, заключенные между максимальным и минимальным значениями.

11. Удалить из списка элементы с повторяющимися более одного раза значениями.

12. Поменять местами элементы с максимальным и минимальным значениями, при этом элементы не должны перемещаться в памяти.

13. Преобразовать список в кольцо. Предусмотреть возможность движения по кольцу в обе стороны с отображением места положения текущего элемента. В конце работы восстановить начальный список.

14. Поменять местами первый и максимальный элементы списка.

15. Поменять местами первый и последний элементы списка.

! Контрольные вопросы !

1. Определение понятия список.
2. Классификация списков по количеству полей указателей.
3. Классификация списков по способу связи элементов.
4. Виды списков. Отличие списков.
5. Основные действия, производимые над элементами односвязного линейного списка.
6. Какие аргументы принимает функция добавления узла в список (ОЛС).
7. Сколько полей указателей содержит каждый узел двусвязного линейного списка?
8. Сколько полей указателей содержит каждый узел односвязного линейного списка?
9. Основные действия, производимые над элементами двусвязного линейного списка.

Содержание отчёта

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Ответы на контрольные вопросы.
6. Выводы.