1. Линейные динамические информационные структуры

Среди линейных динамических информационных структур (ЛДИС) наиболее используемыми являются линейные списки, стеки и очереди.

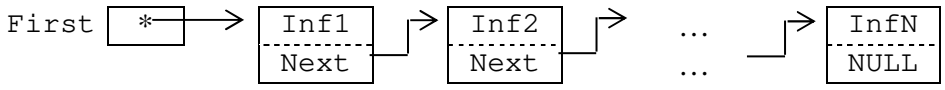
1.1.Линейный список

Наиболее простой способ объединить или связать некоторое множество элементов — это представить его в «линейном» виде, т.е. организовать список. В этом случае каждому элементу нужно сопоставить ссылку на следующий элемент. Для иллюстрации этих положений определим следующие тип:

```
struct list  //структура «линейный список»
{
    int inf;
    list *next;
};
```

Информационное поле `info` структуры `list` может содержать любую информацию произвольной структуры, тип которой в данном случае принципиального значения не имеет.

Графически линейный список можно представить так:

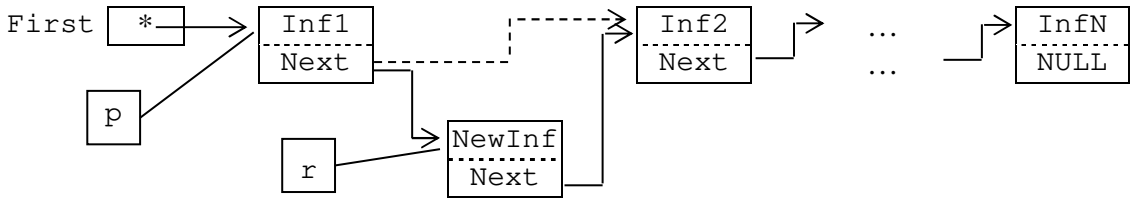


При описании любой динамической структуры данных необходимо определить возможные операции над ее элементами. Так, с элементами линейного списка выполняются следующие элементарные операции:

1. включение нового элемента в заданное место списка;
2. удаление элемента из списка;
3. поиск элемента в списке и пр.

При написании процедур, реализующих основные операции над списками, будем предполагать наличие приведенных ранее описаний типов.

Включение нового элемента. Вставка элемента `NewInf` в линейный список после элемента со ссылкой `p`, приведет к следующему переопределению ссылок:

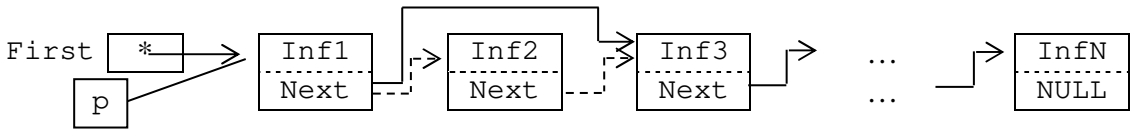


Для полноценного понимания организации работы с ЛДИС, необходимо изучить тему «Указатели» [1, стр. 121]

Процедура включения нового элемента после элемента со ссылкой `p` может выглядеть следующим образом:

```
// Включение элемента в список после элемента со ссылкой p
void NewElement(int x, list *&p)
{
     list *r = new(list);
     r->inf = x;
     r->next = NULL;
     p->next = r;
     p = r;
     return;
}
```

Удаление элемента из списка графически можно представить так:



И, соответственно, процедура **удаления**:

```
// Удаление элемента, следующего за элементом со ссылкой p
void Delete_List(list *&p)
{
     list *r;
     r = p->next;  // Сохраняем ссылку на удаляемый элемент
     p->next = p->next->next;
     delete r;
     return;
}
```

Вывод списка на экран:

```
void Print_List(list *p)  //Вывод списка
{
     while (p != NULL)
     {
          cout << p->info << '\t';
          p = p->next;
     }
     return;
}
```

В качестве примера приведем формирование списка целых чисел. Признак завершения ввода — 0.

```
void main()
{
     setlocale(LC_ALL, "rus");

     list *first, *p1, *r;
     int x;
     cout << "Введите элементы (признак завершения 0)" << endl;
     cin >> x;
     first = new (list);     // Создание списка
     first->inf = x;         // Создание первого элемента
     first->next = NULL;
     p1 = first;
     cin >> x;          // Включение в список остальных элементов
     while (x != 0)
     {
          NewElement(x, p1);
          cin >> x;
     }
     cout << "Итоговый список: " << endl;
     Print_List(first); // Вывод итогового списка
     cout << endl;

     cout << "Введите элемент для вставки x="; cin >> x;
     p1 = first;         // Поиск ссылки на последний элемент
     while ((p1->next != NULL))
     {
          p1 = p1->next;
     }
     NewElement(x, p1);

     cout << "Итоговый список: " << endl;
```

```
Print_List(first); // Вывод итогового списка
cout << endl;

cout << "Элемент для удаления x=";   cin >> x;

if (first->inf == x){
// Если это первый элемент, удаляем его
     r = first;
     first = first->next;
     delete r;
}
else
{    // Ищем среди остальных...
     p1 = first;   // Поиск ссылки на элемент x
     while ((p1->next != NULL) && (p1->next->inf != x))
     {
          p1 = p1->next;
     }
     if (p1->next != NULL) {
          Delete_List(p1);
     }
     else
     {
          cout << "!!! x=" << x << " не найден !!!" << endl;
     }

}

cout << "Итоговый список: " << endl;
Print_List(first); // Вывод итогового списка
cout << endl;

system("pause");
return;
}
```

1.2.Стек

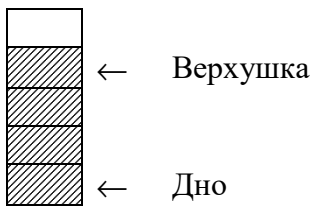
Стек — это ЛДИС, доступ к элементам которого определяется по принципу **LIFO** (Last In — First Out) - последним пришел, первым ушел.

Элемент или узел, который добавляется в стек последним, называется верхушкой стека. Позицию этого элемента также называют верхушкой. Элемент или узел, помещенный в стек первым называют дном стека. В любой момент времени в стеке доступен только один верхушечный элемент.

Функционально стек похож на детскую игрушку «пирамидка». Когда мы заносим элемент в стек (надеваем на стержень новое кольцо), то он помещается поверх прежней вершины (на предыдущее кольцо) и теперь сам находится на вершине стека; при этом выбранный элемент исключается из стека, а на его место становится предыдущий.

Над стеком обычно определены операции:

1. создать стек (инициализация)
2. добавить узел
3. извлечь узел
4. проверка на пустоту (булевская операция, возвращает `true` если в стеке есть хотя бы один узел)



Стек, как абстрактное хранилище информации имеет бесконечный объем. Однако на практике имеющийся в распоряжении программы объем памяти всегда конечен и, следовательно, можно использовать только стек ограниченного объема, так называемый ограниченный стек. Очевидно, что для любого стека операция «извлечь из стека» определена только тогда, когда стек не пуст, а операция «добавить в стек» является частичной для ограниченного стека. Ситуация, в которой делается попытка добавить в ограниченный стек узел, а имеющаяся память уже исчерпана называется операцией переполнения.

Рассмотрим реализацию основных операций над стеком.

Создание стека.

```
// Создание стека
void Init_Stack(stack *&pstack)
{
     pstack = new(stack);
     pstack = NULL;
     return;
}
```

Проверка стека на пустоту.

```
// Проверка стека на пустоту
bool Empty_Stack(stack *pstack)
{
     if (pstack == NULL) {
          cout << "Стек пуст !!! " << endl;
          return true;
     }
     else
          return false;
}
```

Включение нового элемента в стек. Поскольку при решении задачи может использоваться несколько разных стеков, то процедура вставки должна иметь два параметра: сам вносимый элемент и ссылку на нужный стек.

Описание такой процедуры может иметь вид:

```
// Включение элемента в стек
void Push_Stack(int NewInf, stack *&pstack)
{
     stack *r = new(stack); // Создаем новый элемент
     r->inf = NewInf;       // В инф. поле вносим значение
     r->next = pstack;
     pstack = r;            // Созданный элемент – вершина стека
     return;
}
```

Извлечение элемента из стека. При выполнении этой операции информационная часть элемента, находящегося на вершине стека, должна быть присвоена некоторой переменной и весь элемент удален.

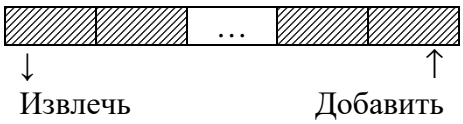
```
// Извлечение элемента из стека
void Out_Stack(int &x, stack *&pstack)
{
     if (!Empty_Stack(pstack)) {
          stack *r;
          x = pstack->inf;   // Извлекаем элемент
          r = pstack;        // Запоминаем ссылку
          pstack = pstack->next; // Исключаем и
          delete r;     // уничтожаем извлеченный элемент
     }
     return;
}
```

Пример использования основных операций над стеком.

```
int main()
{
     setlocale(LC_ALL, "rus");

     stack *head, *p1;
     int x;
     Init_Stack(head);
     cout << "Стек создан..." << endl;

     p1 = head;
     cout << "Элемент стека x="; cin >> x;
     while (x != 0)
     {
          Push_Stack(x, head);        // Включение в стек
          cout << "Элемент стека x="; cin >> x;
     }
     cout << "Элементы стека: " << endl;
     Print_Stack(head);  // Вывод элементов стека

     cout << "Введите элемент для вставки x="; cin >> x;
     Push_Stack(x, head);        // Включение в стек
     cout << "Элементы стека: " << endl;
     Print_Stack(head);  // Вывод элементов стека

     Out_Stack(x, head);
     cout << "Извлекли x=" << x << endl;
     cout << "Элементы стека: " << endl;
     Print_Stack(head);  // Вывод элементов стека
```

```
    Out_Stack(x, head);
    cout << "Извлекли x=" << x << endl;
    cout << "Элементы стека: " << endl;
    Print_Stack(head);  // Вывод элементов стека

    system("pause");
    return 0;
}
```

1.3.Очередь



Это ЛДИС, доступ к узлам которой осуществляется по схеме **FIFO** (First In — First Out: первым пришел, первым ушел). Над очередью определены такие же операции, что и над стеком. Аналогично стеку определяется понятие ограниченной очереди. Можно говорить, что у очереди есть начало и есть конец. Добавляются элементы в начало очереди, а удаляются из конца.

Приведем реализацию основных операций с очередью.

Создание очереди полностью совпадает с процедурой инициализации стека:

```
// Инициализация очереди и включение первого элемента
void Init_Queue(int NewInf, queue *&phead, queue *&ptail)
{
    ptail = new(queue);
    ptail->inf = NewInf;
    ptail->next = NULL;
    phead = ptail;
    return;
}
```

Проверка очереди на пустоту:

```
// Проверка очереди на пустоту
bool Empty_Queue(queue *phead)
{
    if (phead == NULL) {
        cout << "Очередь пуста !!! " << endl;
        return true;
    }
    else
        return false;
}
```

Включение нового элемента в очередь полностью совпадает с процедурой включения элемента в стек:

```
// Включение элемента в очередь
void Push_Queue(int NewInf, queue *&ptail)
{
     queue *r = new(queue); // Создаем новый элемент
     r->inf = NewInf;   // В информационное поле вносим значение
     r->next = NULL;
     ptail->next = r;   // Созданный элемент – начало очереди
     ptail = r;
     return;
}
```

Извлечение элемента из очереди. При выполнении этой операции необходимо найти указатель на последний элемент, извлечь информационную часть элемента и удалить весь элемент.

```
// Извлечение элемента из очереди
void Out_Queue(int &x, queue *&phead)
{
     // Если очередь не пуста
     if (!Empty_Queue(phead)) {
          queue *r;
          x = phead->inf;    // Извлекаем значение элемента
          r = phead;// Сохраняем указатель на "голову" очереди
          phead = phead->next;// "Сдвигаем" указатель
          delete r;     // уничтожаем извлеченный элемент
     }
     return;
}
```

Пример использования основных операций с очередью.

```
int main()
{
     setlocale(LC_ALL, "rus");
     queue *head, *tail;
     int x, k=0;   // k – номер очередного элемента

     // Инициализация очереди и включение первого элемента
     k++; cout << k << "-й Элемент очереди x=";     cin >> x;
     Init_Queue(x, head, tail);
     cout << "очередь создана..." << endl;
```

```
// Включение остальных элементов
k++; cout << k << "-й Элемент очереди x=";    cin >> x;
while (x != 0)
{
     Push_Queue(x, tail);   // Включение в очередь
     k++; cout << k << "-й Элемент очереди x=";    cin >> x;
}
cout << "Элементы очереди: " << endl;
Print_Queue(head);     // Вывод элементов очереди

cout << "Введите элемент для вставки x=";     cin >> x;
Push_Queue(x, tail);   // Включение в очередь

cout << "Элементы очереди: " << endl;
Print_Queue(head);     // Вывод элементов очереди

Out_Queue(x, head);    // Извлечение элемента из очереди
cout << "Извлекли x=" << x << endl;

cout << "Элементы очереди: " << endl;
Print_Queue(head);     // Вывод элементов очереди

system("pause");
return 0;
}
```

Контрольные вопросы и задания

1. Описать процедуру, которая
   а) вставляет в список L новый элемент A за каждым вхождением элемента E.
   б) удаляет из списка L все отрицательные элементы.
2. Описать процедуру, которая
   а) вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом
   б) удаляет из списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.
3. Описать процедуру, которая
   а) вставляет в непустой список L, элементы которого упорядочены по не убыванию новый элемент E так, чтобы сохранилась упорядоченность.
   б) удаляет из списка L первый отрицательный элемент.
4. Описать процедуру, которая
   а) проверяет, есть ли в списке L хотя бы два одинаковых элемента.
   б) добавляет в конец списка L1 все элементы списка L2.
5. Описать процедуру, которая вставляет в список L за первым вхождением элемента E все элементы списка L1, если E входит в L.
6. Описать процедуру, которая
   а) переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке.
   б) подсчитывает число вхождений элемента E в список L.
7. Описать процедуру, которая
   а) формирует список L, включив в него по одному разу элементы, которые входят хотя бы в один из списков L1 и L2.
   б) удаляет из непустых слов списка L их первые буквы.
8. Описать процедуру, которая
   а) формирует список L, включив в него по одному разу элементы, которые входят одновременно в оба списка L1 и L2.
   б) определяет количество слов в непустом списке L, отличных от последнего.
9. Описать процедуру, которая
   а) формирует список L, включив в него по одному разу элементы, которые входят в список L1, но не входят в список L2.
   б) в списке L переставляет местами первое и последнее непустые слова, если в L есть хотя бы два непустых слова.
10. Описать процедуру, которая
    а) формирует список L, включив в него по одному разу элементы, которые входят в один из списков L1 и L2, но в тоже время не входят в другой из них.
    б) подсчитывает количество слов списка L, у которых первая и последняя буквы совпадают.
11. Описать процедуру, которая объединяет два упорядоченных по не убыванию списка L1 и L2 в один упорядоченный по не убыванию список меняя соответствующим образом ссылки в L1 и L2 и присвоив полученный список параметру L1.
12. Описать процедуру, которая в списке L заменяет первое вхождение списка L1 (если такое есть) на список L2.
13. Описать процедуру, которая проверяет на равенство два многочлена (указывается коэффициент и степень каждого элемента).
14. Описать процедуру, которая вычисляет значение многочлена в некоторой целочисленной точке.
15. Описать процедуру, которая строит многочлен p — сумму многочленов q и r.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Основы программирования на языке С++: учеб.-метод. пособие /авт. В.В. Подколзин, Е.П. Лукащик, Н.Ю. Добровольская, О.В. Гаркуша, С.Г. Синица, А.А. Полупанов, А.Н. Полетайкин, Р.Х. Багдасарян, А.В. Харченко, А.А. Михайличенко, А.В. Уварова – Краснодар: Кубанский гос. ун-т, 2019. – 156 с.