Основные операции по работе с односвязным линейным списком

Ими являются:

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

Кроме того, можно выделить ряд вспомогательных методов:

* Переход к следующему узлу
* Очистка списка
* Проверка, пуст ли список
* Получение количества элементов в списке
* Получение корня списка
* Получение последнего элемента списка
* Получение значения элемента списка
* Установка значения элемента списка

Узел списка представляет собой класс, содержащий два скрытых поля — значение и указатель на следующий элемент. Чтобы позволить списку иметь доступ к скрытым полям узла, объявим класс **список** как дружественный.

class Node  
{  
  int field;  
  class Node \*ptr;  
  friend class List;  
};

**Класс «Список»**

Узел является составной частью класса, который также реализует все интерфейсные методы списка, перечисленные выше.

class List  
{  
  Node \*head;    // Корень списка  
  int count = 0; // Количество узлов списка  
  Node\* Prev(Node \*); // Переход к предыдущему узлу (не интерфейсный метод)  
public:  
  List() { head = NULL; } // Инициализация списка  
  int getCount() { return count; } // Получение количества узлов списка  
  bool isEmpty() { return head == NULL; }  // Проверка, пуст ли список  
  int getValue(Node\* p) { return p->field; } // Получение значения узла списка  
  void setValue(Node \*p, int val) { p->field = val; } // Установка значения узла списка  
  Node\* getFirst() { return head; } // Получение корневого узла списка  
  Node\* getLast();      // Получение последнего узла списка  
  void Clear();        // Очистка списка  
  Node\* Next(Node \*);      // Переход к следующему узлу  
  Node\* Add(int, Node\*);    // Добавление узла списка  
  Node\* Delete(Node\*);    // Удаление узла списка  
  void Print();        // Вывод значений узлов списка  
  void Swap(Node\*, Node\*);  // Взаимообмен двух узлов  
};

**Добавление узла списка**

Для добавления необходимо рассмотреть два случая:

* Добавление узла после указанного
* Добавление узла в начало списка

Node\* List::Add(int num, Node\* node = NULL)  
{  
  Node \*elem = new Node();  
  elem->field = num;  
  count++;  
  if (node == NULL) // Добавление нового корня  
  {  
    if (head == NULL) {  
      elem->ptr = NULL;  
      head = elem;  
    }  
    else {  
      elem->ptr = head;  
      head = elem;  
    }  
    return elem;  
  }  
  elem->ptr = node->ptr; // Добавление узла после текущего  
  node->ptr = elem;  
  return elem;  
}

Метод принимает в качестве аргумента текущий узел со значением по умолчанию, равным NULL. Если узел не передается, добавление нового узла происходит в начало списка.

**Удаление узла**

Для удаления узла также необходимо рассмотреть две ситуации:

* Удаление промежуточного узла
* Удаление корневого узла

Node\* List::Delete(Node\* node)  
{  
  if (node == NULL) { return NULL; } // В списке нет узлов  
  count--;  
  if (node == head)  // Удаление корневого узла  
  {  
    head = node->ptr;  
    delete node;  
    return head;  
  }  
  Node\* prev = Prev(node); // Удаление промежуточного узла  
  prev->ptr = node->ptr;  
  delete node;  
  return prev;  
}

**Перемещение к следующему и к предыдущему узлу**

Node\* List::Next(Node\* node)  
{  
  if (isEmpty()) return NULL;  
  return node->ptr;  
}

Перемещение к предыдущему узлу не является интерфейсным методом ОЛС, поэтому находится в области видимости private.

Node\* List::Prev(Node\* node)  
{  
  if (isEmpty()) return NULL;  
  if (node == head) return NULL;  
  Node \*p = head;  
  while (p->ptr != node)  
    p = p->ptr;  
  return p;  
}

**Получение последнего узла списка**

Node\* List::getLast()  
{  
  Node\* p = head;  
  while (Next(p) != NULL)  
    p = Next(p);  
  return p;  
}

**Очистка списка**

void List::Clear()  
{  
  class Node \*p = head;  
  if (p == NULL) return;  
  do {  
    Node \*d = p;  
    p = Next(p);  
    delete d;  
  } while (p != NULL);  
  count = 0;  
  head = NULL;  
}

**Вывод значений узлов списка**

void List::Print()  
{  
  if (isEmpty()) { cout << "Список пуст" << endl; return; }  
  Node \*p = head;  
  do {  
    cout << getValue(p) << " ";  
    p = Next(p);  
  } while (p != NULL);  
  cout << endl;  
}

**Взаимообмен узлов списка**

Здесь тоже необходимо рассмотреть две ситуации:

* меняются соседние узлы  
    
  При этом узлы могут быть переданы в любом порядке относительно корня списка. Для упрощения реализации функции поменяем узлы так, чтобы node1 стоял перед node2
* меняются отстоящие узлы

void List::Swap(Node\* node1, Node\* node2)  
{  
  if (node1 == NULL || node2 == NULL) return; // не допускаем обмен с несуществующим узлом  
  if (node1 == node2) return; // если один узел указан дважды, менять ничего не надо  
  if (node2->ptr == node1) // если node2 находится перед node1, меняем их местами  
  {  
    Node \*p = node1;  
    node1 = node2;  
    node2 = p;  
  }  
  Node \*prev1 = Prev(node1);  
  Node \*prev2 = Prev(node2);  
  Node \*next1 = Next(node1);  
  Node \*next2 = Next(node2);  
  if (next1 == node2) // обмен соседних узлов  
  {  
    if (prev1 != NULL)  
      prev1->ptr = node2;  
    else  
      head = node2;  
    node2->ptr = node1;  
    node1->ptr = next2;  
    return;  
  }  
  if (prev1 != NULL)  // обмен отстоящих узлов  
    prev1->ptr = node2;  
  else  
    head = node2;  
  if (prev2 != NULL)  
    prev2->ptr = node1;  
  else  
    head = node1;  
  node2->ptr = next1;  
  node1->ptr = next2;  
}

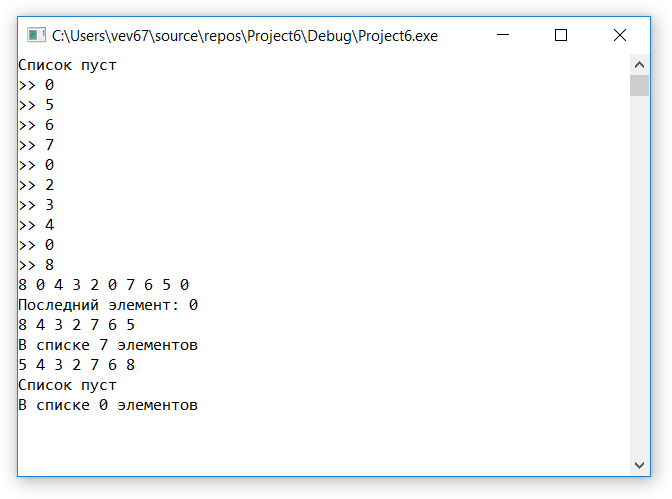
**Пример**

В качестве примера использования ОЛС рассмотрим следующую задачу.

* Создать список из 10 элементов.
* Удалить все элементы, равные нулю.
* Поменять местами первый и последний элементы.

int main()  
{  List list;  
  list.Print();  
  // Создаем список, помещаем элементы в начало  
  for (int i = 0; i < 10; i++)   
  {  
    int z;  
    cout << ">> ";  
    cin >> z;  
    list.Add(z);  
  }  
  list.Print();  
  cout << "Последний элемент: " << list.getValue(list.getLast()) << endl;  
  // Удаляем элементы, равные 0  
  Node \*p = list.getFirst();  
  do {  
    if (list.getValue(p) == 0)  
      p = list.Delete(p);  
    else  
      p = list.Next(p);  
  } while (p != NULL);  
  list.Print();  
  cout << "В списке " << list.getCount() << " элементов" << endl;  
  list.Swap(list.getFirst(), list.getLast());  
  list.Print();  
  list.Clear();  
  list.Print();  
  cout << "В списке " << list.getCount() << " элементов" << endl;  
  cin.get(); cin.get();  
  return 0;  
}

Как видим, новые элементы добавляются в начало списка.  
Если необходимо добавлять элементы в конец списка, изменения коснутся строк 7, 13:



int main()  
{  List list;  
  list.Print();  
  Node \*s = list.getLast();  
  for (int i = 0; i < 10; i++) {  
    int z;  
    cout << ">> ";  
    cin >> z;  
    s = list.Add(z, s);  
  }  
  list.Print();  
  cout << "Последний элемент: " << list.getValue(list.getLast()) << endl;  
  // Удаляем элементы, равные 0  
  Node \*p = list.getFirst();  
  do {  
    if (list.getValue(p) == 0)  
      p = list.Delete(p);  
    else  
      p = list.Next(p);  
  } while (p != NULL);  
  list.Print();  
  cout << "В списке " << list.getCount() << " элементов" << endl;  
  list.Swap(list.getFirst(), list.getLast());  
  list.Print();  
  list.Clear();  
  list.Print();  
  cout << "В списке " << list.getCount() << " элементов" << endl;  
  cin.get(); cin.get();  
  return 0;  
}