ЛЕКЦИЯ №2

ДИНАМИЧЕСКИЕ СТРУКТУРЫ

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Это структуры данных, память под которые выделяется и освобождается по мере необходимости.

ОСОБЕННОСТИ ДИНАМИЧЕСКОЙ СТРУКТУРЫ ДАННЫХ

- ✓ не имеет имени;
- ✓ ей выделяется память в процессе выполнения программы;
- ✓ количество элементов структуры может не фиксироваться;
- ✓ размерность структуры может меняться в процессе выполнения программы;
- ✓ в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

НЕОБХОДИМОСТЬ В ДИНАМИЧЕСКИХ СТРУКТУРАХ ВОЗНИКАЕТ В СЛЕДУЮЩИХ СЛУЧАЯХ:

- ✓ Используются переменные, имеющие довольно большой размер (например, массивы большой размерности), необходимые в одних частях программы и совершенно не нужные в других.
- ✓ В процессе работы программы нужен массив, список или иная структура, размер которой изменяется в широких пределах и трудно предсказуем.
- ✓ Когда размер данных, обрабатываемых в программе, превышает объем сегмента данных.

ДОСТОИНСТВА СВЯЗНОГО ПРЕДСТАВЛЕНИЯ ДАННЫХ

- ✓ размер структуры ограничивается только доступным объемом машинной памяти;
- ✓ при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей;
- ✓ большая гибкость структуры.

НЕДОСТАТКИ СВЯЗНОГО ПРЕДСТАВЛЕНИЯ

- ✓ на поля, содержащие указатели для связывания элементов друг с другом, расходуется дополнительная память;
- ✓ доступ к элементам связной структуры может быть менее эффективным по времени.

ПОРЯДОК РАБОТЫ С ДИНАМИЧЕСКИМИ СТРУКТУРАМИ ДАННЫХ

1. создать (отвести место в динамической памяти);
2. работать при помощи указателя;
3. удалить (освободить занятое структурой место).

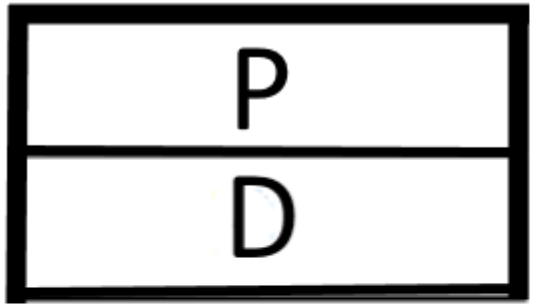
КЛАССИФИКАЦИЯ ДИНАМИЧЕСКИХ СТРУКТУР ДАННЫХ

- однонаправленные (односвязные) списки;
- двунаправленные (двусвязные) списки;
- циклические списки;
- стек;
- дек;
- очередь;
- бинарные деревья.

ЗАМЕТИМ ТАК ЖЕ, ЧТО ДИНАМИЧЕСКИЕ СТРУКТУРЫ МОГУТ:

- **отличатся способом связи отдельных элементов и/или допустимыми операциями;**
- **занимать несмежные участки оперативной памяти;**
- **применяться для более эффективной работы с данными, размер которых известен, особенно для решения задач сортировки.**

ОБЪЯВЛЕНИЕ ДИНАМИЧЕСКИХ СТРУКТУР ДАННЫХ



где *поле* `P` – *указатель*; *поле* `D` – данные.

ЭЛЕМЕНТ ДИНАМИЧЕСКОЙ СТРУКТУРЫ СОСТОИТ ИЗ

информационного поля (поля данных), в котором содержатся те данные, ради которых и создается структура; в общем случае информационное поле само является интегрированной структурой – вектором, массивом, другой динамической структурой и т.п.;

адресного поля (поля связок), в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры;

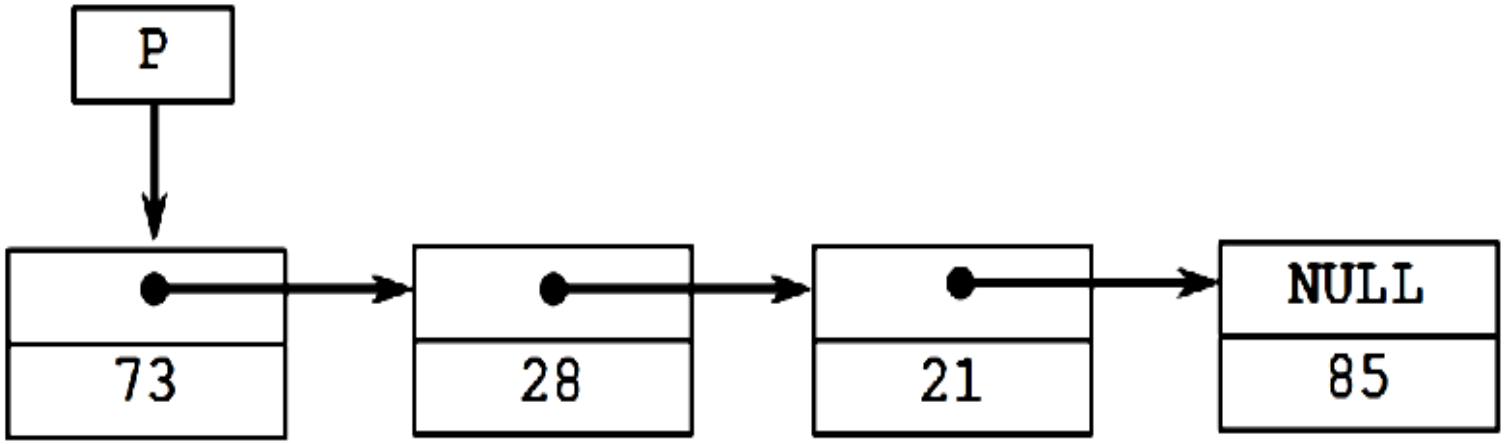
ОБЪЯВЛЕНИЕ ЭЛЕМЕНТА ДИНАМИЧЕСКОЙ СТРУКТУРЫ

```
struct имя_типа {
        информационное поле;
        адресное поле;
      };
```

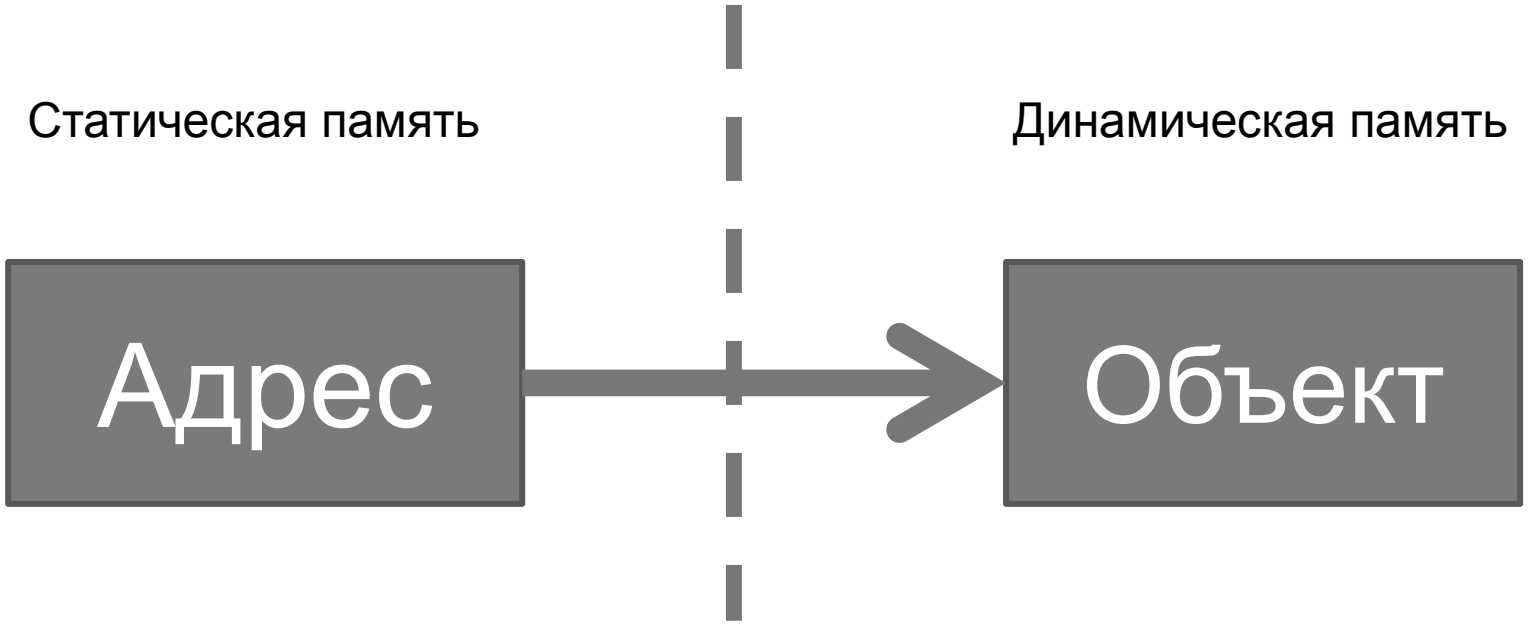
Например:

```
struct TNode {
        int Data;//информационное поле
        TNode *Next;//адресное поле
      };
```

ПРИМЕР ДИНАМИЧЕСКОЙ СТРУКТУРЫ



ДОСТУП К ДАННЫМ В ДИНАМИЧЕСКИХ СТРУКТУРАХ



Связь указателя с адресуемым объектом

ДОСТУП К ДАННЫМ В ДИНАМИЧЕСКИХ СТРУКТУРАХ

При помощи двуместной операции "стрелка" (->).

```
УказательНаСтруктуру-> ИмяЭлемента
```

Например:

```
p->Data;
p->Next;
```

РАБОТА С ПАМЯТЬЮ ПРИ ИСПОЛЬЗОВАНИИ ДИНАМИЧЕСКИХ СТРУКТУР

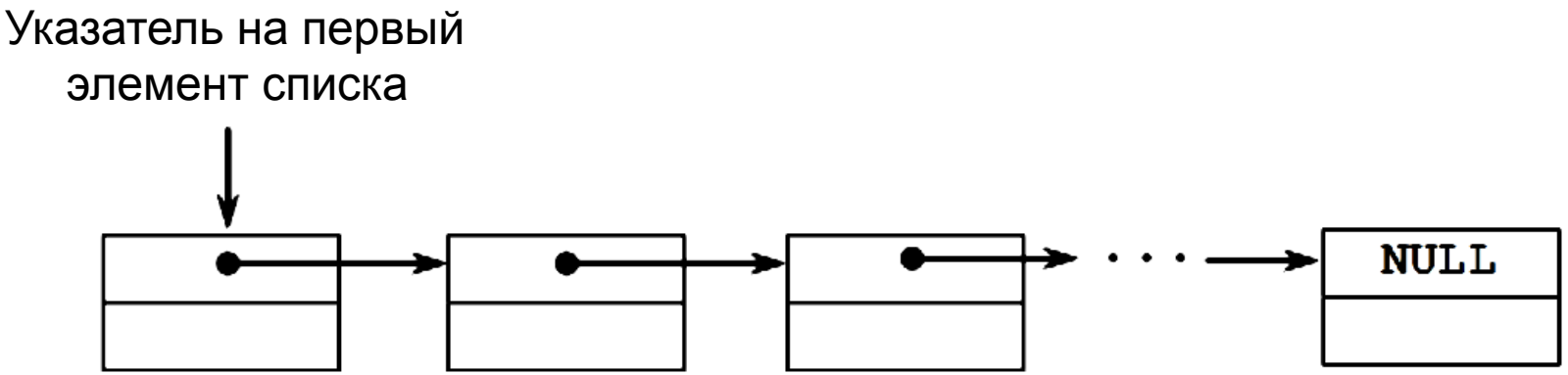
```
struct Node {char *Name;
             int Value;
             Node *Next
            };
Node *PNode; //объявляется указатель

PNode = new Node; //выделяется память

PNode->Name = "STO"; //присваиваются значения
PNode->Value = 28;
PNode->Next = NULL;

delete PNode; // освобождение памяти
```

ЛИНЕЙНЫЕ СВЯЗНЫЕ СПИСКИ

- ✓ однонаправленные (односвязные) списки;
- ✓ двунаправленные (двусвязные) списки;
- ✓ циклические (кольцевые) списки.

ОДНОНАПРАВЛЕННЫЕ (ОДНОСВЯЗНЫЕ) СПИСКИ

это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент равен NULL



ОПИСАНИЕ ПРОСТЕЙШЕГО ЭЛЕМЕНТА ТАКОГО СПИСКА

```
struct имя_типа { информационное поле; адресное поле; };
```

где информационное поле – это поле любого, ранее объявленного или стандартного, типа;

адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

ПРИМЕР

```
struct Node {
              int key;//информационное поле
              Node*next;//адресное поле
             };
```

Информационных полей может быть несколько.

```
struct point {
               char*name;//информационное поле
               int age;//информационное поле
               point*next;//адресное поле
              };
```

ОСНОВНЫЕ ОПЕРАЦИИ НАД СПИСКОМ

- ✓ создание списка;
- ✓ печать (просмотр) списка;
- ✓ вставка элемента в список;
- ✓ удаление элемента из списка;
- ✓ поиск элемента в списке
- ✓ проверка пустоты списка;
- ✓ удаление списка.

ОБЪЯВЛЕНИЕ

```
struct Single_List {//структура данных
                     int Data; //информационное поле
                     Single_List *Next; //адресное поле
                    };
. . . . . .
Single_List *Head; //указатель на первый элемент списка
. . . . . .
Single_List *Current; //указатель на текущий элемент
списка (при необходимости)
```

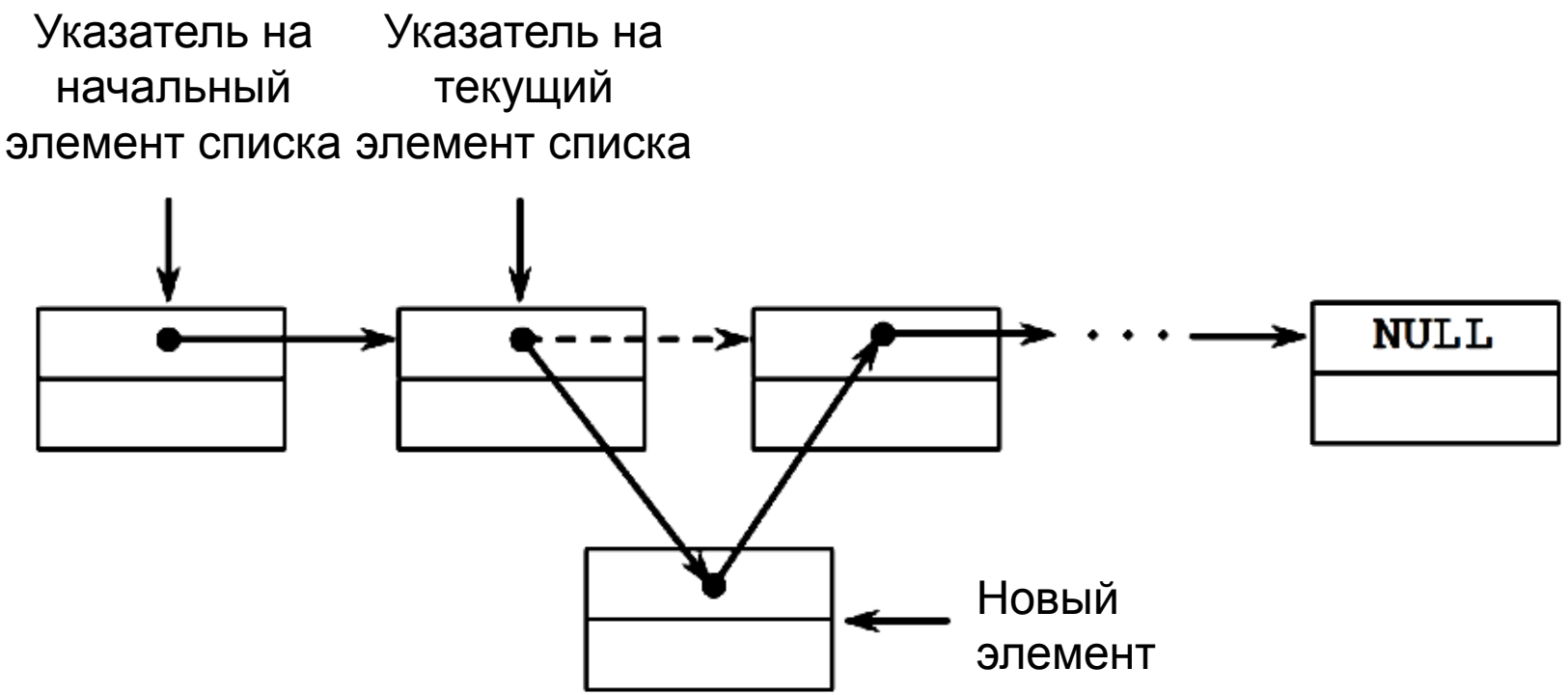
СОЗДАНИЕ ОДНОНАПРАВЛЕННОГО СПИСКА

```
//создание однонаправленного списка (добавления в конец)
void Make_Single_List(int n,Single_List** Head){
  if (n > 0) {
    (*Head) = new Single_List();
    //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Next=NULL;//обнуление адресного поля
     Make_Single_List(n-1,&((*Head)->Next));
  }
}
```

ПЕЧАТЬ (ПРОСМОТР) ОДНОНАПРАВЛЕННОГО СПИСКА

```
//печать однонаправленного списка
void Print_Single_List(Single_List* Head) {
  if (Head != NULL) {
    cout << Head->Data << "\t";
    Print_Single_List(Head->Next);
    //переход к следующему элементу
  }
  else cout << "\n";
}
```

ВСТАВКА ЭЛЕМЕНТА В ОДНОНАПРАВЛЕННЫЙ СПИСОК



ВСТАВКА ЭЛЕМЕНТА

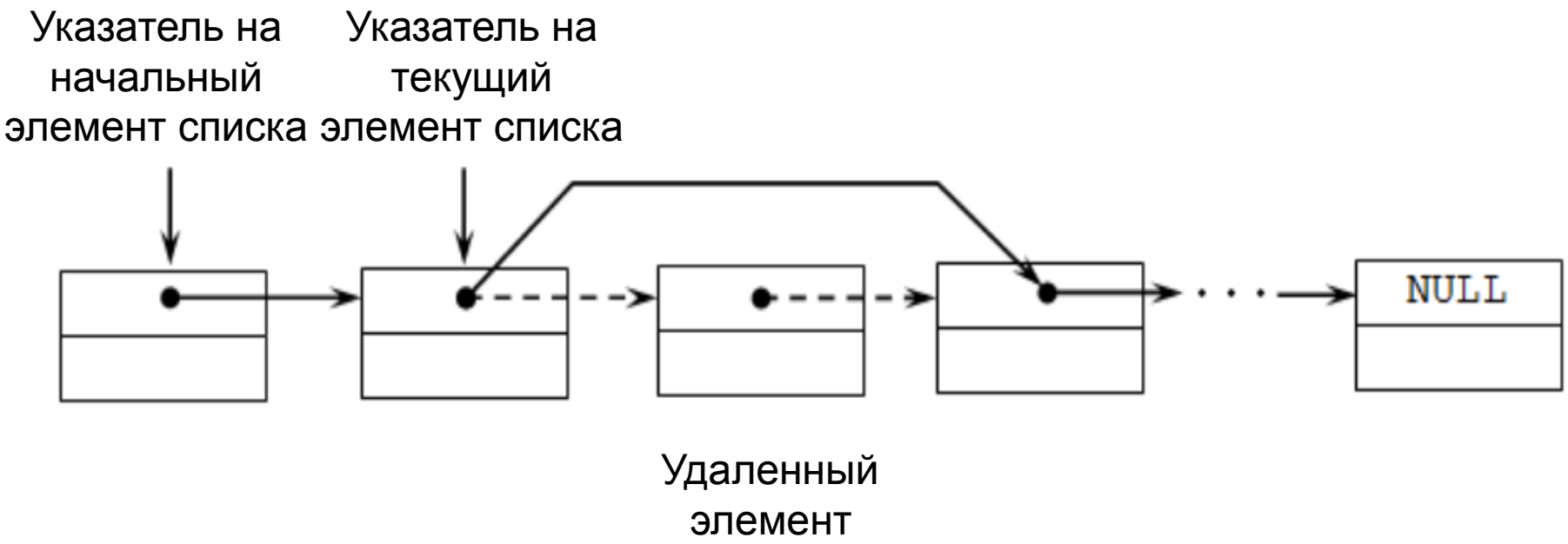
```
/*вставка элемента с заданным номером в
однонаправленный список*/
Single_List* Insert_Item_Single_List(Single_List* Head,
     int Number, int DataItem){
  Number--;
  Single_List *NewItem=new(Single_List);
  NewItem->Data=DataItem;
  NewItem->Next = NULL;
  if (Head == NULL) {//список пуст
    Head = NewItem;//создаем первый элемент списка
  }
else {//список не пуст
    Single_List *Current=Head;
```

ВСТАВКА ЭЛЕМЕНТА

Продолжение 26

```
for(int i=1; i < Number && Current->Next!=NULL; i++)
    Current=Current->Next;
    if (Number == 0){
    //вставляем новый элемент на первое место
      NewItem->Next = Head;
      Head = NewItem;
    }
    else {//вставляем новый элемент на непервое место
      if (Current->Next != NULL)
            NewItem->Next = Current->Next;
      Current->Next = NewItem;
    }
  }
  return Head;
}
```

УДАЛЕНИЕ ЭЛЕМЕНТА ИЗ ОДНОНАПРАВЛЕННОГО СПИСКА



УДАЛЕНИЕ ЭЛЕМЕНТА ИЗ ОДНОНАПРАВЛЕННОГО СПИСКА

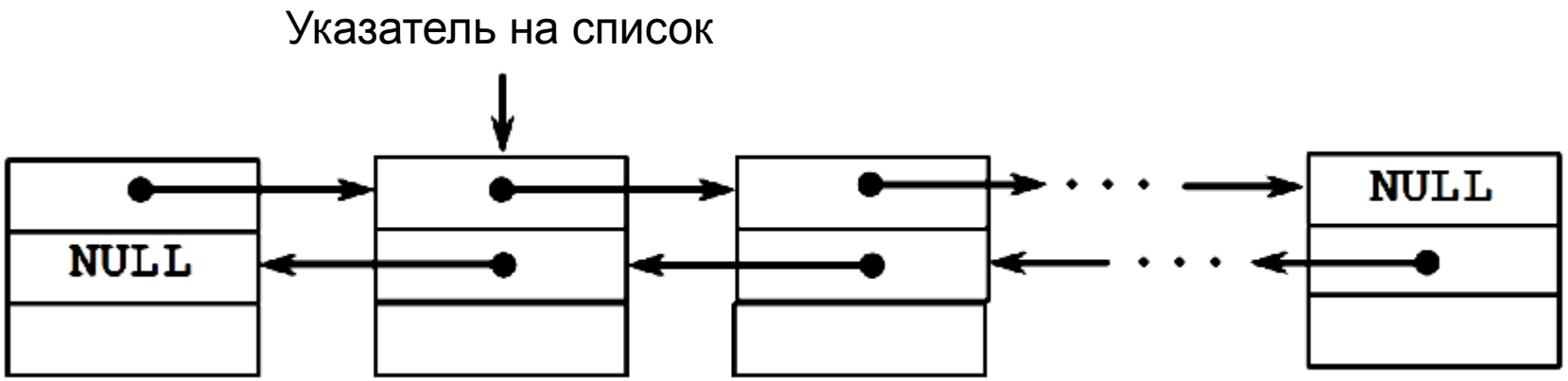
```
/*удаление элемента с заданным номером из однонаправленного списка*/

Single_List* Delete_Item_Single_List(Single_List* Head,
      int Number){
  Single_List *ptr;//вспомогательный указатель
  Single_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current == Head){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Current = Head;
    }
    else {//удаляем не первый элемент
      ptr = Head;
      while (ptr->Next != Current)
        ptr = ptr->Next;
      ptr->Next = Current->Next;
      delete(Current);
      Current=ptr;
    }
  }
return Head;
}
```

УДАЛЕНИЕ ОДНОНАПРАВЛЕННОГО СПИСКА

```
/*освобождение памяти, выделенной под однонаправленный список*/
void Delete_Single_List(Single_List* Head){
  if (Head != NULL){
    Delete_Single_List(Head->Next);
    delete Head;
  }
}
```

ДВУНАПРАВЛЕННЫЕ (ДВУСВЯЗНЫЕ) СПИСКИ

это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы. При этом два соседних элемента должны содержать взаимные ссылки друг на друга.



ОПИСАНИЕ ПРОСТЕЙШЕГО ЭЛЕМЕНТА ДВУХСВЯЗНОГО СПИСКА

```
struct имя_типа {
        информационное поле;
        адресное поле 1;
        адресное поле 2;
      };
```

ПРИМЕР ОПИСАНИЯ

```
struct list {
        type elem ;
        list *next, *pred ;
       }
list *headlist ;
```

где type – тип информационного поля элемента списка;

***next, *pred – указатели на следующий и предыдущий элементы этой структуры соответственно.**

Переменная-указатель headlist задает список как единый программный объект, ее значение – указатель на первый (или заглавный) элемент списка.

ОСНОВНЫЕ ОПЕРАЦИИ
НАД ДВУСВЯЗНЫМ (ДВУНАПРАВЛЕННЫМ) СПИСКОМ

- ✓ создание списка;
- ✓ печать (просмотр) списка;
- ✓ вставка элемента в список;
- ✓ удаление элемента из списка;
- ✓ поиск элемента в списке;
- ✓ проверка пустоты списка;
- ✓ удаление списка.

ОБЪЯВЛЕНИЕ ДСС

```
struct Double_List {//структура данных
                int Data; //информационное поле
                Double_List *Next, //адресное поле
                        *Prior; //адресное поле
              };
. . . . . .
Double_List *Head; //указатель на первый элемент списка
. . . . . .
Double_List *Current;
//указатель на текущий элемент списка (при необходимости)
```

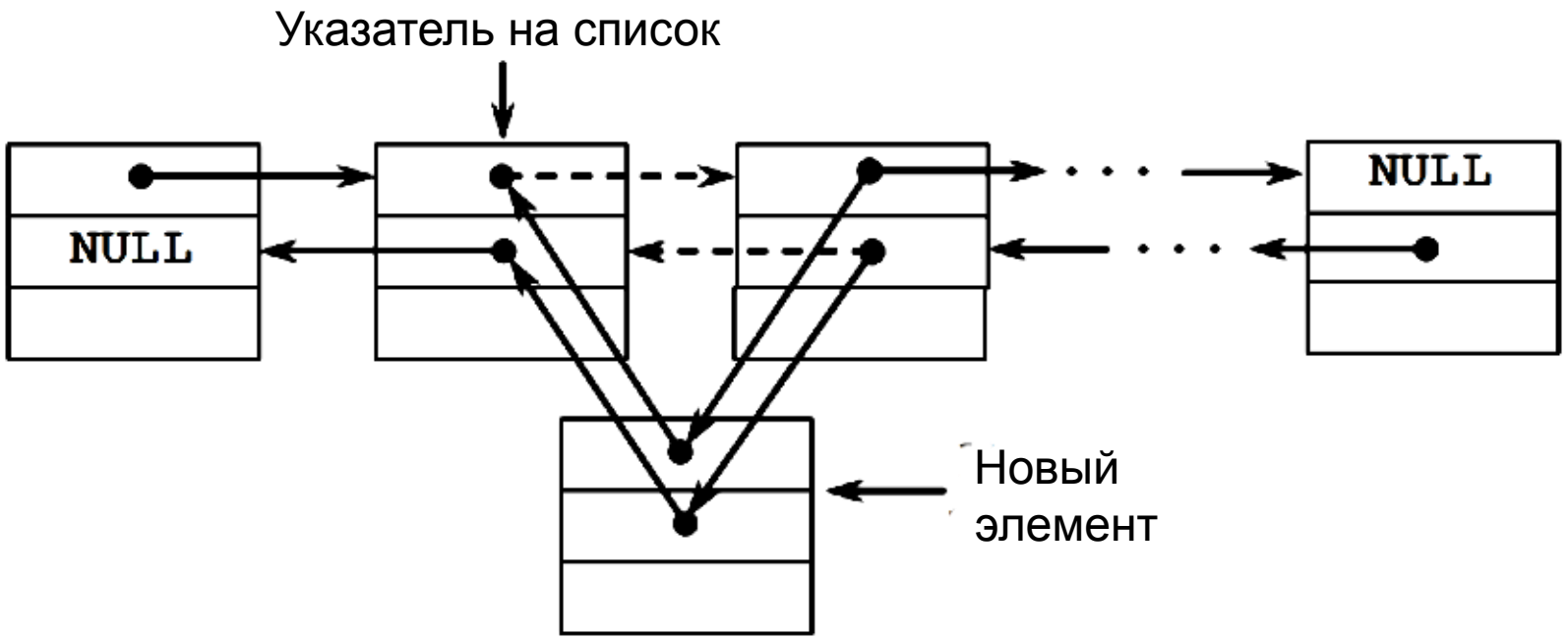
СОЗДАНИЕ ДВУНАПРАВЛЕННОГО СПИСКА

```
//создание двунаправленного списка (добавления в конец)
void Make_Double_List(int n,Double_List** Head,
        Double_List* Prior){
  if (n > 0) {
    (*Head) = new Double_List();
     //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Prior = Prior;
    (*Head)->Next=NULL;//обнуление адресного поля
    Make_Double_List(n-1,&((*Head)->Next),(*Head));
  }
  else (*Head) = NULL;
}
```

ПЕЧАТЬ (ПРОСМОТР) ДВУНАПРАВЛЕННОГО СПИСКА

```
//печать двунаправленного списка
void Print_Double_List(Double_List* Head) {
  if (Head != NULL) {
    cout << Head->Data << "\t";
    Print_Double_List(Head->Next);
    //переход к следующему элементу
  }
  else cout << "\n";
}
```

ВСТАВКА ЭЛЕМЕНТА в ДВУНАПРАВЛЕННЫЙ СПИСОК



ВСТАВКА ЭЛЕМЕНТА В ДВУНАПРАВЛЕННЫЙ СПИСОК

```
//вставка элемента с заданным номером в двунаправленный список
Double_List* Insert_Item_Double_List(Double_List* Head,
     int Number, int DataItem){
  Number--;
  Double_List *NewItem=new(Double_List);
  NewItem->Data=DataItem;
  NewItem->Prior=NULL;
  NewItem->Next = NULL;
  if (Head == NULL) {//список пуст
    Head = NewItem;
  }
```

ВСТАВКА ЭЛЕМЕНТА В ДВУНАПРАВЛЕННЫЙ СПИСОК

Продолжение 39

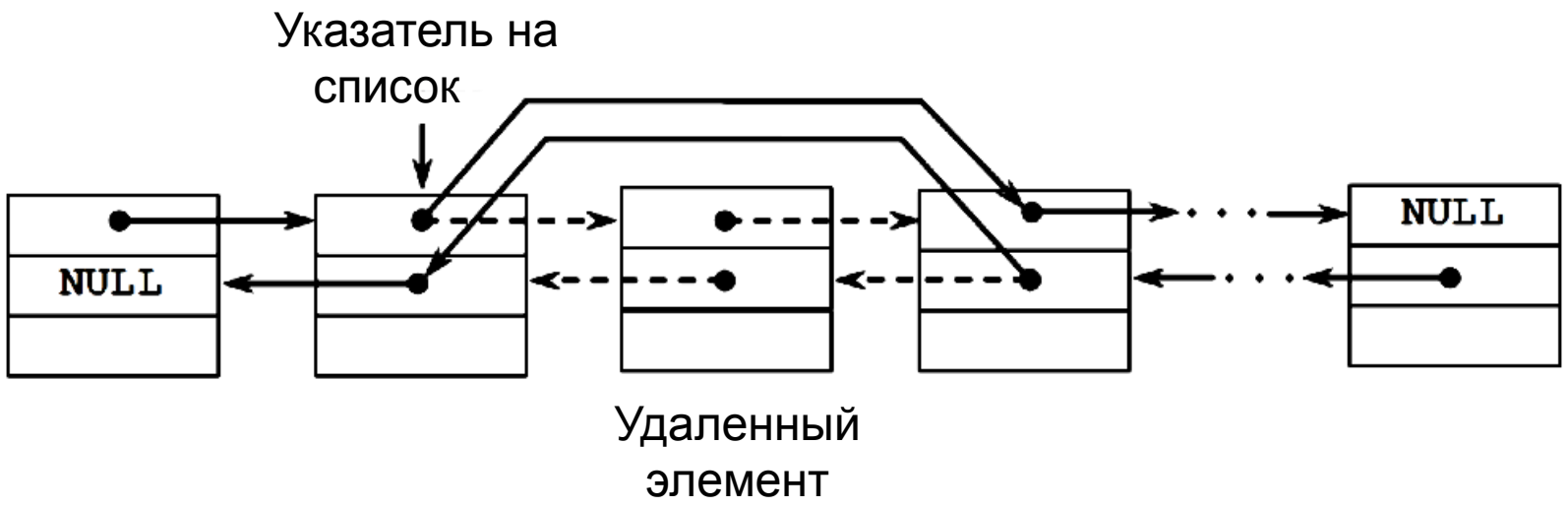
```
 else {//вставляем новый элемент на непервое место
      if (Current->Next != NULL) Current->Next->Prior =
NewItem;
      NewItem->Next = Current->Next;
      Current->Next = NewItem;
            NewItem->Prior = Current;
      Current = NewItem;
    }
  }
  return Head;
}
```

ВСТАВКА ЭЛЕМЕНТА в ДВУНАПРАВЛЕННЫЙ СПИСОК

Продолжение 40

```
else {//список не пуст
    Double_List *Current=Head;
    for(int i=1; i < Number && Current->Next!=NULL; i++)
    Current=Current->Next;
    if (Number == 0){
    //вставляем новый элемент на первое место
      NewItem->Next = Head;
      Head->Prior = NewItem;
      Head = NewItem;
    }
```

УДАЛЕНИЕ ЭЛЕМЕНТА из ДВУНАПРАВЛЕННОГО СПИСКА



УДАЛЕНИЕ ЭЛЕМЕНТА из ДВУНАПРАВЛЕННОГО СПИСКА

```
/*удаление элемента с заданным номером из двунаправленного списка*/
Double_List* Delete_Item_Double_List(Double_List* Head,
      int Number){
  Double_List *ptr;//вспомогательный указатель
  Double_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current->Prior == NULL){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Head->Prior = NULL;
      Current = Head;
    }
```

УДАЛЕНИЕ ЭЛЕМЕНТА из ДВУНАПРАВЛЕННОГО СПИСКА

```
else {//удаляем непервый элемент
    if (Current->Next == NULL) {
    //удаляем последний элемент
      Current->Prior->Next = NULL;
      delete(Current);
      Current = Head;
    }
```

УДАЛЕНИЕ ЭЛЕМЕНТА из ДВУНАПРАВЛЕННОГО СПИСКА

```
else {//удаляем непервый и непоследний элемент
        ptr = Current->Next;
        Current->Prior->Next =Current->Next;
        Current->Next->Prior =Current->Prior;
        delete(Current);
        Current = ptr;
      }
    }
  }
  return Head;
}
```

ПОИСК ЭЛЕМЕНТА в ДВУНАПРАВЛЕННОМ СПИСКЕ МОЖНО ВЕСТИ:

- ✓ **просматривая элементы от начала к концу списка;**
- ✓ **просматривая элементы от конца списка к началу;**
- ✓ **просматривая список в обоих направлениях одновременно: от начала к середине списка и от конца к середине (учитывая, что элементов в списке может быть четное или нечетное количество).**

ПОИСК ЭЛЕМЕНТА

```
//поиск элемента в двунаправленном списке
bool Find_Item_Double_List(Double_List* Head,
    int DataItem){
  Double_List *ptr; //вспомогательный указатель
  ptr = Head;
  while (ptr != NULL){//пока не конец списка
    if (DataItem == ptr->Data) return true;
    else ptr = ptr->Next;
    }
  return false;
}
```

ПРОВЕРКА ПУСТОТЫ
ДВУНАПРАВЛЕННОГО СПИСКА

```
//проверка пустоты двунаправленого списка
bool Empty_Double_List(Double_List* Head){
  if (Head!=NULL)  return false;
  else  return true;
}
```

УДАЛЕНИЕ
ДВУНАПРАВЛЕННОГО СПИСКА

```
//освобождение памяти, выделенной под двунаправленный список
void Delete_Double_List(Double_List* Head){
  if (Head != NULL){
    Delete_Double_List(Head->Next);
    delete Head;
  }
}
```

КЛЮЧЕВЫЕ ТЕРМИНЫ

- **Адрес сегмента – это одно из машинных слов, составляющих адрес динамического элемента, которое представляет собой адрес первого элемента структуры.**
- **Адресное поле (поле связок) – это поле структуры, в котором содержится указатель, связывающий данный элемент с другими элементами структуры.**
- **Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается не на этапе компиляции, а в процессе работы программы.**
- **Динамический элемент – это элемент динамической структуры, который в конкретный момент выполнения программы может либо существовать, либо отсутствовать в памяти.**

КЛЮЧЕВЫЕ ТЕРМИНЫ

- **Динамическое распределение памяти – это выделение памяти под отдельные элементы в тот момент, когда они "начинают существовать" в процессе выполнения программы.**
- **Информационное поле (поле данных) – это поле структуры, в котором содержатся непосредственно обрабатываемые данные.**
- **Связное представление данных – это установление связи между элементами динамической структуры через указатели.**
- **Смещение – это одно из машинных слов, составляющих адрес динамического элемента, которое представляет собой изменение адреса относительно первого элемента структуры.**

КЛЮЧЕВЫЕ ТЕРМИНЫ

- **Однонаправленный (односвязный) список – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка.**
- **Пустой список – это список нулевой длины.**
- **Связанный список – это структура, элементами которой служат записи одного формата, связанные друг с другом с помощью указателей, хранящихся в самих элементах.**

КЛЮЧЕВЫЕ ТЕРМИНЫ

- **Двунаправленный (двусвязный) список – это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы.**
- **Длина списка – это величина, равная числу элементов в списке.**
- **Линейный список – это список, отражающий отношения соседства между элементами.**

КЛЮЧЕВЫЕ ТЕРМИНЫ

- **Список – это упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения.**
- **Указатель начала списка (голова списка) – это указатель на первый элемент списка.**

КРАТКИЕ ИТОГИ

- ✓ В программах возникает необходимость обрабатывать данные, размер которых заранее неизвестен.
- ✓ Для данных с достаточно большим или переменным размером используются динамические структуры.
- ✓ Динамические структуры не имеют имени, под них выделяется память в процессе выполнения программы, количество их элементов может не фиксироваться, в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

КРАТКИЕ ИТОГИ

- ✓ Каждой динамической структуре ставится в соответствие статическая переменная – ее адрес.
- ✓ Представление динамических структур в памяти определяется как связное.
- ✓ Связное представление данных в программах имеет как достоинства, так и недостатки.
- ✓ Существует классификация динамических структур данных в зависимости от связей между элементами и допустимых операций.

КРАТКИЕ ИТОГИ

- ✓ **Элемент динамической структуры состоит как минимум из двух полей: адресного и информационного.**
- ✓ **Адресное поле формируется из двух слов: адрес сегмента и смещение.**
- ✓ **Доступ к данным в динамических структурах осуществляется с помощью операции косвенного выбора.**

КРАТКИЕ ИТОГИ

- ✓ **Список является динамической структурой, для элементов которого определены операции включения, исключения.**
- ✓ **В связанном списке элементы линейно упорядочены указателями, входящими в состав элементов списка.**
- ✓ **Линейные связные списки являются простейшими динамическими структурами данных и в зависимости от организации связей делятся на однонаправленные и двунаправленные.**
- ✓ **В однонаправленном (односвязном) списке каждый из элементов содержит информационную часть и указатель на следующий элемент списка. Адресное поле последнего элемента имеет значение NULL.**

КРАТКИЕ ИТОГИ

- ✓ **Каждый элемент списка содержит ключ, который идентифицирует этот элемент.**
- ✓ **Основными операциями с однонаправленными списками, являются: создание списка; печать (просмотр) списка; вставка элемента в список; удаление элемента из списка; поиск элемента в списке; проверка пустоты списка; удаление списка.**
- ✓ **В двунаправленном (двусвязном) списке каждый из элементов содержит информационную часть и два указателя на соседние элементы.**
- ✓ **Основные операции, выполняемые над двунаправленным списком, те же, что и для однонаправленного списка.**