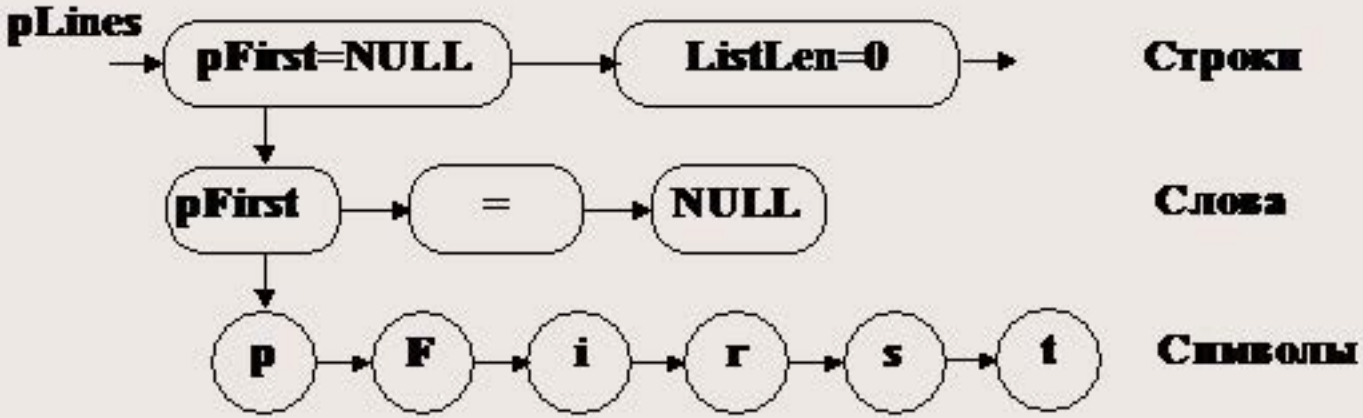
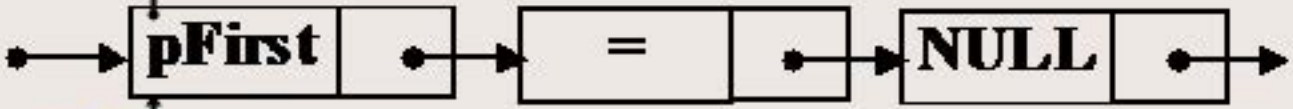
# Математическая модель текста – иерархическая структура представления (*дерево*)



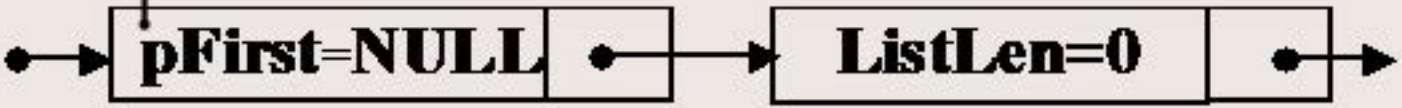
## 1. Уровень символов – список символов


## 2. Уровень слов – список слов




☑ В первом поле звеньев вместо значения-слов можно поместить указатель на соответствующий список символов

## 3. Уровень строк – список строк



☑ На всех уровнях представления (кроме символов) значение задается указателем на соответствующую структуру ниже расположенного уровня

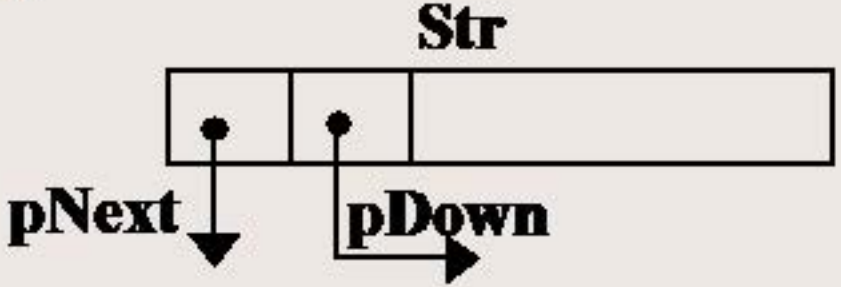
**Определение 2.1**. Разработанная структура хранения называется *связным (иерархическим) списком*

- Абстрактная структура типа дерева представима в виде связного списка
- В списке существуют делимые и неделимые (*атомарные, терминальные*) элементы (А-не, ТОМ-часть)

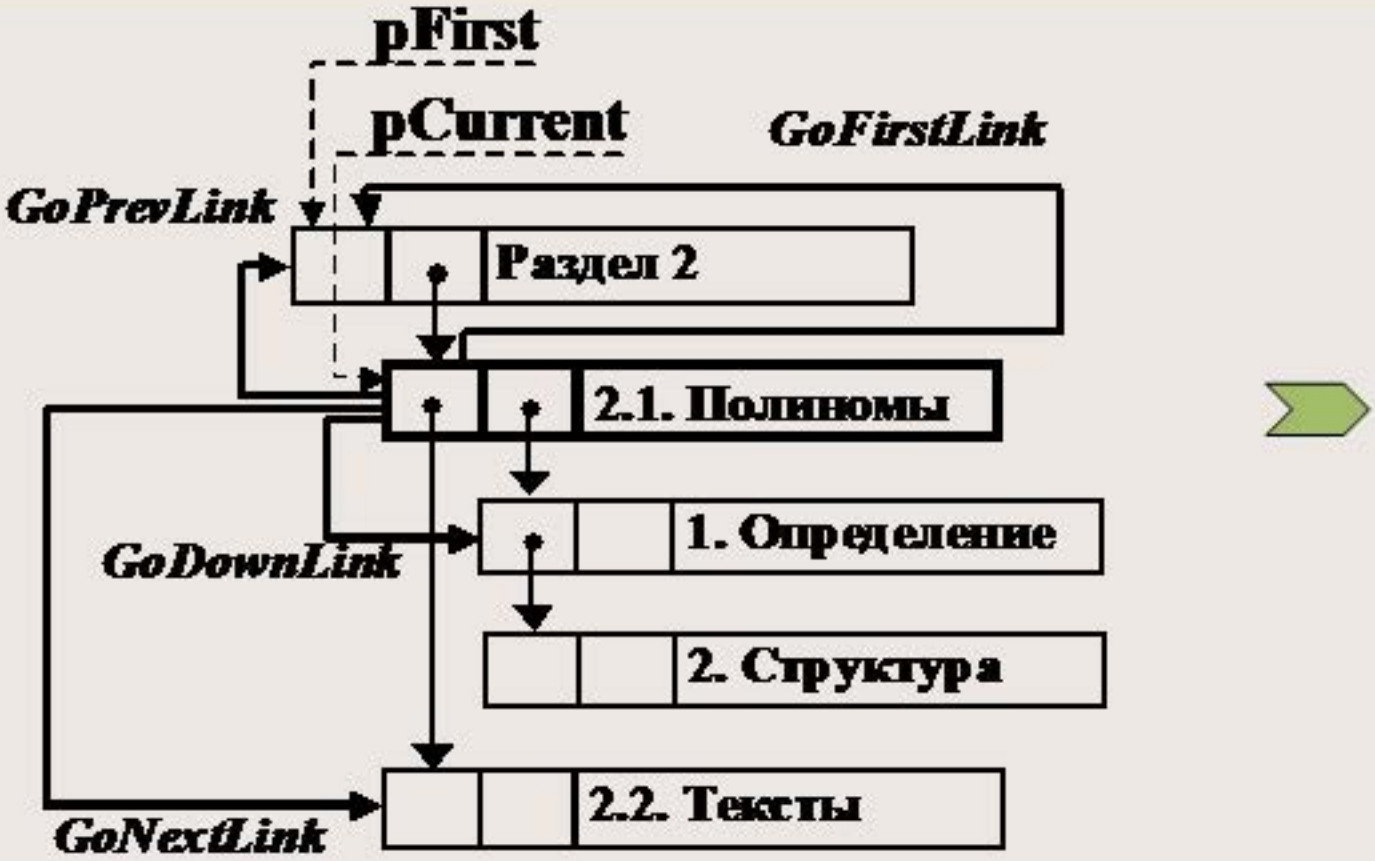
☑ Визуальное представление текста содержит только атомарные элементы, структура хранения должна включать все элементы

## Структура звена

```
#define TextLineLength  20
typedef char TStr[TextLineLength];
class TTextLink : public TDatValue {
  protected:
    TStr Str;
    TTextLink *pNext, *pDown;
};
```

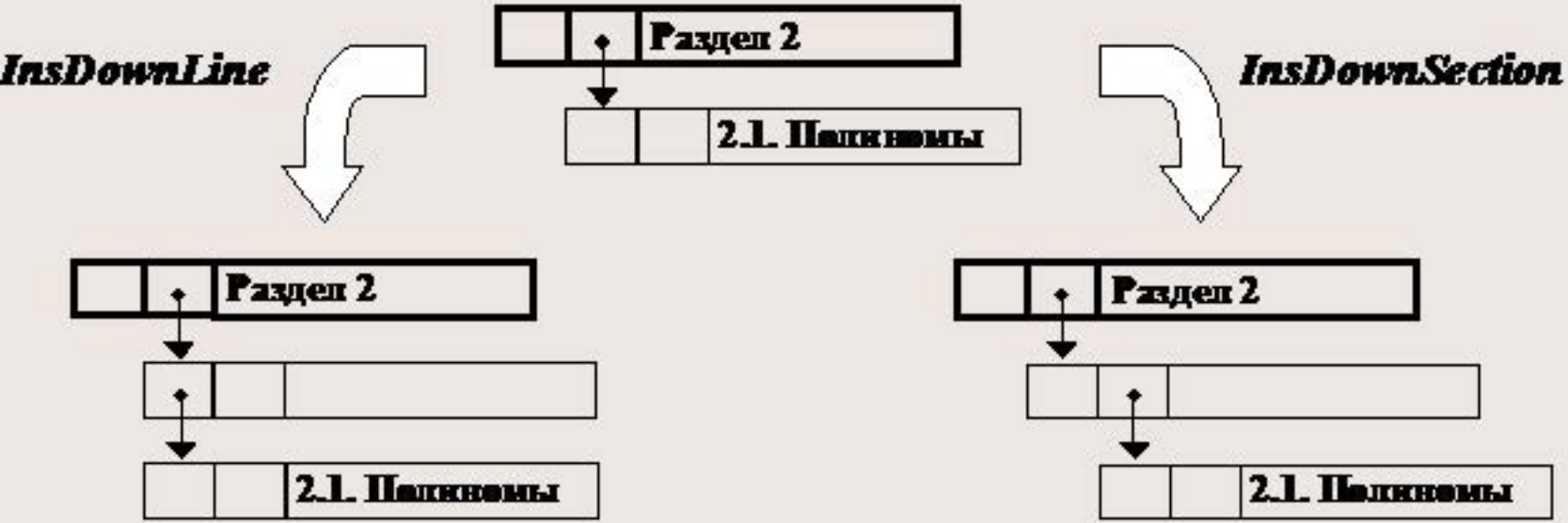


# Пример структуры хранения

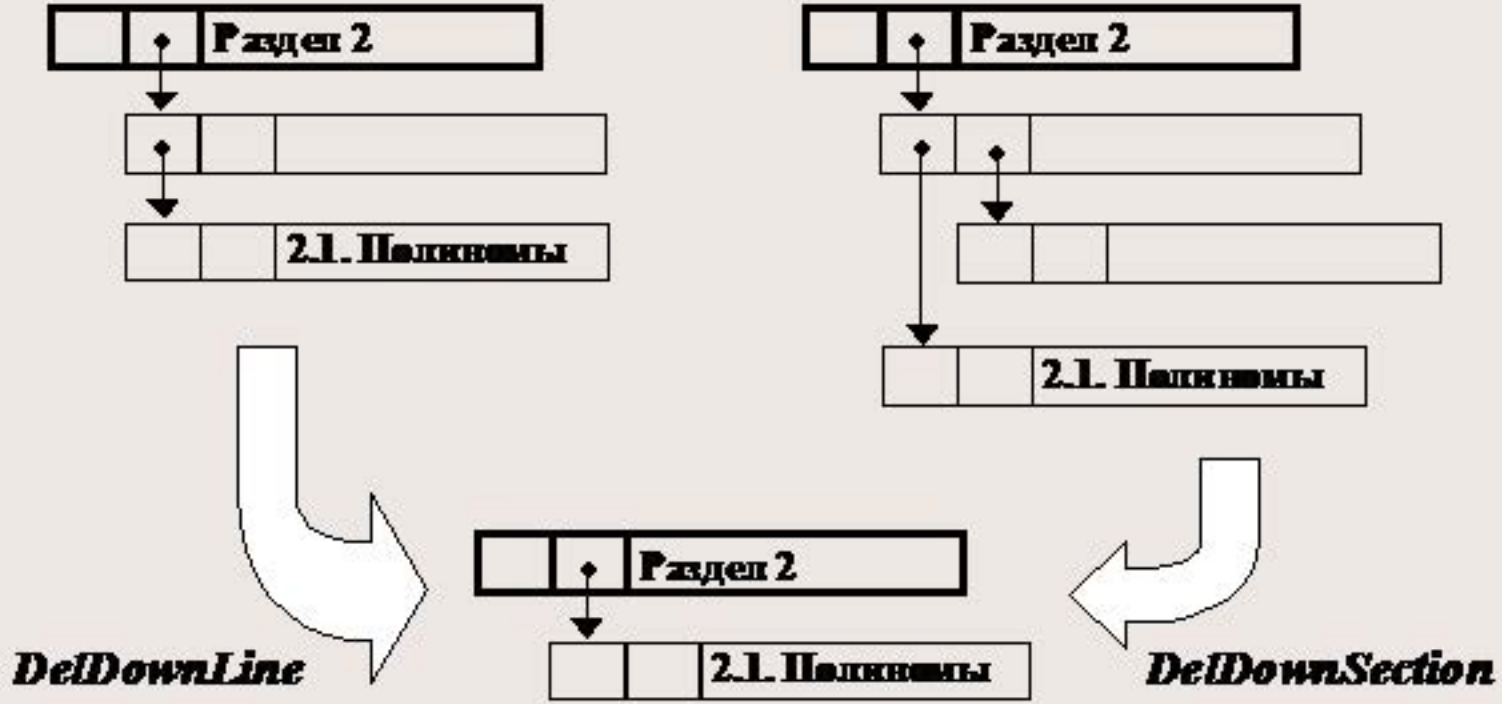



☑ В стеке Path размечаются указатели на все звенья, лежащие на пути от начала текста до текущего звена

# Вставка строки и раздела в подуровне



# Удаление строки и раздела в подуровне




- ☑ Операция удаление строки не выполняется, если исключаемый элемент не является атомарным

**Печать текста**: схема обхода – текст текущей строки, текст подуровня, текст следующего раздела текста того же уровня (top-down-next)

```
while (1) {
 if ( pLink != NULL ) {
    cout << pLink->Str;   // обработка звена
    St.push(pLink);       // запись в стек
    pLink = pLink->pDown;// переход на подуровень
 }
 else if ( St.empty() ) break;
 else {
    pLink = St.top(); St.pop();// выборка из стека
    pLink = pLink->pNext; // переход по тому же
 }                        // уровню
}
```

**Ввод текста из файла**: уровень текста в файле можно выделить строками специального вида (например, скобками '{' и '}')

Глава 2
{
2.1. Полиномы
{
1. Определение
2. Структура
{
2.2. Тексты
{
1. Определение
2. Структура
{
{

## Общая схема алгоритма

Повторить

- ввод строки
- ЕСЛИ '}' ТО Завершить
- ЕСЛИ '{' ТО Выполнить рекурсивно Ввод_текста
- Добавить строку на том же уровне

## Переход к следующему звену текста (GoNext)

- Получить звено из стека
- ЕСЛИ звено не является корнем всего текста ТО поместить следующие звенья (по указателям pNext и pDown) в стек

```
pCurrent = St.top(); St.pop();
if (pCurrent != pFirst ) {
   if ( pCurrent->pNext != NULL )
     St.push(pCurrent->pNext);
   if ( pCurrent->pDown != NULL )
     St.push(pCurrent->pDown);
}
```

## Общая схема алгоритма

- ☑ Для навигации по исходному тексту и тексту-копии используется один – объединенный - стек
- ☑ Каждое звено текста копируется за два прохода:

**1 проход** – при подъеме из подуровня (pDown)

- создание копии звена
- заполнение поля pDown (подуровень уже скопирован)
- запись в поле Str значения Copy (для распознания звена при попадании на него при втором проходе)
- запись в поле pNext указателя на звено-оригинал (для возможности последующего копирования текста исходной строки)
- запись указателя на звено-копию в стек

**2 проход** – при извлечении звена из стека

- заполнение полей Str и pNext
- указатель на звено-копию запоминается в переменной cpl

4. Повторное использование памяти (сборка мусора)…

**Общая замечания**

☑ При удалении разделов текста для освобождения звеньев следует учитывать следующие моменты:

– обход всех звеньев удаляемого текста может потребовать длительного времени,

– при множественности ссылок на разделы текста (для устранения дублирования одинаковых частей) удаляемый текст нельзя исключить – этот текст может быть задействован в других фрагментах текста

↳ Память, занимаемая удаляемым текстом, не освобождается, а удаление текста фиксируется установкой указателей в состояние NULL (например, pFirst=NULL)

☑Подобный способ выполнения операций удаления текста может привести к ситуации, когда в памяти, используемой для хранения текста, могут присутствовать звенья, на которые нет ссылок в тексте и которые не возвращены в систему управления памятью для повторного использования. Элементы памяти такого вида носят наименование "*мусора*"

☑Наличие "мусора" в системе может быть допустимым, если имеющейся свободой памяти достаточно для работы программ. В случае нехватки памяти необходимо выполнить "*сборку мусора*" (*garbage collection*)

## Общая схема подхода…

- ☑ Для системы управления память выделяется полностью при начале работы программы; вся память форматируется и представляется в виде линейного списка свободных звеньев
- ☑ Для фиксации состояния памяти в классе TTextLink создается статическая переменная MemHeader типа TTextMem

```
class TTextMem {
  PTTextLink pFirst; // первое звено
  PTTextLink pLast;  // последнее звено
  PTTextLink pFree;  // первое свободное
};
```

## Общая схема подхода…

☑ Для выделения и форматирования памяти определяется статический метод **InitMemSystem** класса TTextLink

```
void InitMemSystem (int size) {
 char *p = new char[sizeof(TTextLink)*size];
 MemHeader.pFirst = (PTTextLink) p;
 MemHeader.pFree  = MemHeader.pFirst;
 MemHeader.pLast  = MemHeader.pFirst + (size-1);
 PTTextLink pLink = MemHeader.pFirst;
 for ( int i=0; i<size-1; i++, pLink++ ) {
   pLink->pNext = pLink+1;
 }
 pLink->pNext = NULL;
}
```

## Общая схема подхода…

☑ При запросе памяти в операторе **new** выделяется первое свободное звено

```
// выделение звена
void * operator new (size_t size) {
  PTTextLink pLink = MemHeader.pFree;
  if ( MemHeader.pFree != NULL )
    MemHeader.pFree = pLink->pNext;
  return pLink;
}
```

## Общая схема подхода…

☑ При освобождение звена в операторе **delete** звено включается в список свободных звеньев

```
// освобождение звена
void operator delete (void *pM) {
  PTTextLink pLink = (PTTextLink)pM;
  pLink->pNext = MemHeader.pFree;
  MemHeader.pFree = pLink;
}
```

# Заключение

- Характер использования текста определяет способ его представления
- Для хранения иерархически-представленного текста могут быть использованы связные списки
- Связные списки является общей структурой хранения структур типа дерева
- Использование итераторов позволяет обеспечить единый и простой способ обработки различных структур данных
- При разработке структур хранения сложных данных может потребоваться создание дополнительных средств управления памятью
- Возможный подход управления памятью – накопление и сборка мусора