


Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Учебный курс

Методы программирования - 2

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Учебный курс:

Методы программирования - 2

Тема 2.2:

Редактирование текстов

Гергель В.П., профессор ,
директор института ИТММ

Содержание

Глава 2. ***Динамические структуры и представление на ЭВМ сложных математических моделей***

2.2. **Редактирование текстов**

1. Выбор модели представления текста
2. Выбор структуры хранения текста
3. Реализация
   - схема наследования,
   - навигация,
   - доступ,
   - модификация структуры,
   - алгоритмы обхода,
   - итератор,
   -копирование
4. Повторное использование памяти (сборка мусора)

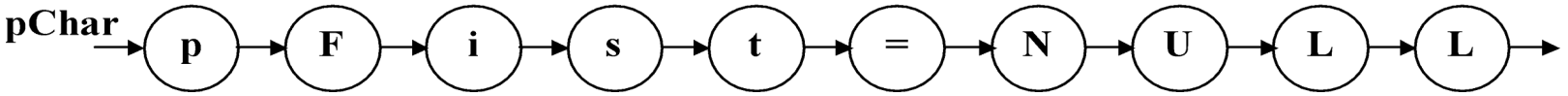
Вопросы для обсуждения

1. Выбор модели представления текста …

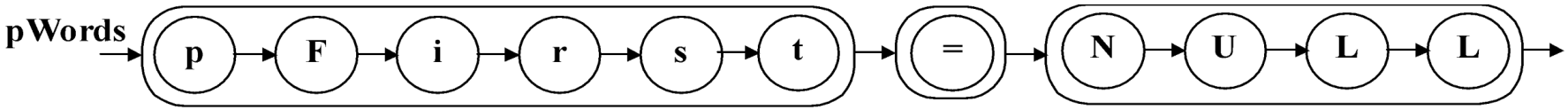
Пример:

```
pFirst  = NULL;
ListLen = 0;
```

1. Текст – линейная последовательность символов



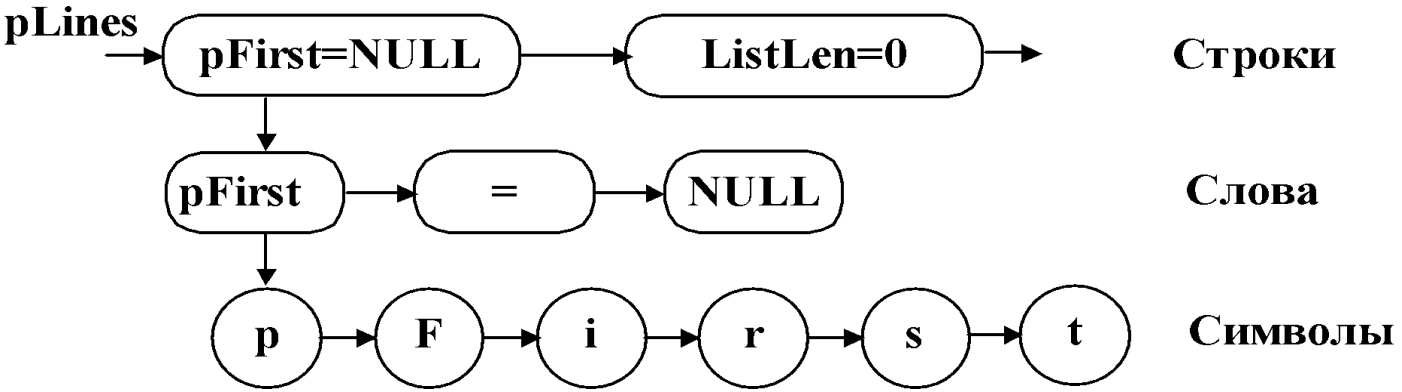
2. Текст – линейная последовательность слов (слово - линейная последовательность символов)



3. Текст – линейная последовательность строк, строки состоят из слов, слова – из символов и т.д.

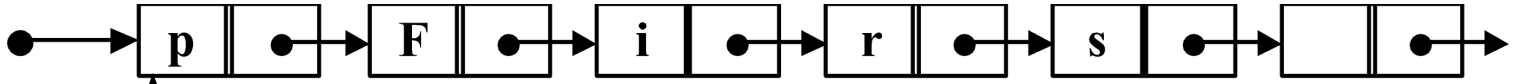
1. Выбор модели представления текста

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

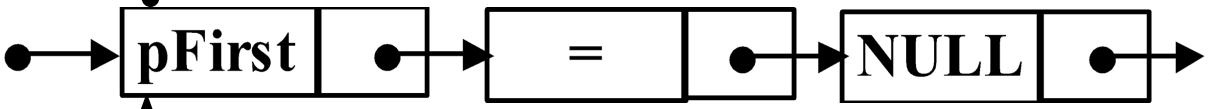


2. Выбор структуры хранения текста …

1. Уровень символов – список символов

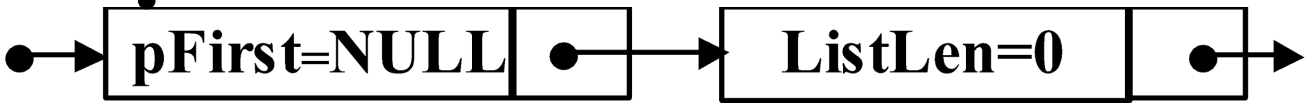


2. Уровень слов – список слов



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

3. Уровень строк – список строк



2. Выбор структуры хранения текста …

- 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

2. Выбор структуры хранения текста …

- Разные типы звеньев – трудности при управлении памятью, дублирование программ обработки

Единый тип звена

```
typedef Tlink *PTLink;
class TLink{
  PTLink pNext;
  int Atom; // =1 - звено-атом
  union {
    PTLink pDown;
    char   Symb;
};
```

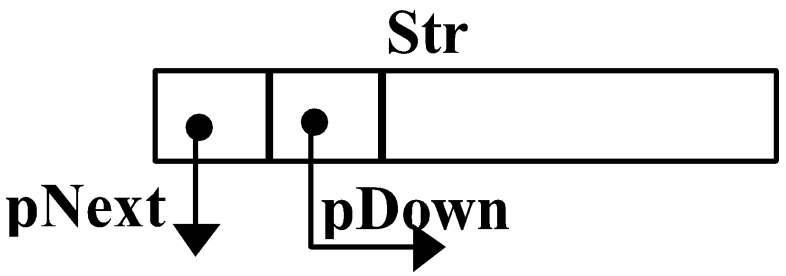
2. Выбор структуры хранения текста …

- Размер введенного унифицированного звена – 10 байт (=16 при учете округления при динамическом выделении памяти) ⇒ для представления символов такой вид звена является неэкономичным
- Возможное решение проблемы – повышение уровня атомарности
- Целесообразный уровень – **уровень строк**

2. Выбор структуры хранения текста …

Структура звена

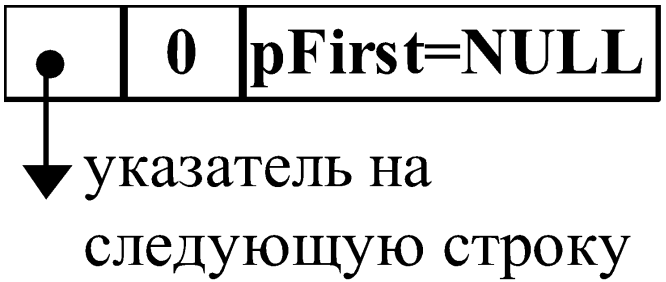
```
#define TextLineLength  20
typedef char TStr[TextLineLength];
class TTextLink : public TDatValue {
  protected:
    TStr Str;
    TTextLink *pNext, *pDown;
};
```



2. Выбор структуры хранения текста …

- Указатель **pNext** есть ссылка на следующий элемент того же уровня
- Указатель **pDown** есть ссылка на структуру хранения ниже расположенного уровня

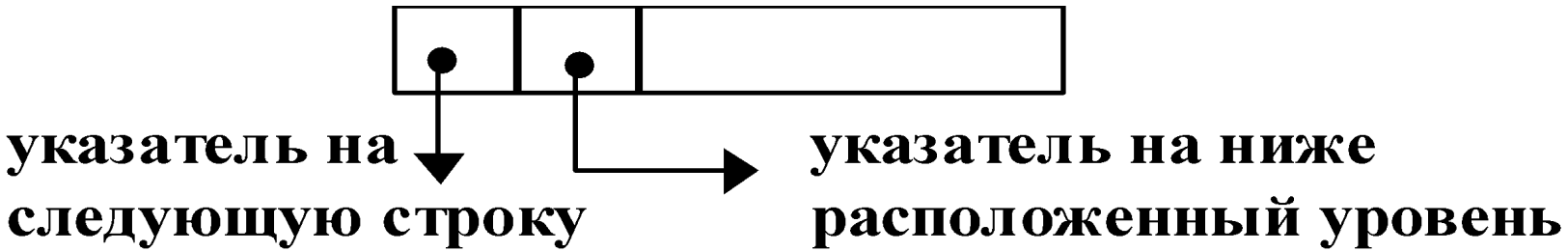
Представление строки (атомарного элемента)



- Признак атомарности элемента **pDown==NULL**

2. Выбор структуры хранения текста …

Представление структурного элемента



- Для ссылки на ниже расположенный уровень используется указатель **pDown**
- Поле **Str** можно использовать для именования структурных элементов текста (название всего текста, его разделов, подразделов и т.д.)

2. Выбор структуры хранения текста …

Пример структуры хранения

2. Выбор структуры хранения текста

Базовые операции обработки звена

- Порождение звена (конструктор)

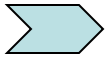
```
TTextLink ( TStr s=NULL,
            PTTextLink pn=NULL,
            PTTextLink pd=NULL );
```

- Переход к следующей звену

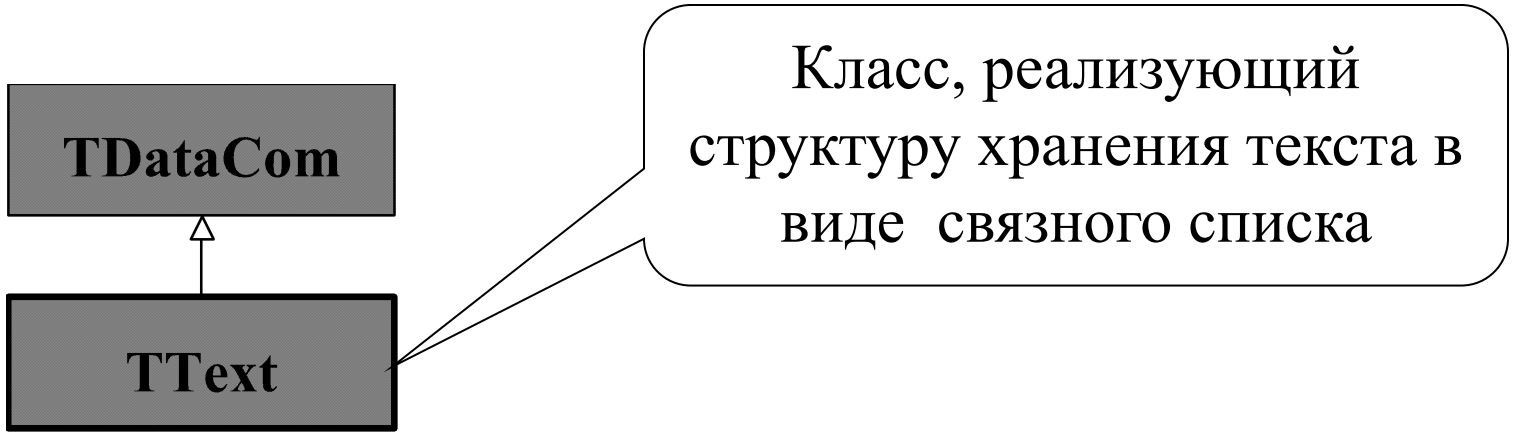
```
PTTextLink  GetNext() { return pNext; }
```

- Переход на подуровень

```
PTTextLink  GetDown() { return pDown; }
```



3. Реализация – схема наследования …

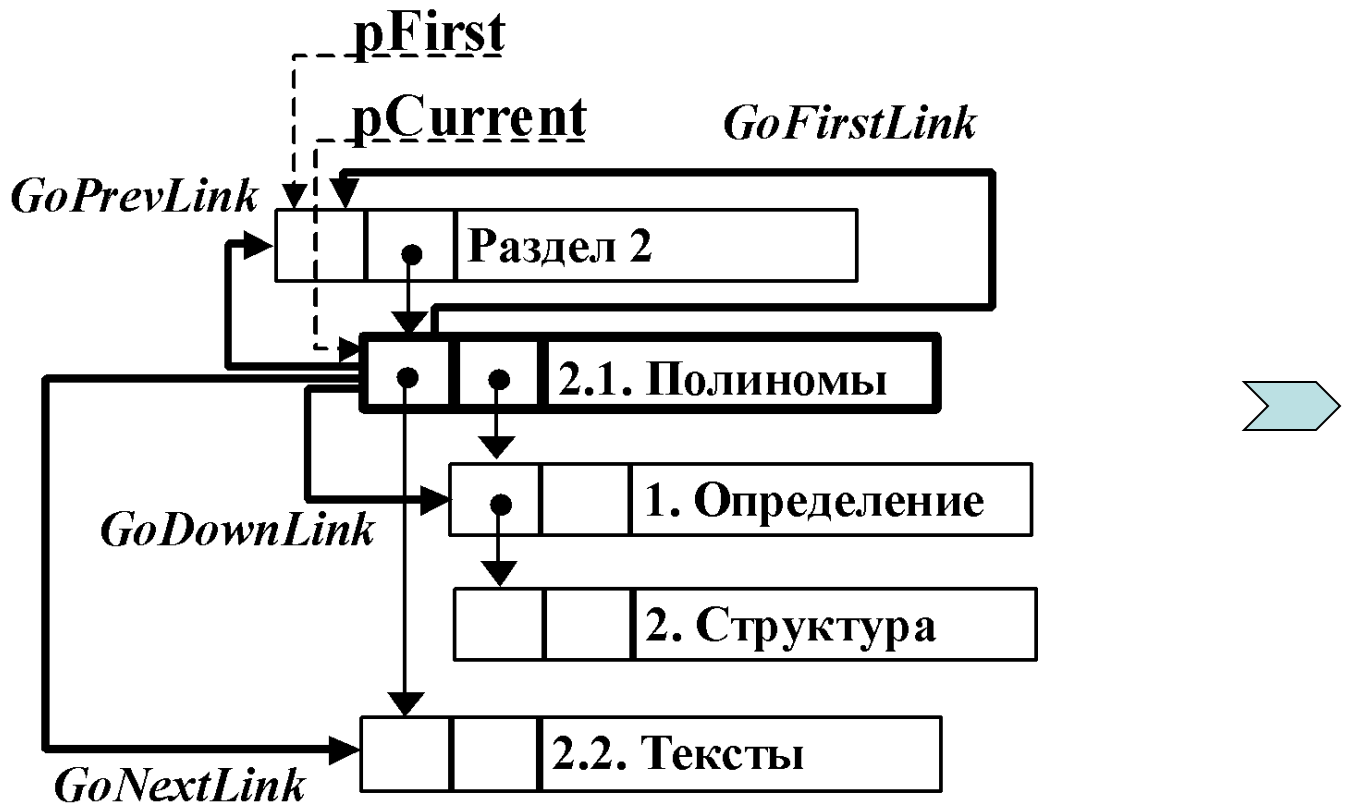


3. Реализация – навигация по структуре …

```
PTTextLink pFirst;       // корень дерева
PTTextLink pCurrent;     // текущая строка
stack<PTTextLink> Path; // стек траектории
// навигация
int GoFirstLink(void); // к первой строке
int GoDownLink (void); // к след. строке по Down
int GoNextLink (void); // к след. строке по Next
int GoPrevLink (void); // к пред. позиции
```

- Указатель текущей строки pCurrent не включается в стек траектории движения по тексту

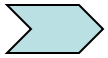
3. Реализация – навигация по структуре



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

3. Реализация – доступ к текущей строке

```
// доступ
string GetLine  (void); // чтение текста текущего звена
void SetLine(string s); // замена текста текущего звена
```

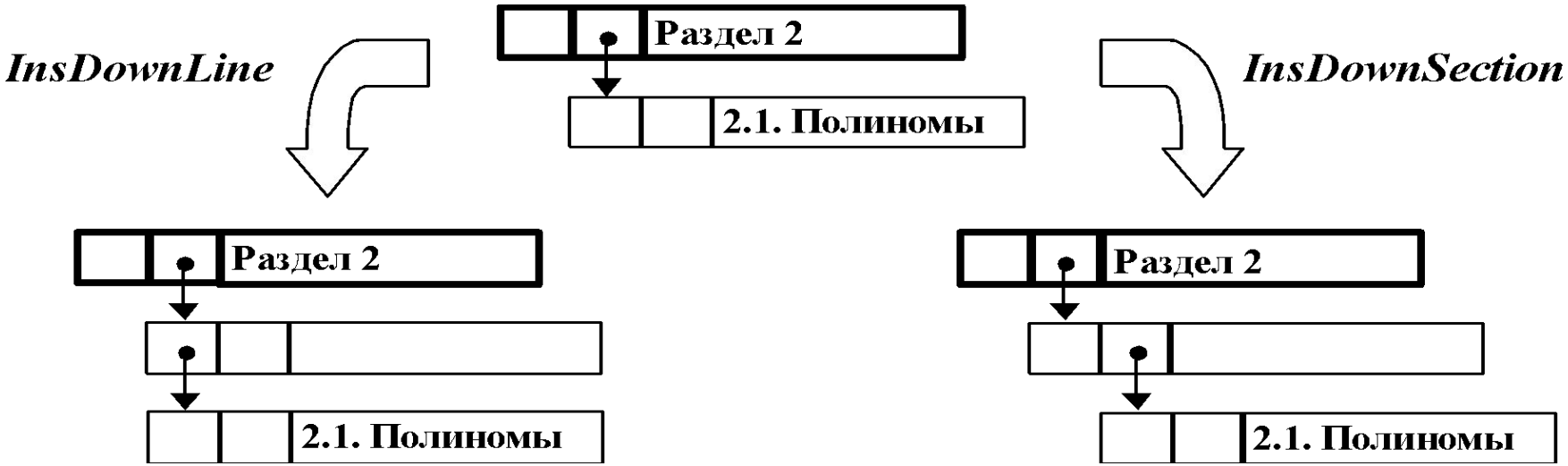


3. Реализация – модификация структуры …

```
// вставка и удаление строки в подуровне
void InsDownLine(string s);
void DelDownLine(void);
// вставка и удаление раздела в подуровне
void InsDownSection(string s);
void DelDownSection(void);
// вставка и удаление строки на том же уровне
void InsNextLine(string s);
void DelNextLine(void);
// вставка и удаление раздела на том же уровне
void InsNextSection(string s);
void DelNextSection(void);
```

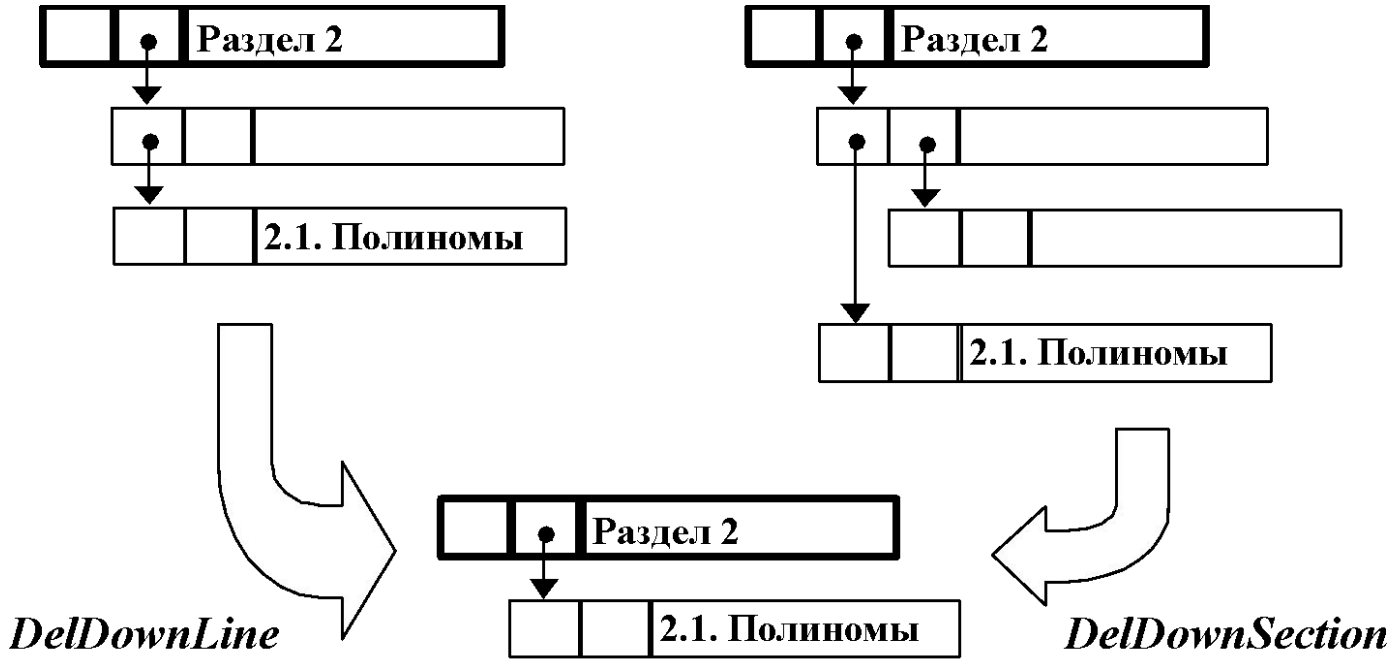
3. Реализация – модификация структуры …

Вставка строки и раздела в подуровне

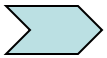


3. Реализация – модификация структуры

Удаление строки и раздела в подуровне



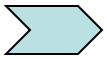
- Операция удаление строки не выполняется, если исключаемый элемент не является атомарным



3. Реализация – алгоритмы обхода …

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
}
```

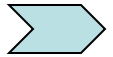


3. Реализация – алгоритмы обхода

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

3. Реализация – итератор …

- Схема обхода TDN, нерекурсивный вариант
- Корневые звенья необработанных разделов текста запоминаются в стеке
- Текущая строка в стеке не хранится (кроме корневого звена всего текста)

3. Реализация – итератор …

Инициализация (Reset) – установка на корневое звено текста

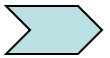
```
pCurrent = pFirst;
if ( pCurrent != NULL ) {
  St.push(pCurrent);
  if ( pCurrent->pNext != NULL )
  St.push(pCurrent->pNext);
  if ( pCurrent->pDown != NULL )
  St.push(pCurrent->pDown);
}
```

3. Реализация – итератор

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);
}
```



Пример: программа

3. Реализация – копирование текста …

Общая замечания

- Для копирования текста необходимо предварительно скопировать разделы текста, на которые указывают указатели pDown и pNext ⇒ алгоритмы обхода NDT или DNT
- Для навигации по тексту-копии также необходим стек; использование стеков исходного текста и текста-копии должны быть согласованы

3. Реализация – копирование текста …

Общая схема алгоритма

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

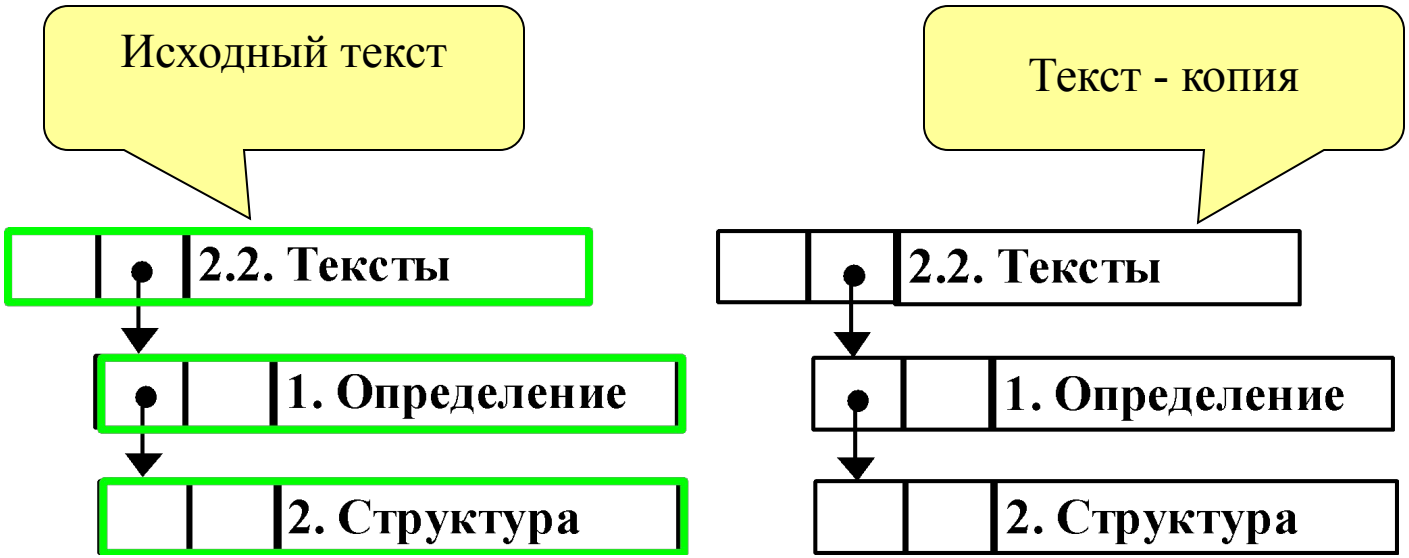
1 проход – при подъеме из подуровня (pDown)

- 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

3. Реализация – копирование текста



Пример: программа

3. Реализация – 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)

3. Реализация – повторное использование памяти (сборка мусора) …

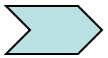
- Подобный способ выполнения операций удаления текста может привести к ситуации, когда в памяти, используемой для хранения текста, могут присутствовать звенья, на которые нет ссылок в тексте и которые не возвращены в систему управления памятью для повторного использования. Элементы памяти такого вида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

3. Реализация – повторное использование памяти (сборка мусора) …

Общая схема подхода…

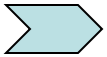
- При освобождение звена в операторе **delete** звено включается в список свободных звеньев

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



3. Реализация – повторное использование памяти (сборка мусора)

- Как при сканировании памяти различать звенья "мусора" и звенья текста ?
- Возможный подход – маркировка текстовых звеньев и звеньев списка свободных звеньев



Пример: программа

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

Вопросы для обсуждения

- Дополнительные модели представления текста
- Набор операций при работе со связными списками
- Рекурсивные и итеративные алгоритмы обработки данных
- Статические данные и статические методы классов
- Перегрузка операторов new и delete
- Способы недопущения мусора при множественности ссылок на структурные элементы структуры хранения

Темы заданий для самостоятельной работы

- Расширение набора операций обработки текста (изменение структуры текста, копирование)
- Разработка визуальных средств работы с иерархическим текстом

Следующая тема

- Структуры хранения геометрических объектов на ЭВМ

Контакты

Нижегородский государственный университет им. Н.И. Лобачевского (www.unn.ru)

Институт информационных технологий, математики и механики (www.itmm.unn.ru)

603950, Нижний Новгород, пр. Гагарина, 23,
р.т.: (831) 462-33-56,

Гергель Виктор Павлович

(http://www.software.unn.ru/?dir=*17*)

E-mail: gergel@unn.ru