

## 1. Синтаксис команд

Синтаксис описывает структуру команд и их элементов:

- **<команда\_shell>** ::= <список\_команд>
- **<список\_команд>** ::= <конвейер> { [&, &&, ||, ;] <конвейер>} [&, ;]
- **<конвейер>** ::= <команда> { | <команда>}
- **<команда>** ::= <имя\_файла> {<аргумент>} [<имя\_файла>] [ [>, >>] <имя\_файла>]

## 2. Тестирование

### a) Корректные команды

1. **echo "Hello from file.in" > file.in**
  - Результат: Текст записан в файл **file.in**.
2. **cat < file.in > file.out**
  - Результат: Содержимое **file.in** записано в **file.out**.
3. **ls && pwd**
  - Выводит содержимое директории и текущий путь (если **ls** выполнена успешно).
4. **ls | cat >> my\_home.txt && pwd**
  - Добавляет список файлов в **my\_home.txt**, затем выводит путь.
5. **ls | pwd | wc**
  - Передаёт результат команды **pwd** на вход **wc**. Вывод: 1 1 17.
6. **cat f; date; pwd > zz**
  - Последовательно выводит файл **f**, текущую дату и записывает путь в файл **zz**.
7. **ls | cat | cat | cat | cat**
  - Многократная передача вывода **ls** через **cat**.

### б) Ошибочные команды

1. **wc < file1.in < file2.in**
  - Ошибка: Одновременно переданы два входных файла.
2. **ls | pwd || | wc**
  - Ошибка: Некорректное использование оператора **||**.
3. **ls & &**
  - Ошибка: Неверное использование **&**.
4. **sleep ;**

- Ошибка: Пропущен аргумент команды `sleep`.

### 3. Используемые структуры данных

#### 1. Список лексем (Node)

Для хранения лексем используется односвязный список:

```
typedef char* Data;
struct Node {
    Data data;           // Стока-лексема
    Node* next;          // Указатель на следующий элемент
};
```

#### Применение:

- Разделение строки на лексемы.
- Хранение команд, аргументов и спецсимволов.
- Замена переменных окружения (`$HOME`, `$USER`, `$EUID`).
- Проверка на синтаксические ошибки.

#### 2. Дерево команд (cmd\_inf)

Для представления синтаксиса используется дерево команд:

```
struct cmd_inf {
    int argc;           // Количество аргументов
    char **argv;        // Массив аргументов
    char *infile;       // Имя входного файла
    char *outfile;      // Имя выходного файла
    int append;         // Флаг "добавить в файл" (>>)
    int backgrnd;       // Флаг фонового выполнения
    struct cmd_inf *psubcmd; // Указатель на вложенную
команду
    struct cmd_inf *pipe; // Указатель на следующую
команду в конвейере
    struct cmd_inf *next; // Указатель на следующую
команду (после &&, ||, ;)
    enum type_of_next type; // Тип связи (;, &&, ||)
};
```

#### Применение:

- Хранение синтаксического дерева.
- Реализация конвейеров (`pipe`).
- Перенаправление ввода/вывода (`infile`, `outfile`, `append`).
- Условное выполнение команд (`&&`, `||`).
- Фоновый режим (`backgrnd`).

## Дополнительные структуры

### Перечисление `type_of_next`

Описывает тип связи между командами:

```
enum type_of_next {
    NXT, // Последовательное выполнение ( ; )
    AND, // Условное выполнение при успехе ( && )
    OR   // Условное выполнение при ошибке ( || )
};
```

### Перечисление ошибок

Определяет коды синтаксических ошибок:

```
enum {
    ERR_BRCKT = 1, // Ошибка в скобках
    ERR_QTN_MRK = 2, // Ошибка в кавычках
    ERR_SMCLN = 3 // Ошибка в точке с запятой
    // Другие ошибки...
};
```

## Отчёт по заданию 5

Автор: Исхаков Шамиль, группа 215

### 1. Синтаксис команд

Синтаксис описывает структуру команд и их элементов:

- `<команда_shell>` ::= `<список_команд>`
- `<список_команд>` ::= `<конвейер> { [&, &&, ||, ;] <конвейер>} [&, ;]`
- `<конвейер>` ::= `<команда> { | <команда>}`
- `<команда>` ::= `<имя_файла> {<аргумент>} [<имя_файла>] [>, >>] <имя_файла>`

### 2. Тестирование

#### a) Корректные команды

##### 1. `echo "Hello from file.in" > file.in`

- Результат: Текст записан в файл `file.in`.

## 2. **cat < file.in > file.out**

- Результат: Содержимое `file.in` записано в `file.out`.

## 3. **ls && pwd**

- Выводит содержимое директории и текущий путь (если `ls` выполнена успешно).

## 4. **ls | cat >> my\_home.txt && pwd**

- Добавляет список файлов в `my_home.txt`, затем выводит путь.

## 5. **ls | pwd | wc**

- Передаёт результат команды `pwd` на вход `wc`. Вывод: 1 1 17.

## 6. **cat f; date; pwd > zz**

- Последовательно выводит файл `f`, текущую дату и записывает путь в файл `zz`.

## 7. **ls | cat | cat | cat | cat**

- Многократная передача вывода `ls` через `cat`.

## 6) Ошибочные команды

### 1. **wc < file1.in < file2.in**

- Ошибка: Одновременно переданы два входных файла.

### 2. **ls | pwd || | wc**

- Ошибка: Некорректное использование оператора `||`.

### 3. **ls & &**

- Ошибка: Неверное использование `&`.

### 4. **sleep ;**

- Ошибка: Пропущен аргумент команды `sleep`.

## 3. Используемые структуры данных

### 1. Список лексем (Node)

Для хранения лексем используется односвязный список:

```
typedef char* Data;  
struct Node {  
    Data data;           // Стока-лексема  
    Node* next;         // Указатель на следующий элемент  
};
```

**Применение:**

- Разделение строки на лексемы.
- Хранение команд, аргументов и спецсимволов.

- Замена переменных окружения (\$HOME, \$USER, \$EUID).
- Проверка на синтаксические ошибки.

## 2. Дерево команд (cmd\_inf)

Для представления синтаксиса используется дерево команд:

```
struct cmd_inf {
    int argc;                      // Количество аргументов
    char **argv;                    // Массив аргументов
    char *infile;                  // Имя входного файла
    char *outfile;                 // Имя выходного файла
    int append;                     // Флаг "добавить в файл" (>>)
    int backgrnd;                  // Флаг фонового выполнения
    struct cmd_inf *psubcmd; // Указатель на вложенную
команду
    struct cmd_inf *pipe;          // Указатель на следующую
команду в конвейере
    struct cmd_inf *next;          // Указатель на следующую
команду (после &&, ||, ;)
    enum type_of_next type; // Тип связи (;, &&, ||)
};
```

**Применение:**

- Хранение синтаксического дерева.
- Реализация конвейеров (pipe).
- Перенаправление ввода/вывода (infile, outfile, append).
- Условное выполнение команд (&&, ||).
- Фоновый режим (backgrnd).

## Дополнительные структуры

### Перечисление type\_of\_next

Описывает тип связи между командами:

```
enum type_of_next {
    NXT, // Последовательное выполнение (;}
    AND, // Условное выполнение при успехе (&&)
    OR   // Условное выполнение при ошибке (||)
};
```

### Перечисление ошибок

Определяет коды синтаксических ошибок:

```
enum {
    ERR_BRCKT = 1, // Ошибка в скобках
```

```
ERR_QTN_MRK = 2, // Ошибка в кавычках
ERR_SMCLN = 3 // Ошибка в точке с запятой
// Другие ошибки...
};
```

#### 4. Построение списка слов и дерева команд



