1　命令预处理

1.1 命令传入

sed-command 可以通过命令行参数或读取文件获得。如果传入了文件路径，则读取。

```
def preprocess_script(scripts, is_file) -> List[str]:
    """ Preprocesses a script by removing comments and splitting it into a list of patterns.

    Args:
        scripts (str): The input script or file path.
        is_file (bool): Indicates whether the input is a file path.

    Returns:
        list: A list of patterns extracted from the script.
    """
    if is_file:
        with open(scripts, 'r') as f:
            scripts = f.read()
```

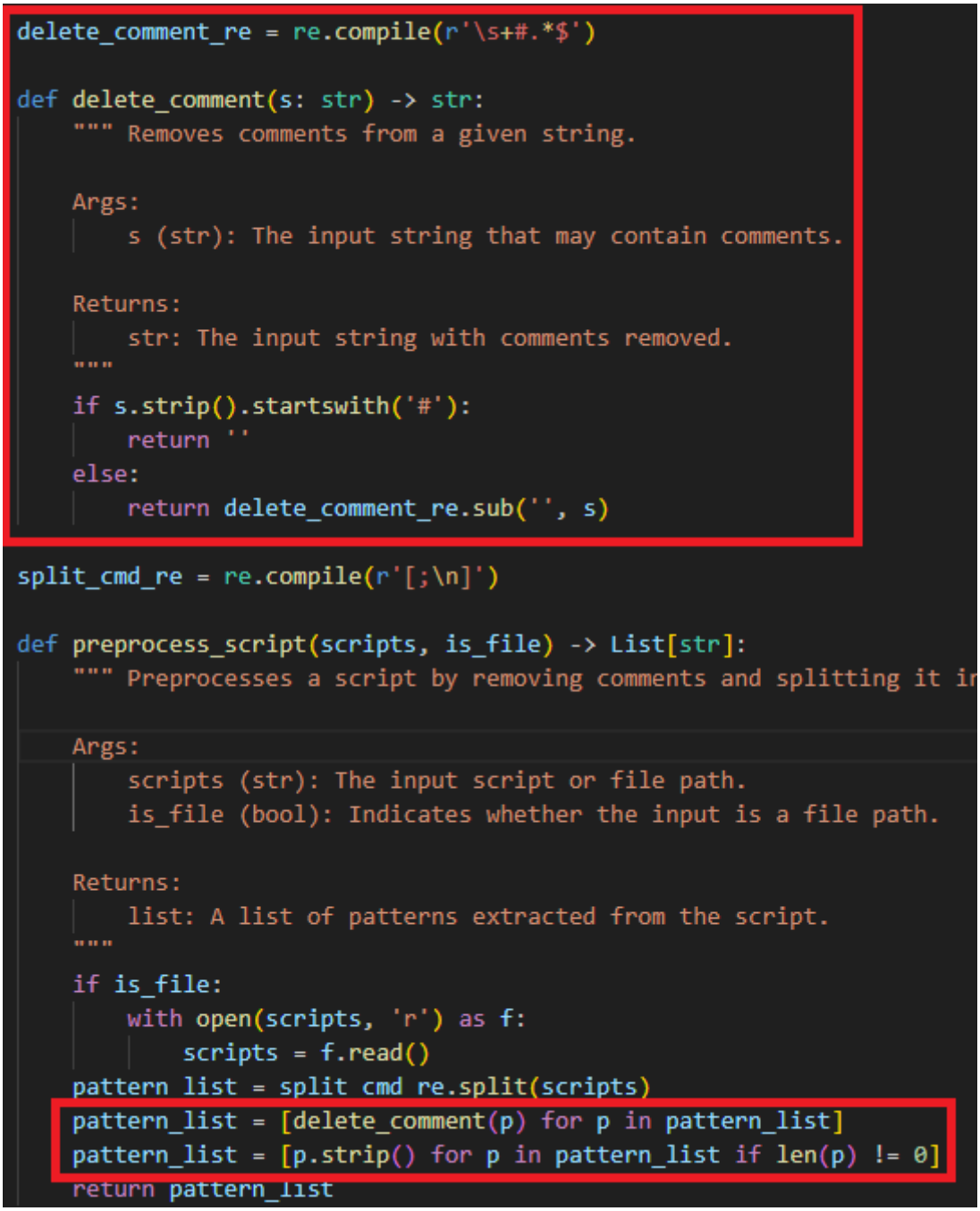
1.2 命令分割

传入的命令可能是单个，也可能是多个。如果是多个命令，是以换行或分号来间隔的，分割也就是换行符和分号进行。

```
split_cmd_re = re.compile(r'[;\n]')

def preprocess_script(scripts, is_file) -> List[str]:
    """ Preprocesses a script by removing comments and splitting it into a list of patterns.

    Args:
        scripts (str): The input script or file path.
        is_file (bool): Indicates whether the input is a file path.

    Returns:
        list: A list of patterns extracted from the script.
    """
    if is_file:
        with open(scripts, 'r') as f:
            scripts = f.read()
    pattern_list = split_cmd_re.split(scripts)                    # Split the script into a list of patterns
```

1.3 去除多余空格和注释

上一步完成了命令分割，现在对分割好的单个命令去除空格和注释处理。
如果存在一行只有注释，这种情况判断是不是“#”开头，是就直接返回空字符串。如果不是，那就存在命令和注释混合，使用正则表达式替换去除注释。
在完成注释处理以后，移除空字符串（仅含注释的），并去除命令字段首尾的空格。



```python
delete_comment_re = re.compile(r'\s+#.*$')

def delete_comment(s: str) -> str:
    """ Removes comments from a given string.

    Args:
        s (str): The input string that may contain comments.

    Returns:
        str: The input string with comments removed.
    """
    if s.strip().startswith('#'):
        return ''
    else:
        return delete_comment_re.sub('', s)

split_cmd_re = re.compile(r'[;\n]')

def preprocess_script(scripts, is_file) -> List[str]:
    """ Preprocesses a script by removing comments and splitting it ir

    Args:
        scripts (str): The input script or file path.
        is_file (bool): Indicates whether the input is a file path.

    Returns:
        list: A list of patterns extracted from the script.
    """
    if is_file:
        with open(scripts, 'r') as f:
            scripts = f.read()
    pattern_list = split_cmd_re.split(scripts)
    pattern_list = [delete_comment(p) for p in pattern_list]
    pattern_list = [p.strip() for p in pattern_list if len(p) != 0]
    return pattern_list
```

2 命令字段分割

实现的命令有 q、p、s、d。命令前方可能存在地址，也可能没有，共有 7 种情况。s 命令后存在替换规则字段，其它三个命令没有。

于是定义了一个结构来存储每个命令，classification 是命令类别，根据地址类型来分类的；command 用于存储纯命令，即 q、p、s 和 d 中的一种；address 用于存储地址；pattern 用于存储 s 命令的替换规则。

```python
class AddressCommandStruct:
    def __init__(self, classification, command, address=None, pattern=None):
        """ Initializes an instance of AddressCommandStruct.

        Args:
            classification (str): The classification of the address command.
            command (str): The command associated with the address.
            address (Optional): The address value (default: None).
            pattern (Optional): The pattern value (default: None).

        Attributes:
            classification (str): The classification of the address command.
            command (str): The command associated with the address.
            address: The address value.
            pattern: The pattern value.
        """
        self.classification: str = classification
        self.command: str = command
        self.address = address
        self.pattern = pattern
```

地址部分，可能无地址，指定行数 、正则表达式匹配、范围（行数范围、正则表达式范围以及行数和正则表达式混合的范围）。classification 具体分为为：无地址（None）、指定行数地址（line）、行数范围地址（line_line）、正则表达式（re）、正则表达式范围（re_re）、行数起始-正则表达式结束范围（line_re）、正则表达式起始-行数结束范围（re_line）。分别使用以下正则表达式匹配：

```python
address_command_patterns = [                                        # line: line number, re: regular expression
    re.compile(r'(\d+|\$)\s*,\s*(\d+|\$)\s*(.)\s*(.*?)$'),          # line_line
    re.compile(r'/\s*(.+?)\s*/\s*,\s*/\s*(.+?)\s*/\s*(.)\s*(.*?)$'),# re_re
    re.compile(r'(\d+|\$)\s*,\s*/\s*(.+?)\s*/\s*(.)\s*(.*?)$'),     # line_re
    re.compile(r'/\s*(.+?)\s*/\s*,\s*(\d+|\$)\s*(.)\s*(.*?)$'),     # re_line
    re.compile(r'/\s*(.+?)\s*/\s*(.)\s*(.*?)$'),                    # re
    re.compile(r'(\d+|\$)\s*(.)\s*(.*?)$'),                         # line
    re.compile(r'(.)\s*(.*)$')                                      # cmd
]
```

如果地址是范围的，会把起始和结束分别提取出来作为一个元组保存，如(start,end)。其中 s 命令后部有替换规则，上面正则表达式匹配的时候会把它单独提取出来。

下面的这个函数就是遍历上面的正则表达式进行匹配分割，并把分割结果保存到上面定义的 AddressCommandStruct 结构中。

```python
def split_address_command(cmd: str) -> AddressCommandStruct:
    """ Splits an address command string into its components and returns an instance of AddressCommandStruct

    Args:
        cmd (str): The address command string to be split.

    Returns:
        AddressCommandStruct: An instance of AddressCommandStruct representing the address command.
    """
    for idx, pattern in enumerate(address_command_patterns):
        match = pattern.match(cmd)
        if match:
            return AddressCommandStruct(
                ['line_line', 're_re', 'line_re', 're_line', 're', 'line', None][idx],
                match.group(3) if idx < 4 else (match.group(1) if idx == 6 else match.group(2)),
                (match.group(1), match.group(2)) if idx < 4 else (None if idx == 6 else match.group(1)),
                match.group(4) if idx < 4 else (match.group(2) if idx == 6 else match.group(3)),
            )
    return None
```

main 函数这里就是将参数传入命令预处理，获得去除注释和空格的单个命令字符串，再进一步调用命令分割，获得一个包含 AddressCommandStruct 结构的列表。

```python
def main():
    args = process_args()

    pattern_list = preprocess_script(args['f'], True) if args['f'] else preprocess_script(args['patterns'], False)
    address_command_list = [split_address_command(pattern) for pattern in pattern_list]
```

3 命令执行

3.1 待处理内容读取

如果指定了文件路径，就打开文件，如果没有指定文件路径就从标准输入 stdin 读取。先预读取一行，然后循环读取下一行，读取下一行失败则代表已经成功读取的一行是最后一行。这个是否为最后一行的标志用于命令中“$”末行地址匹配，同时用于判断是否停止读取。

```python
def process_input(address_command_list: List[AddressCommandStruct], is_auto_print: bool, input_stream=None):
    """ Processes input lines based on the address commands and options.

    Args:
        address_command_list (List[AddressCommandStruct]): List of address commands.
        is_auto_print (bool): Whether automatic printing is enabled.
        input_stream: The input stream to read lines from. If None, sys.stdin is used.
    """
    if input_stream is None:
        input_stream = sys.stdin

    current_line = next(input_stream).strip()
    current_line_number = 1
    is_end = False
    while True:
        try:
            next_line = next(input_stream).strip()
        except Exception:
            is_end = True

        process_line(current_line, address_command_list, current_line_number, is_auto_print, is_end)

        if is_end:
            break

        current_line = next_line
        current_line_number += 1

```

3.2 命令处理

实现 p 命令打印操作，如果地址为空，则即使当前处理行为空字符串也要打印，地址不为空，则不打印空字符串；s 命令替换操作，先使用正则表达式分割出匹配内容、替换内容和全局替换标志 g，再用 Python 正则表达式替换执行替换操作；q 设置退出标志，d 返回 None 标志当前行删除。

```python
is_quit = False
s_cmd_re = re.compile(r'(\S)(.*?)\1(.*?)\1([g]?)$')

def process_command(line: str, cmd: str, pattern: str, address):
    """ Process a command on a given line based on the specified command, pattern, and address.

    Args:
        line (str): The input line to process.
        cmd (str): The command to execute.
        pattern (str): The pattern to use for searching or replacing.
        address: The address to specify the line(s) to operate on.

    Returns:
        str: The processed line after applying the command, or None if the command is 'd'.
    """
    global is_quit
    if cmd == 'p':
        if address is None and line is not None:
            print(line)
        elif address is not None and line:
            print(line)
    elif cmd == 'd':
        return None
    elif cmd == 's':
        match = s_cmd_re.match(pattern)
        if match:
            search, replace, flag = match.group(2), match.group(3), 0 if match.group(4) == 'g' else 1
            return re.sub(search, replace, line, count=flag)
        else:
            print_info('process_command: s no pattern.')
    elif cmd == 'q':
        if line is not None:
            is_quit = True
    else:
        print_info('process_command: unknow pattern.')

    return line
```

3.3 地址匹配

地址匹配在 process_line 函数中执行，前面每读取一行待处理内容就会调用 process_line 执行，process_line 根据地址匹配决定当前行是否调用 process_command 执行对应命令。

“-n”选项的判断也在这里，如果使用了“-n”参数则不打印该行。

另外如果执行了 d 命令返回 None，则删除了当前行的内容；如果执行了 q 设置了退出标志，会在 process_line 中执行 exit(0) 退出。