Minishell Parser - Visual Precedence Guide

Example Command

bash

```
echo hello | cat > out.txt && pwd || ls -la
```

Token Stream

```
[T_WORD:"echo"] [T_WORD:"hello"] [T_PIPE] [T_WORD:"cat"]
[T_REDIR_OUT] [T_WORD:"out.txt"] [T_AND] [T_WORD:"pwd"]
[T_OR] [T_WORD:"ls"] [T_WORD:"-la"]
```

Parsing Trace (Recursive Descent)

Level 1: `parse_or()` - LOWEST PRECEDENCE

```
parse_or() called
├─ Calls parse_and() for left side
│  └─ Returns: [AND node with pipe and pwd]
├─ Sees T_OR token
├─ Advances past ||
├─ Calls parse_and() for right side
│  └─ Returns: [CMD: ls -la]
└─ Creates OR node combining both sides
```

Level 2: `parse_and()` - MIDDLE PRECEDENCE

```
parse_and() called (first time - left side of OR)
├── Calls parse_pipe() for left side
│   └── Returns: [PIPE node with echo|cat>out.txt]
├── Sees T_AND token
├── Advances past &&
├── Calls parse_pipe() for right side
│   └── Returns: [CMD: pwd]
└── Creates AND node combining both sides
```

Level 3: `parse_pipe()` - HIGHER PRECEDENCE

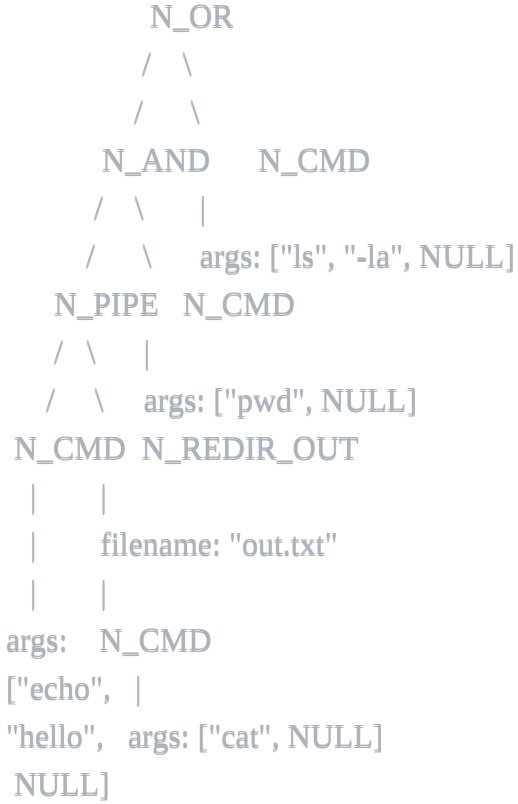
```
parse_pipe() called
├── Calls parse_command() for left side
│   └── Returns: [CMD: echo hello]
├── Sees T_PIPE token
├── Advances past |
├── Calls parse_command() for right side
│   └── Returns: [REDIR_OUT: cat > out.txt]
└── Creates PIPE node combining both sides
```

Level 4: `parse_redirections()` - HIGHEST PRECEDENCE (with commands)

```
parse_redirections() called (for "cat > out.txt")
├── Calls parse_simple_cmd()
│   └── Returns: [CMD: cat]
├── Sees T_REDIR_OUT token
├── Advances and reads "out.txt"
└── Creates REDIR_OUT node with filename and cmd as child
```

Final AST Structure

Tree in Detail with Node Types



Step-by-Step Parsing Flow

Step 1: Start at Lowest Precedence

```
parse_or()
  "I handle || operators"
  "Let me call parse_and() to get my left operand"
```

Step 2: Move to Next Level

```
parse_and()
  "I handle && operators"
  "Let me call parse_pipe() to get my left operand"
```

Step 3: Move Higher

```
parse_pipe()
  "I handle | operators"
  "Let me call parse_command() to get my left operand"
```

Step 4: Handle Command with Redirections

parse_command() → parse_redirections()
"First, let me get the base command"
parse_simple_cmd() → parse_primary()
"Found T_WORD: echo"
"Collect all consecutive words: echo hello"
Returns: CMD[echo, hello]
"No redirections after this command"
Returns: CMD[echo, hello]

Step 5: Back in parse_pipe()

parse_pipe()
left = CMD[echo, hello]
"I see a T_PIPE token!"
"Let me parse the right side"
Calls parse_command() again
Returns: REDIR_OUT[cat > out.txt]
Creates: PIPE[left=CMD[echo,hello], right=REDIR_OUT[...]]

Step 6: Back in parse_and()

parse_and()
left = PIPE[...]
"I see a T_AND token!"
"Let me parse the right side"
Calls parse_pipe() again
Returns: CMD[pwd]
Creates: AND[left=PIPE[...], right=CMD[pwd]]

Step 7: Back in parse_or()

```
parse_or()
  left = AND[...]
  "I see a T_OR token!"
  "Let me parse the right side"
  Calls parse_and() again
    Returns: CMD[ls, -la]
  Creates: OR[left=AND[...], right=CMD[ls,-la]]
  DONE!
```

Why This Order? Precedence!

Operator Precedence (Lowest to Highest)

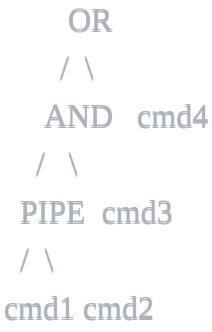
1. || (OR) - Evaluated LAST → Root of tree
2. **&&** (AND) - Evaluated before OR
3. | (PIPE) - Evaluated before AND
4. **Redirections** - Bind tightest to commands
5. **Commands** - Leaf nodes

Example: `cmd1 | cmd2 && cmd3 || cmd4`

Parsing order (function calls):

```
parse_or
  → parse_and
    → parse_pipe
      → parse_command (cmd1)
      → parse_command (cmd2)
    → parse_command (cmd3)
  → parse_and
    → parse_command (cmd4)
```

Result tree:



Execution order (left to right, respecting operators):

1. Execute `cmd1 | cmd2` (pipe)
2. If pipe succeeds (exit 0), execute `cmd3`
3. If AND result fails, execute `cmd4`

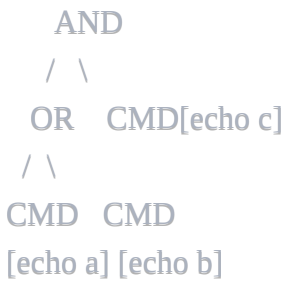
Parentheses Example

Command: `(echo a || echo b) && echo c`

Parsing:

```
parse_or()
  → parse_and()
    → parse_pipe()
      → parse_command()
        → parse_primary()
          "See T_OPEN_BRACKET!"
          → parse_or() [RECURSIVE!]
            "Parse inside parentheses"
            Returns: OR[echo a, echo b]
          "See T_CLOSE_BRACKET - good!"
          Returns: OR[echo a, echo b]
    "See T_AND!"
    → parse_pipe()
      Returns: CMD[echo c]
    Creates: AND[left=OR[...], right=CMD[echo c]]
```

Result tree:



The parentheses force the OR to be evaluated first (as left child of AND), even though AND normally has higher precedence!

Key Insights

1. **Lower precedence = Higher in tree** - OR is root, commands are leaves
2. **Recursive calls go DOWN the precedence chain** - or → and → pipe → command
3. **Operators at same level = left-to-right** - Multiple ANDs chain left-associative
4. **Parentheses restart from top** - parse_primary() calls parse_or() again
5. **Each level only sees its operator** - parse_pipe() only looks for |, ignoring && and ||