How to Parse Input

VE482 Project1 Presentation

Group 5, Yihua Liu, Boqun Li, Yu Xiao

UM-SJTU Joint Institute

September 30, 2021

<string.h>

- This is the C library includes most of the functions that operate on C strings.
- For C strings comparison, we have:
  - **strstr()**
  - **strcmp()**
- For C strings partition, we have:
  - **strtok()**

strstr()

Declaration

char *strstr(const char *str, const char *substr)

A function that can tell whether a string contains a substring.

Parameters

- *str*: pointer to the null-terminated byte string to examine
- *substr*: pointer to the null-terminated byte string to search for
  Return value

Return value

Pointer to the first character of the found substring in str, or a null pointer if such substring is not found. If substr points to an empty string, str is returned.

strcmp()

Declaration

```
int strcmp( const char *lhs, const char *rhs )
```

A function that can tell whether two strings are identical.

Parameters

- *lhs*, *rhs*: pointers to the null-terminated byte strings to compare

Return value

- Negative value if lhs appears before rhs in lexicographical order.
- Zero if lhs and rhs compare equal.
- Positive value if lhs appears after rhs in lexicographical order.

strtok()

Declaration

char *strtok(char *str, const char *delim)

A function that can spilt strings by certain delimiter.

Parameters

- *str*: pointer to the null-terminated byte string to tokenize
- *delim*: pointer to the null-terminated byte string identifying delimiters

Return value

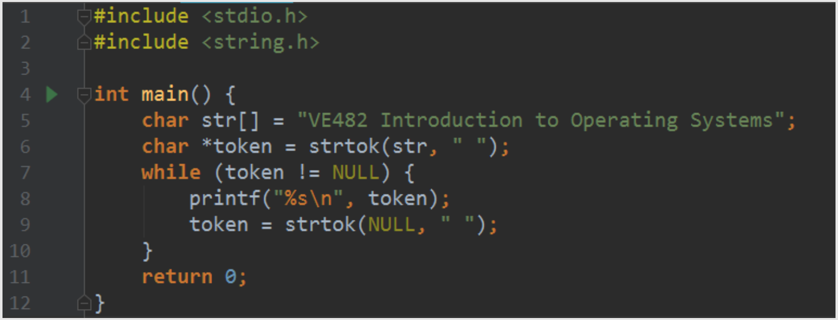
Returns pointer to the beginning of the next token or a null pointer if there are no more tokens.

strtok()

Note

This function is destructive. It will replace every string matching *delim* by '\0'.

Example

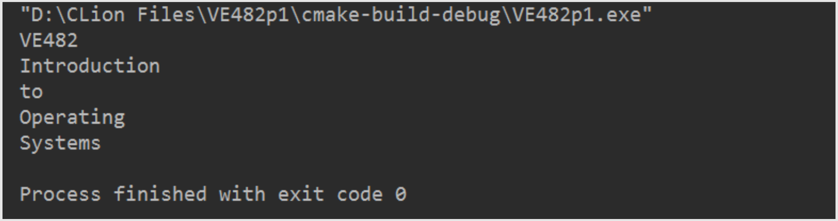


```c
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "VE482 Introduction to Operating Systems";
    char *token = strtok(str, " ");
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, " ");
    }
    return 0;
}
```

strtok()

Result



```
"D:\CLion Files\VE482p1\cmake-build-debug\VE482p1.exe"
VE482
Introduction
to
Operating
Systems

Process finished with exit code 0
```

parse char by char

1. Use a pointer pointing to the address of command as iterator.
2. Deal with one character in each iteration.
3. Deal with pipe '|' seperately.
4. there are several cases:
    1. *iter = ' '
    2. *iter = '>'
    3. *iter = '<'
    4. *iter = '>' && *(iter+1) = '>'
    5. *iter is other normal character

parse_v1()

Algorithm 1: parse()

```
Input   : original command line cmd, empty list argv[][]
Output: resulted char** argv list
parse(char *cmd, char** argv)
$j \leftarrow 0$,$i \leftarrow 0$
$char * iter \leftarrow cmd$
while *iter do
    if *iter == ' ' then
        $i \leftarrow i + 1$, $j \leftarrow 0$;
    else if *iter == '>' then
        skip spaces
        while next arg or end of line do
            $iter \leftarrow iter + 1$
    else if ... then
        ...
    else
        argv[i][j] = *iter
        $j \leftarrow j + 1$
```

parse()

1. redirection_parse()
2. Check whether `token[0] == '>'`
    - Check whether `strlen(token) == 2 && token[1] == '>'`
3. Check whether `token[0] == '<'`
4. Check whether `token[0] == '|'`
5. `token = strtok(NULL, token delimiters)`

This method would pass all words to `execvp()` function until `'>'`, `'<', or '|'`, so it is the same for cases with arguments and without arguments.

redirection_parse()

Algorithm 2: redirection_parse

```
Data: original command line cmdln
Result: command line with all proper spaces added parsedln,
        token
begin
    $j \leftarrow 0$
    for $i \leftarrow 1$ to strlen(cmdln) do
        /* Forward parsing                                  */
        if cmdln[i]=$<$ or $>$ or $|$ and cmdln[$i-1$]$\neq$' ' and not
         two adjacent $>$ then
            parsedln[j]$\leftarrow$' '
            $j \leftarrow j+1$
        /* Backward parsing                                 */
        if cmdln[$i-1$]=$<$ or $>$ or $|$ and cmdln[i]$\neq$' ' and not
         two adjacent $>$ then
            parsedln[j]$\leftarrow$' '
            $j \leftarrow j+1$
    token$\leftarrow$first word of parsedln separated by token delimeters
```

Thanks!