

# Programming C

## *2st report*

【担当教員】 長谷川 亨

WU HSINJU  
08Z17024  
電子情報工学部  
交換留学生  
2018/8/9

# 1 課題内容

C言語でいろんな機能ができる、そしてLinuxで実行するシェールを作る。

## 2 プログラム全体の説明

### 2.1 仕様

このプログラムはC言語で作ったLinuxのシェールである。実装した機能は、外部コマンド、ディレクトリ管理機能、ヒストリ管理機能、ワイルドカード機能、プロンプト機能、エイリアス機能、そして自分で考えた一つの機能である。スクリプト機能は実装できなかった。

### 2.2 処理の流れ

以下のステップをループする。

- Print the prompt
- Get the command line
- Store the line in history
- Parse the line
- Store the parsed line in args array
- (args[0]: command, args[1],args[2]....: other arguments)
- Check if args contain alias command
- Check if args contain wildcard command
- Check if args contain “!”
- Make a child process
- Execute args

## 2.3 実装方法

下に各項で説明する。

- **Print the prompt** : The name of the prompt is stored in the char prompt\_name, get the prompt name from it and print it to the screen every time when the loop starts. Also, the prompt name is initialized to “command”.
- **Get the command line** : I wrote a function lsh\_read\_line to read the command line and store it in char \*line. The lsh\_read\_line function is basically using getchar() function.
- **Parse the line** : I wrote a function lsh\_split\_line to parse the line with space, basically using strtok function.
- **Store the parsed line in args** : char \*\*args, store the main command in args[0], the other arguments in args[1],args[2]...
- **Check alias command** : please refer to 8.
- **Check wildcard command** : please refer to 6.
- **Check ! Command** : please refer to 5.
- **Make a child process** : using fork().
- **Execute args** : using execvp(args[0],args).

## 3 外部コマンド機能

### 3.1 仕様

ls、firefox などの外部コマンドをできるようにした。

### 3.2 処理の流れ

Input the args[0], args to `execvp()` function.

### 3.3 実装方法

`execvp(args[0],args)`.

## 4 デイリクトリ管理機能

### 4.1 仕様

cd : change directory

pushd : save the current directory to stack

popd : remove the directory saved in the top of stack

dirs : show the directories in stack

### 4.2 処理の流れ

#### 4.2.1 cd 機能

- If `args[0] == cd`
- Call the function `lsh_cd`

#### 4.2.2 push 機能

- If `args[0] == push`
- Call the function `lsh_pushd`

#### 4.2.3 popd 機能

- If `args[0] == pope`
- Call the function `lsh_popd`

#### 4.2.3 dirs 機能

- If `args[0] == dirs`
- Call the function `lsh_dirs`

### 4.3 実装方法

#### 4.3.1 cd 機能

I wrote a function `lsh_cd`, it will change the directory to `args[1]` by using `chdir( )` function. If `args[1] == NULL`, then set `args[1]` to `"/home/"`.

#### 4.3.2 pushd, popd, dirs 機能

I set global variables `char *stack[ ]` to store the directories, and `int stack_number` to count the number of directories stored in stack. And I also wrote the following functions,

**lsh\_pushd** : get the current directory by using `getcwd(NULL,0)`, and store it in the `stack[stack_number]`, `stack_number++`.

**lsh\_popd** : remove the directory stored in the top of stack by using `strcpy("stack[stack_number],NULL")`, `stack_number--`.

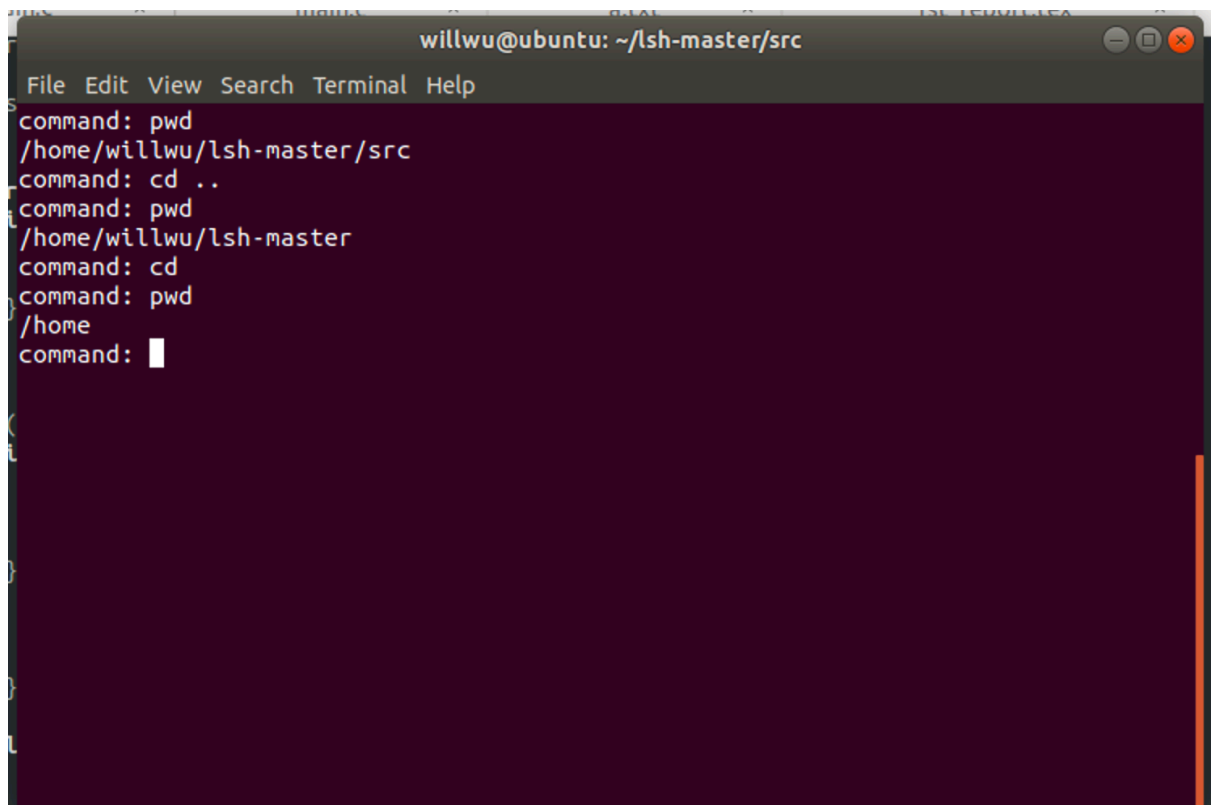
**lsh\_dirs** : for(`i=0`; `i<stack_number`; `i++`) print the directories stored in stack.

## 4.4 テスト

### 4.4.1 cd 機能

方法：type cd and cd .. in the command line, and use pwd command to check if we really entered the right directory.

結果：succeeded

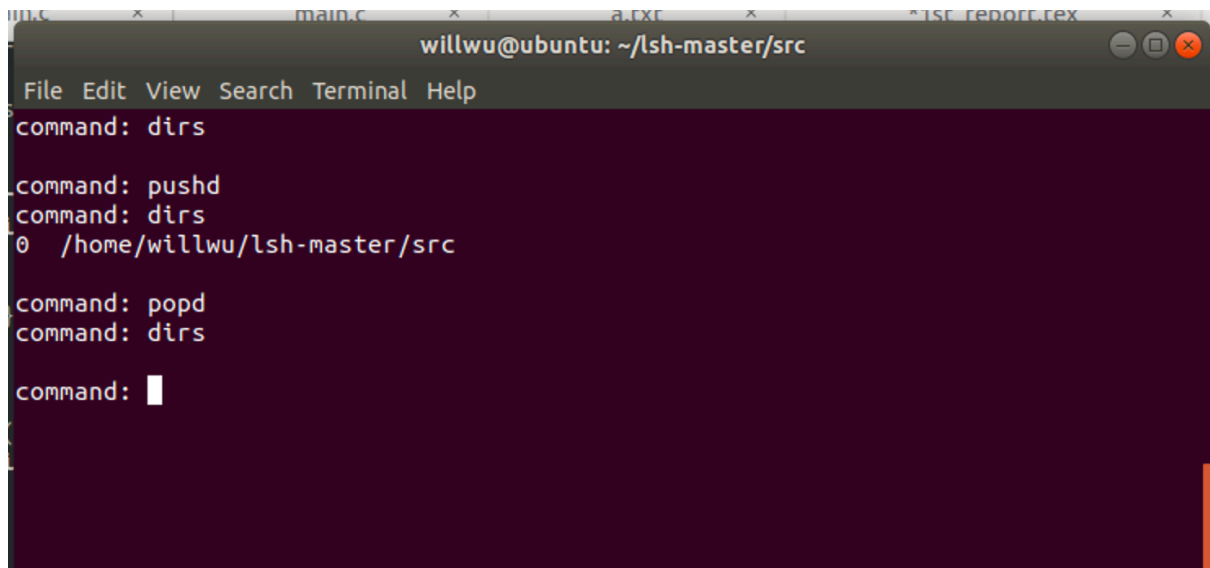


```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: pwd
/home/willwu/lsh-master/src
command: cd ..
command: pwd
/home/willwu/lsh-master
command: cd
command: pwd
/home
command: 
```

#### 4.4.2 pushd, popd, dirs 機能

方法: type pushd and popd to the command line, and use dirs to check if the current directory is saved to the stack.

結果 : succeeded

A terminal window titled 'willwu@ubuntu: ~/lsh-master/src' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following sequence of commands and output:

```
command: dirs
command: pushd
command: dirs
0 /home/willwu/lsh-master/src
command: popd
command: dirs
command: 
```

## 5 ヒストリ機能

### 5.1 仕様

前に実行したコマンドをヒストリに保存

### 5.2 処理の流れ

- Get command line

- Save the command to history
- history\_number++

## 5.3 実装方法

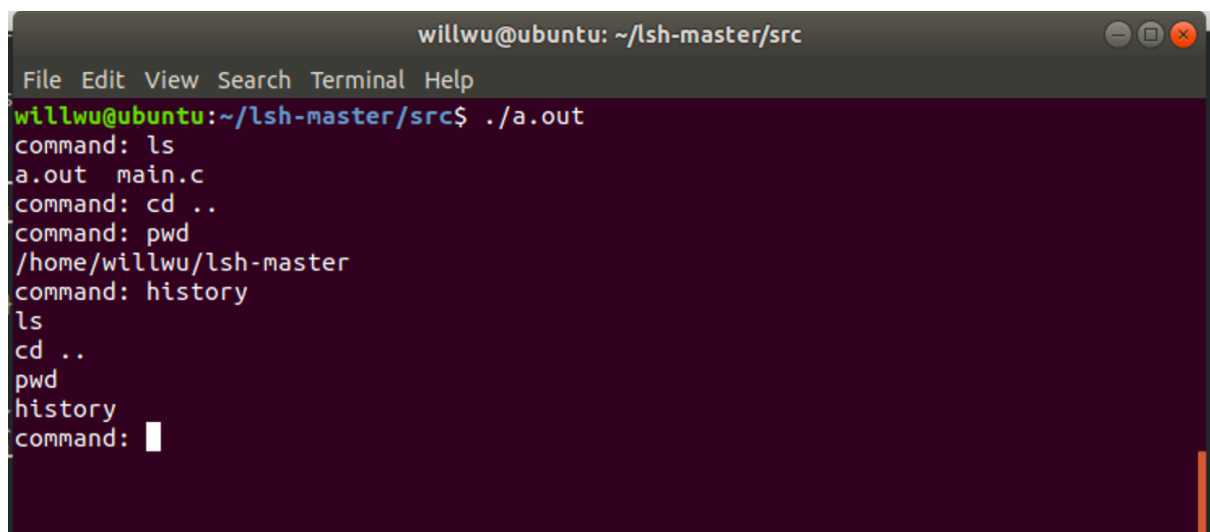
Declare global variables `char history[ ][ ]` to store the history command, and `int history_number` to count the number of commands in history.

When the “history” is input to the command line, `lsh_history` function will print out all commands stored in history.

## 5.4 テスト

方法：input some commands, and then execute the history command.

結果：succeeded



```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
willwu@ubuntu:~/lsh-master/src$ ./a.out
command: ls
a.out main.c
command: cd ..
command: pwd
/home/willwu/lsh-master
command: history
ls
cd ..
pwd
history
command: 
```



# 6 !string、!! 機能

## 6.1 仕様

!! : execute the last command.

!string : execute the command stored in history and has matched the string

## 6.2 処理の流れ

- Get command line
- if the first letter of command is “!”
  - if the second letter is also “!”, execute the last command stored in history
  - if the second letter is not “!” but a string, search the string in history.

## 6.3 実装方法

use while loop and strcmp( ) to check if the first letter of command line is “!”. If it is, then use the same way to check the second letter. If the second letter is still “!”, set the args to history[history\_number-2], then when the shell execute the args, the last command stored in history will be executed.

The other case, if the first letter of command line is “!” but the second letter is not, then it means it is “!string”. So, set the args to search\_command\_from\_history( ).

search\_command\_from\_history( ) is made to search if there is any command stored in history who match the string. It also use while loop and strcmp( ) function to match the string and history command.

## 6.4 テスト

方法 : input some commands first, and test “!!” and “!string”

結果 : succeeded

```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: ls
a.out main.c src
command: !!
a.out main.c src
command:
```

```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: history
ls -l
pushd
popd
dirs
ls
clear
history
command: !l
a.out main.c
command: █
```

## 7 ワイルドカード機能

### 7.1 仕様

コマンドに記号 "\*" があつたら、"\*"をフォルダにある全てのファイル名に置換する。

### 7.2 処理の流れ

- Check if the args contain "\*"
- If yes, replace the the array item with all files name in the folder
- (for example, if args[1] contains "\*", then replace args[1] with all files name)

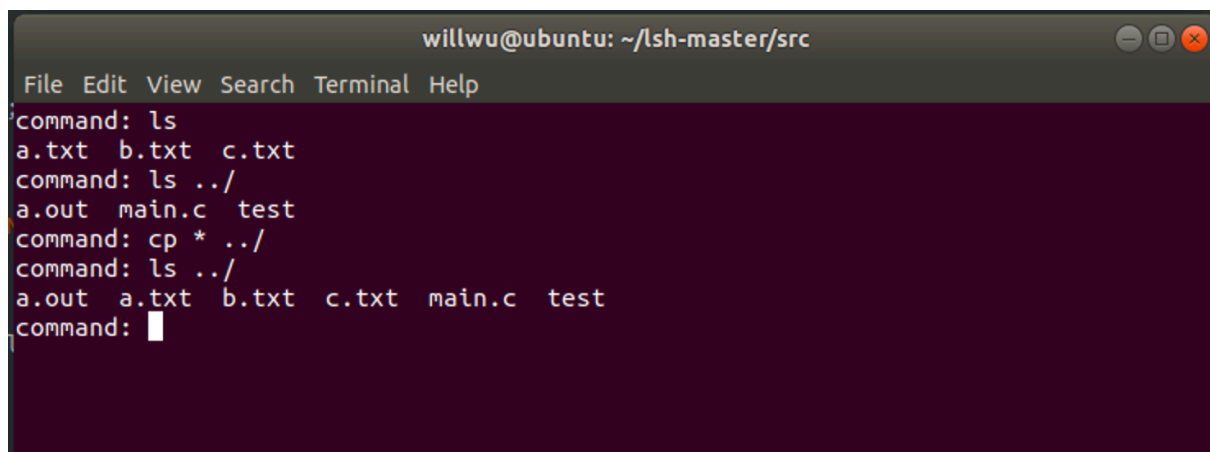
## 7.3 実装方法

after getting the command line and parse it into args, use while loop and strcmp() function to check if it contains “\*”. If yes, then use the approach we learned in the 演習課題13 to get all files name in the directory and replace “\*” with them. And then execute args.

## 7.4 テスト

方法: In the /home/willwu/lsh-master/src/test , there are a.txt, b.txt, c.txt 3 files, I will use the command “ cp \* ../ ” to test if the wildcard function works.

結果：succeeded

A terminal window titled 'willwu@ubuntu: ~/lsh-master/src' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and outputs:

```
command: ls
a.txt b.txt c.txt
command: ls ../
a.out main.c test
command: cp * ../
command: ls ../
a.out a.txt b.txt c.txt main.c test
command: 
```

# 8 プロンプト機能

## 8.1 仕様

prompt コマンドによってプロンプトを変更できるようにする。

## 8.2 処理の流れ

If the user input the command “ prompt string ”, change prompt name to the string.

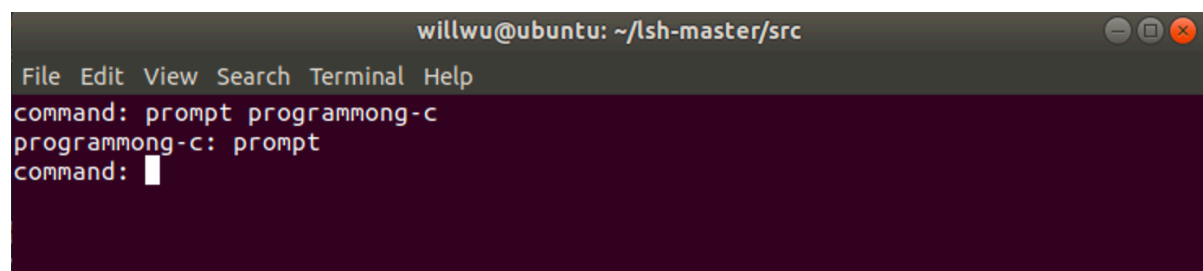
## 8.3 実装方法

The prompt name is stored in the global variable char \*prompt\_name. Every time when the loop begins, the prompt name will be printed to the screen. If the user inputs the command “ prompt string ”, then the lsh\_prompt function will be executed. The lsh\_prompt function will change the prompt name to the string inputed by user, by changing the value of global variable prompt\_name.

## 8.4 テスト

方法： type the command “prompt programming-c” , and “prompt”, see if the prompt name changes to the string or the default properly.

結果： succeeded

A terminal window titled 'willwu@ubuntu: ~/lsh-master/src' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following sequence of commands and prompts: 'command: prompt programming-c', 'programming-c: prompt', and 'command: ' followed by a cursor. The background is dark purple.

```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: prompt programming-c
programming-c: prompt
command: █
```

# 9 alias, unalias機能

## 9.1仕様

alias コマンドによってコマンドの別名を設定する。

” alias command1 command2 ” を入力すると、command2 にcommand1で別名つける。それで、次command1を入力したとき、command2が実行られる。そして、unalias command1を入力したら、別名のcommand 1 を削除する。

## 9.2処理の流れ

when the user inputs “alias command1 command2” to the command line, store both command command1 and command2 to the structures of alias commands. Every time after the loop began and the user inputted the command, check if there is any command matches in the alias structures. If yes, then replace the nicknamed command with original command in the args. If the user inputs “unalias command1”, then remove the command 1 in structures of alias commands.

## 9.3実装

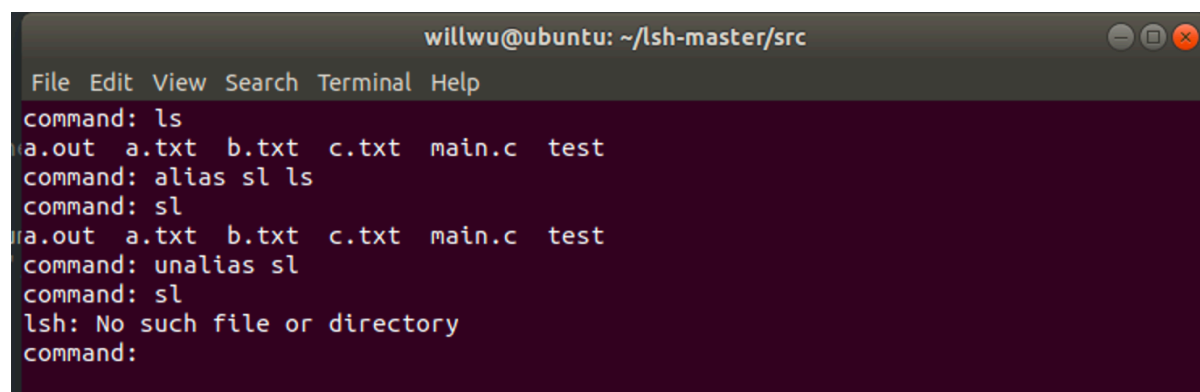
I set a global structure alias[ ]. In every structure, there is char new\_name, char original\_name, which is going to store the new name of command and the original name of command. For example, if the user inputs “ alias sl ls ” to the command line, sl will be saved to the alias[ ].new\_name, ls will be saved to alias[ ].original\_name. From next time the loop starts, it will check if the user inputted the nickname command by searching it in the alias structures and then replace it in args with its original name. So, if the user inputs the command “sl”, then args[0] will be replaced to “ls”. Through that, when the program executes the args, instead of “sl”, “ls” will be executed.

In the other hand, if the user inputs “alias sl”, then we will search “sl” in the alias structures and remove it.

## 9.4テスト

方法：Input “ alias sl ls ” to make sl as the new name of ls command. After that, unalias it.

結果：succeeded



```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: ls
a.out a.txt b.txt c.txt main.c test
command: alias sl ls
command: sl
a.out a.txt b.txt c.txt main.c test
command: unalias sl
command: sl
lsh: No such file or directory
command:
```

# 1 0 自分で考えた機能-push

## 10.1仕様

Push command allowed user to switch the directory to the directory stored in the directory stack quickly by using it. For example, if there are

0     directory0

1     directory1

2     directory2

in the directory stack, and the user wants to access the directory2, then all he has to do is to input “push 1” command.

## 10.2処理の流れ

Get the command and the number of directory he wants to switch to, search the number in the directory stack and find the matched directory. Last, change the directory.

## 10.3実装方法

use `for(i=0 ; i<stack_number ; i++)` to and if `strcmp(args[1],stack[i])` to find the directory matches to the number inputed by user, and then change directory with `chdir()` command.

## 10.4テスト

方法：there are already some directories in the directory stack, and I will use the push command to switch to one of them.

結果：succeeded

```
willwu@ubuntu: ~/lsh-master/src
File Edit View Search Terminal Help
command: dirs
0 /home/willwu/lsh-master/src
1 /home/willwu/lsh-master
2 /home/willwu
3 /home

command: pwd
/home
command: push 1
command: pwd
/home/willwu/lsh-master
command:
```

# 1 1 工夫点、考察

In the beginning , I didn't know too much about how actually linux command works, I thought I can easily execute any commands through the shell with `exec()` function, but apparently I was wrong.

For example, when I tried to execute `cd` command through `exec` function in the shell, I failed. After searching and searching for where the problem is, it turns out that `cd` does not exist as an executable command because a process can only change the working directory of itself, not of its parent. I would need to implement `cd` myself as a builtin, using the `chdir()` system call.

Also, in order to execute the command through `exec()` function, I need to parse the command line inputed by user and split it into arrays of string properly as the input of `exec()` function. That took a lot of string work, such as `strcmp()`, `strcpy()`, `strtok()`, which I was not too familiar with. I found that if I want to change the string stored in the pointer, I can not just use `string1 = string2`, instead I need to use `strcpy(string1, string2)`. And if I want to split up the `string1` with space, `strtok(string1, " ")` did it for me.

The last thing which took me a lot of works was the wildcard function. I was wondering how can I get all the files name in the directory and replace the "\*" symbol with them. Then the homework 13 came to my mind and helped me out of that.

These were the issues I had trouble dealing on when I was writing this program.

# 1 2 感想

まず、先生、自分の力の不足そしてもうすぐ帰国するから帰国の準備のため、このレポート課題を遅れて提出してしまつて本当にすみませんでした。でもこの課題でいろいろ勉強になったと思います。Linuxシェルの動作原理をだいぶ分かるようになりました。たまにわからないことがあつて、クラスメートに聞いたら自分の日本語能力が足りなくてクラスメートが何回も説明してくれたのに私が理解できなくて、わかつたフリにしてすんだ困る時もあったけど、頑張つてググつて答えを探してこの課題をやりました。とても達成感があります。

最後に、プログラミングCは、この学期で一番充実な授業だったと思います。先生ありがとうございます、お疲れ様でした。

# 1 3 参考資料

<https://brennan.io/2015/01/16/write-a-shell-in-c/>



# 1 3 プログラムリスト

```
#include <sys/wait.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define stack_max_buff 32
#define history_max_buff 32
#define alias_max_buff 32
#define wild_card_buff 100

char *stack[stack_max_buff];
int stack_number = 0;
int history_number = 0;
int alias_number = 0;
char history_command[history_max_buff][10];
char prompt_name[15] = "command";
struct Alias{
    int index;
    char new_name[10];
    char original_name[10];
}alias[alias_max_buff];

//char *history[history_max_buff]; // declare storage for history
//int history_number = 0;          // history counter
```

```

/*
    Function Declarations for builtin shell commands:
*/

int lsh_cd(char **args);
int lsh_help(char **args);
int lsh_exit(char **args);
int lsh_pushd(char **args);
int lsh_popd(char **args);
int lsh_dirs(char **args);
int lsh_history(char **args);
int lsh_prompt(char **args);
int lsh_alias(char **args);
int lsh_unalias(char **args);
int lsh_push(char **args);

/*
    List of builtin commands, followed by their corresponding functions.
*/
char *builtin_str[] = {
    "cd",
    "help",
    "exit",
    "dirs",
    "history",
    "prompt",
    "pushd",
    "popd",
    "alias",
    "unalias",
    "push"
};

```

```

int (*builtin_func[]) (char **) = {
    &lsh_cd,
    &lsh_help,
    &lsh_exit,
    &lsh_dirs,
    &lsh_history,
    &lsh_prompt,
    &lsh_pushd,
    &lsh_popd,
    &lsh_alias,
    &lsh_unalias,
    &lsh_push

};

int lsh_num_builtins() {
    return sizeof(builtin_str) / sizeof(char *);
}

/*
    Builtin function implementations.
*/

/**
    @brief Bultin command: change directory.
    @param args List of args. args[0] is "cd". args[1] is the directory.
    @return Always returns 1, to continue executing.
*/

int lsh_cd(char **args)
{
    if (args[1] == NULL) {

```

```

        //printf("NULL\n");
        chdir("/home/");
        //fprintf(stderr, "lsh: expected argument to \"cd\\\"\\n");
    } else {
        if (chdir(args[1]) != 0) {
            perror("lsh");
        }
    }
    return 1;
}

/**
 * @brief Builtin command: print help.
 * @param args List of args. Not examined.
 * @return Always returns 1, to continue executing.
 */
int lsh_help(char **args)
{
    int i;
    printf("Stephen Brennan's LSH\n");
    printf("Type program names and arguments, and hit enter.\n");
    printf("The following are built in:\n");

    for (i = 0; i < lsh_num_builtins(); i++) {
        printf("  %s\n", builtin_str[i]);
    }

    printf("Use the man command for information on other programs.\n");
    return 1;
}

/* dirs functopn */
int lsh_dirs(char **args){

```

```

    int i=0;
    while(stack[i]!=NULL){
        printf("%0d  %s\n",i,stack[i]);
        i++;
    }
    printf("\n");
    return 1;
}

/* pushd functopn */
int lsh_pushd(char **args){
    //printf("***** pushd function *****\n");
    stack[stack_number] = getcwd(NULL,0);
    //printf("dir has been saved, stack[%0d] = %s\n",stack_number,stack[stack_number]);
    stack_number ++;
    return 1;
}

/* popd functopn */
int lsh_popd(char **args){
    //printf("***** popd function *****\n");
    stack_number--;
    stack[stack_number] = NULL;
    //printf("the dir has been removed\n");
}

int lsh_push(char **args){
    //printf("push func!\n");
    //printf("push to %s\n",stack[atoi(args[1])]);
    chdir(stack[atoi(args[1])]);
    return 1;
}

```

```

    /* alias function */
int lsh_alias(char **args){
    //printf("before: args[1] = %s, args[2] = %s\n",args[1],args[2]);
    if(args[1]==NULL){
        //printf("All alias commands : \n");
        for(int i=0;i<alias_number;i++){
            //printf("%s %s\n",alias[i].new_name,alias[i].original_name);
        }
        return 1;
    }
    //printf("***** alias function *****\n");
    alias[alias_number].index = alias_number;
    strcpy(alias[alias_number].new_name,args[1]);
    strcpy(alias[alias_number].original_name,args[2]);
    //printf("new command name has been saved!\n");
    //printf("alias[%d]: %s =
%s\n",alias[alias_number].index,alias[alias_number].new_name,alias[alias_number].original
_name);
    alias_number++;
    //printf("after free : args[1] = %d, args[2] = %d\n",args[1],args[2]);
    //printf("alias number = %d\n",alias_number);
    return 1;
}

/* unalias function */
int lsh_unalias(char **args){
    for(int i=0;i<alias_number;i++){
        if(strcmp(args[1],alias[i].new_name)==0){
            strcpy(alias[i].new_name,"");
            //printf("alias command %s has been romoved\n",args[1]);
        }
    }
}

```

```

/* prompt function */
int lsh_prompt(char **args){
    char *default_prompt;
    default_prompt = "command";
    if(*(args+1)=='\0'){*(args+1)=default_prompt;} //strcpy(args[1],"command");
    strcpy(prompt_name,args[1]);
    return 1;
}

```

```

int lsh_history(char **args){
    print_history();
    return 1;
}

```

```

int print_history(){
    for(int i=0;i<history_number;i++) printf("%s\n",history_command[i]);
}

```

```

/**
    @brief Builtin command: exit.
    @param args List of args. Not examined.
    @return Always returns 0, to terminate execution.
*/

```

```

int lsh_exit(char **args)
{
    return 0;
}

```

```

/**
    @brief Launch a program and wait for it to terminate.

```

```

@param args Null terminated list of arguments (including program).
@return Always returns 1, to continue execution.
*/
int lsh_launch(char **args)
{
    pid_t pid;
    int status;

    pid = fork();
    if (pid == 0) {
        // Child process
        if (execvp(args[0], args) == -1) {
            perror("lsh");
        }
        exit(EXIT_FAILURE);
    } else if (pid < 0) {
        // Error forking
        perror("lsh");
    } else {
        // Parent process
        do {
            waitpid(pid, &status, WUNTRACED);
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
    }

    return 1;
}

/**
@brief Execute shell built-in or launch program.
@param args Null terminated list of arguments.
@return 1 if the shell should continue running, 0 if it should terminate
*/

```



```

int lsh_execute(char **args)
{
    int i;

    if (args[0] == NULL) {
        // An empty command was entered.
        return 1;
    }

    for (i = 0; i < lsh_num_builtins(); i++) {
        if (strcmp(args[0], builtin_str[i]) == 0) {
            //printf("find builtin function : %d\n",i);
            return (*builtin_func[i])(args);
        }
    }

    return lsh_launch(args);
}

#define LSH_RL_BUFSIZE 1024
/**
    @brief Read a line of input from stdin.
    @return The line from stdin.
 */
char *lsh_read_line(void)
{
    int bufsize = LSH_RL_BUFSIZE;
    int position = 0;
    char *buffer = malloc(sizeof(char) * bufsize);
    int c;

    if (!buffer) {
        fprintf(stderr, "lsh: allocation error\n");
    }

```

```

    exit(EXIT_FAILURE);
}

while (1) {
    // Read a character
    c = getchar();

    if (c == EOF) {
        exit(EXIT_SUCCESS);
    } else if (c == '\n') {
        buffer[position] = '\0';
        return buffer;
    } else {
        buffer[position] = c;
    }
    position++;

    // If we have exceeded the buffer, reallocate.
    if (position >= bufsize) {
        bufsize += LSH_RL_BUFSIZE;
        buffer = realloc(buffer, bufsize);
        if (!buffer) {
            fprintf(stderr, "lsh: allocation error\n");
            exit(EXIT_FAILURE);
        }
    }
}
}
}
}

```

```

#define LSH_TOK_BUFSIZE 64
#define LSH_TOK_DELIM " \t\r\n\a"

```

```

/**
  @brief Split a line into tokens (very naively).
  @param line The line.
  @return Null-terminated array of tokens.
 */
char **lsh_split_line(char *line)
{
    int bufsize = LSH_TOK_BUFSIZE, position = 0;
    char **tokens = malloc(bufsize * sizeof(char*));
    char *token, **tokens_backup;

    if (!tokens) {
        fprintf(stderr, "lsh: allocation error\n");
        exit(EXIT_FAILURE);
    }

    token = strtok(line, LSH_TOK_DELIM);
    while (token != NULL) {
        tokens[position] = token;
        position++;

        if (position >= bufsize) {
            bufsize += LSH_TOK_BUFSIZE;
            tokens_backup = tokens;
            tokens = realloc(tokens, bufsize * sizeof(char*));
            if (!tokens) {
                free(tokens_backup);
                fprintf(stderr, "lsh: allocation error\n");
                exit(EXIT_FAILURE);
            }
        }
    }

    token = strtok(NULL, LSH_TOK_DELIM);

```

```

    }
    tokens[position] = NULL;
    return tokens;
}

char *get_wildcard(char **args)
{
    struct stat  filestat;
    struct dirent *directory;
    DIR          *dp;
    static char str[wild_card_buff];

    dp = opendir(".");

    while((directory=readdir(dp))!=NULL){
        if(!strcmp(directory->d_name, ".") ||
           !strcmp(directory->d_name, ".."))
            continue;
        if(stat(directory->d_name,&filestat)==-1){
            perror("main");
            exit(1);
        }else{
            strcat(str,directory->d_name);
            strcat(str, " ");
        }
    }

    //printf("%s\n",str);

    closedir(dp);
    return str;
    exit(0);
}

```

```

char *search_command_from_history(char *line){
    int i;
    char str[10];
    char *command;
    for(i=0;i<strlen(line);i++) str[i] = *(line+i+1);
    //printf("history command matching...\n");
    //printf("input line = %s\nstr = %s\n",line,str);
    //printf("history number = %d\n",history_number);
    for(i=history_number-1;i>=0;i--){
        //printf("history[%d] = %s\n",i,history_command[i]);
        if(strncmp(str,history_command[i],strlen(str))==0){
            //strcpy(command,history_command[i]);
            command = history_command[i];
            //printf("matched to history[%d] = %s,command =
%s\n",i,history_command[i],command);
            break;
        }
    }
    if(command == NULL) printf("Error : can't find command in history!!\n");

    return command;
}

int get_args_number(char **args){
    int i=0;
    int command_number=0;
    while(*(args+i)!='\0'){
        i++;
        command_number++;
    }
    return command_number;
}

```

```

}

/**
 @brief Loop getting input and executing it.
 */
void lsh_loop(void)
{
    char *tmp;
    char *line;
    char **args;
    char args_tmp[10];
    int status;
    int args_number;
    int i;

    //stack[0] = getcwd(NULL,0);
    //printf("path = %s\n",stack[0]);

    do {
        /* prompt */
        printf("%s: ",prompt_name);

        /* read command */
        line = lsh_read_line();
        //printf("size = %d\n",strlen(line));

        /* history*/
        strcpy(history_command[history_number],line);
        //          printf("history[%d] =
%s\n",history_number,history_command[history_number]);
        history_number++;

        /* check alias*/
        for(int i=0;i<alias_number;i++){

```

```

        if(strcmp(line,alias[i].new_name)==0){
            //printf("found command in alias :
%s\n",alias[i].original_name);
            strcpy(line,alias[i].original_name);
        }
    }

    /* check the wildcard and !string */
    if(*line=="!"){
        if(*(line+1)=="!"){
            if(history_number==0) printf("Error, no previous command\n");
            else args = lsh_split_line(history_command[history_number-2]);

        }else{
            char *command;
            command = search_command_from_history(line);
            args = lsh_split_line(command);
        }
    }else{
        //printf("case 3\n");
        args = lsh_split_line(line);
        //print_command(args);
    }

    /*get args number */
    args_number = get_args_number(args);
    //printf("args number = %d\n",args_number);

    /* copy the last args*/
    //printf("args[%d] = %s\n",args_number-1,args[args_number-1]);
    strcpy(args_tmp,args[args_number-1]);

```

```

//printf("args[%d] = %s\n",args_number-1,args_tmp);

/* check wild card */
for(i=0;i<args_number;i++){
    if(strcmp(args[i], "*")==0){
        //printf("find star symble in args[%d] = %s\n",i,args[i]);
        char str[100];

        strcpy(str,args[0]);
        strcat(str, " ");
        strcat(str,get_wildcard(args));
        strcat(str, " ");
        strcat(str,args_tmp);
        //printf("string = %s\n",str);

        free(line);
        free(args);
        strcpy(line,str);
        //line = test();
        args = lsh_split_line(line);
        //printf("line = %s\nargs = %s\n",line,args);
        break;
    }
}

//print_command(args);
status = lsh_execute(args);
free(line);
free(args);
} while (status);
}

```



```

int print_command(char **args){
    int i=0;
    int command_number=0;
    while(*(args+i)!='\0'){
        printf("args[%0d] = %s\n",i,*(args+i));
        i++;
        command_number++;
    }
    //printf("number of command = %d\n",command_number);
}

/**
 @brief Main entry point.
 @param argc Argument count.
 @param argv Argument vector.
 @return status code
 */
int main(int argc, char **argv)
{
    // Load config files, if any.

    // Run command loop.
    lsh_loop();
    //get_wildcard();

    // Perform any shutdown/cleanup.

    return EXIT_SUCCESS;
}

```