プログラミングC演習報告書
プログラミングC第2回レポート課題

【担当教員】　樽谷 優弥, まつ本 真佑　教員

【提出者】　小山 亮　(09B15028)
計算機科学コース・2年
u745409b@ecs.osaka-u.ac.jp
【提出日】　平成28年8月4日

1　課題内容

以下の機能を有するサブセット版のシェルを C 言語で作成せよ。

- 外部コマンドの実行機能
  - このプログラムに実装されていないコマンドに関して、このプログラムを実行しているシェルのコマンドを実行できるようにせよ。
- ディレクトリの管理機能
  - cd コマンド -　カレントディレクトリを指定したディレクトリに、指定されていない場合はホームディレクトリに、移動できるようにせよ。
  - pushd コマンド - ディレクトリスタックの一番上にカレントディレクトリを保存できるようにせよ。
  - dirs コマンド - ディレクトリスタックの内容を表示できるようにせよ。
  - popd コマンド - ディレクトリスタックの一番上のディレクトリにカレントディレクトリを移動できるようにせよ。
- ヒストリー機能
  - history コマンド -　実行したコマンドを保存しておき、そのリストを実行した順番とともに出力できるようにせよ。
  - !!コマンド -　前回実行したコマンドを実行できるようにせよ。
  - ![string] コマンド -　保存したコマンドのうち [string] に前方一致する最新のコマンドを実行できるようにせよ。
- ワイルドカード機能
  - “*” と入力された部分をカレントディレクトリ内のすべてのディレクトリ、ファイルに適用されるようにせよ。
- プロンプト機能
  - プロンプトを指定した文字列に変更できるようにせよ。
- スクリプト機能
  - このプログラム実行時にリダイレクトで受け渡されたファイルからもコマンドを実行できるようにせよ。
- エイリアス機能
  - alias コマンド - コマンドに対して指定した文字列が入力された時も実行できるようにせよ。
  - unalias コマンド -コマンドに指定した文字列の関連付けを解除できるようにせよ。

またこれら以外にも 1 つ以上、自分で考えた機能を実装せよ。

2　プログラム全体の説明

C 言語を用いてサブセット版のシェルを作成した。指定されたコマンドはプログラムの中で実行し、実装されていないコマンドについては外部コマンドとしてプログラムを実行しているシェルで実行されるようにした。

2.1　シェルの仕様

本章では実装した機能について説明する。

- 外部コマンドの実行機能
  - このプログラムに実装されていないコマンドを外部コマンドとしてこのプログラムを実行しているシェルで実行されるようにした。
- ディレクトリの管理機能
  - cd コマンド -　カレントディレクトリを指定したパスに移動できるようにした。
  - pushd コマンド - ディレクトリスタックを作成しスタックのの一番上に指定したパスをを保存できるようにした。指定しない場合はカレントディレクトリを保存する。
  - dirs コマンド - ディレクトリスタックの内容を表示できるようにした。
  - popd コマンド - ディレクトリスタックの一番上のディレクトリにカレントディレクトリを移動できるようにした。
- ヒストリー機能
  - history コマンド -　実行したコマンドを構造体リストに保存しておき、そのコマンドのリストを実行した順番とともに出力できるようにした。保存できるコマンドは最新の 32 個までである。
  - !!コマンド -　前回実行したコマンドを実行できるようにした。
  - ![string] コマンド -　保存したコマンドのうち [string] に前方一致する最新のコマンドを実行できるようにした。
- ワイルドカード機能
  - “*” と入力された部分をカレントディレクトリ内のすべてのディレクトリ、ファイルに適用されるようにした。
- プロンプト機能
  - プロンプトを指定した文字列に変更できるようにした。指定されない場合は “prompt” となる。
- スクリプト機能
  - このプログラム実行時にリダイレクトで指定されたファイルからもコマンドを実行できるようにした。
- エイリアス機能
  - alias コマンド - コマンドを指定した文字列が入力された時も実行できるようにした。
  - unalias コマンド -コマンドに指定した文字列の関連付けを解除できるようにした。

2.2 処理の流れ・実装方法

まず、コマンドラインを配列に読み込む。次に関数によってコマンドラインを引数として単語ごとに分ける。実際には、コマンドラインの単語の先頭アドレスをポインタ配列に代入していく。その後、ポインタ配列の 1 つ目にはコマンドが入っているので if 文によってコマンドごとに行う処理を変える。まず実装しているコマンドであれば if 文で分岐してそれぞれの機能が実現される。実装されていないコマンドであれば、まず alias コマンドによって登録されたコマンドでないかを確認する。登録されたコマンドでなければ、ポインタ配列が関数に渡され、外部コマンドとして実行される。プログラム終了の際は、構造体リストや構造体二分木のために動的に確保した領域を開放する。

3 外部コマンド実行機能

3.1 仕様

このプログラムに実装されていないコマンドを外部コマンドとして実行する。コマンドラインの最後に “&” があればバックグラウンドで実行する。

3.2 処理の流れ

子プロセスを生成し、その中でコマンドを実行する。コマンドの処理が終われば子プロセス上で実行しているプログラムを終了させる。バックグラウンドでの実行であれば、親プロセスは子プロセスの終了を待たずに次のプロンプトを表示し入力待ちとなる。フォアグラウンドでの実行であれば、親プロセスは子プロセスの終了を待つ。

3.3 実装方法

関数 execvp によって実装した。関数の引数はコマンドラインを分割したもの、コマンドの状態である。

3.4 テスト

3.4.1 テスト方法

アプリケーション emacs を起動して、コマンド ps によって起動を確認する。

3.4.2 テスト結果

```
prompt : ps
  PID TT  STAT     TIME COMMAND
13233  1  Is+   0:00.04 -csh (csh)
13267  2  Ss    0:00.08 csh
13294  2  I    19:02.81 firefox
14987  2  S+    0:00.00 ./mysh
14990  2  R+    0:00.00 ps

prompt : emacs &

prompt : ps
  PID TT  STAT     TIME COMMAND
13233  1  Is+   0:00.04 -csh (csh)
13267  2  Is    0:00.08 csh
13294  2  I    19:04.46 firefox
14987  2  S+    0:00.00 ./mysh
15007  2  S+    0:00.32 emacs (emacs-24.5)
15012  2  R+    0:00.00 ps
```

コマンドによって emacs が起動したことがわかる。

4 ディレクトリの管理機能

4.1 仕様

この機能にはコマンド cd、pushd、dirs、popd がある。コマンド cd は指定したパスに移動するのみであるが、コマンド pushd は指定したパスに移動するだけでなく、ディレクトリスタックに保存する。コマンド dirs はディレクトリスタックにあるディレクトリのリストを表示する。コマンド popd はディレクトリスタックの一番上にあるパスに移動する。

4.2 処理の流れ

まず if 文の分岐によってそれぞれの処理に入る。cd はカレントディレクトリを移動する関数 getcwd を用いた。pushd はメモリ容量を確保し、スタックの一番初めに追加する。dirs はスタックの内容を構造体リストを追って出力する。出力はプッシュした逆順である。popd はスタックの一番上にあるパスを出力する。

4.3 実装方法

cd は関数 chdir によって実装した。pushd、dirs、popd は構造体リストを用いてスタックを実現した。コマンド pushd が実行されるたびに、ディレクトリ

4.4 テスト

4.4.1 テスト方法

様々な引数（ディレクトリ）を指定して、数回ずつ実行してみる。

4.4.2 テスト結果

まず cd の実行結果について以下に示す。

```
prompt : cd /.amd_mnt/home/exp/exp5/r-koyama
/.amd_mnt/home/exp/exp5/r-koyama

prompt : cd r-koyama/pro-s2016
r-koyama/pro-s2016: No such file or directory
/.amd_mnt/home/exp/exp5/r-koyama

prompt : cd pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : cd .
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : cd ..
/.amd_mnt/home/exp/exp5/r-koyama

prompt : cd pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : cd 2nd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2nd

prompt : cd ../1st
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st
```

絶対パスでも相対パスでもコマンドが実行されていることがわかる。また、存在しないディレクトリを指定するとエラーを返していることがわかる。次に pushd,dirs,popd の実行結果を以下に示す。

```
prompt : pushd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st

prompt : pushd ..
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : pushd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : pushd 2nd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2nd

prompt : pushd ../1st
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st

prompt : dirs
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2nd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st

prompt : popd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st

prompt : popd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2nd

prompt : popd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : popd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016

prompt : popd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st
```

pushd に関して、ディレクトリを指定していない場合はカレントディレクトリをスタックにプッシュし、ディレクトリを指定した場合は、そのディレクトリに移動し、スタックにプッシュしていることがわかる。dirs に関して、ディレクトリスタックのディレクトリを上から新しい順に表示していることがわかる。popd に関してディレクトリスタックの一番上にあるディレクトリに移動していることがわかる。

5　ヒストリー機能

5.1　仕様

実行したコマンドを 32 個まで保存し、コマンド history によってその内容を出力する。コマンド!!によって前回実行したコマンドを実行する。コマンド![string] によって [string] に前方一致する最新のコマンドを実行する。また [string] が数字であった場合 10 進数の int 型に変換することによって、数字に準ずる履歴を実行することもできる。

5.2　処理の流れ

まず if 文の分岐によってそれぞれの処理に入る。history は配列の内容を下に一番新しいコマンドが来るように順に出力する。!!は最新のコマンドを実行する。![string] は [string] に前方一致するコマンドを新しい方から検索し、あれば実行する。

5.3　実装方法

配列を用意し、履歴を保存していく。history は配列の中身を入っている分だけ出力する。!!は最新のコマンドをコマンドラインに移し、初めから処理を行う。![string] は文字か数字かを判断し文字であればそれに前方一致するコマンドを検索する。数字であれば、アスキーコードから数字を判断し一致する番号の履歴を探す。

5.4　テスト

5.4.1　テスト方法

他の機能のテストを行った後にこれらのコマンドを実行してみて動作を確認する。

5.4.2 テスト結果

```
prompt : history
 1 pwd
 2 ps
 3 ls -a
 4 history
 5 pushd ..
 6 pushd 1st
 7 ls
 8 emacs &
 9 ps
10 kill 14141
11 emacs groupA
12 history

prompt : !1
pwd
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/1st

prompt : !9
ps
  PID TT  STAT     TIME COMMAND
13233  1  Is+   0:00.04 -csh (csh)
13267  2  Is    0:00.05 csh
13294  2  I    16:08.68 firefox
14137  2  S+    0:00.01 ./mysh
14141  2  Z+    0:00.43 <defunct>
14153  2  R+    0:00.00 ps

prompt : !p
ps
  PID TT  STAT     TIME COMMAND
13233  1  Is+   0:00.04 -csh (csh)
13267  2  Is    0:00.05 csh
13294  2  I    16:09.00 firefox
14137  2  S+    0:00.01 ./mysh
14141  2  Z+    0:00.43 <defunct>
14154  2  R+    0:00.00 ps

prompt : !!
ps
  PID TT  STAT     TIME COMMAND
13233  1  Is+   0:00.04 -csh (csh)
13267  2  Is    0:00.05 csh
13294  2  I    16:09.00 firefox
14137  2  S+    0:00.01 ./mysh
14141  2  Z+    0:00.43 <defunct>
14154  2  R+    0:00.00 ps
```

実装されていることがわかる。!!コマンドも実現されている。

6　ワイルドカード機能

6.1　仕様

コマンドラインに “*” があれば、カレントディレクトリのディレクトリとファイル全てに置き換える。

6.2　処理の流れ

引数のポインタ配列の “*” の場所をカレントディレクトリのディレクトリとファイル全てに置き換える。

6.3　実装方法

コマンドラインを分割する段階で、“*” があれば、カレントディレクトリのディレクトリとファイル全てにスペースで区切って置き換える。

6.4　テスト

6.4.1　テスト方法

このプログラムには “*” を使用できるコマンドがないので外部コマンドを用いてテストする。

6.4.2 テスト結果

```
prompt : ls
#simple_shell.c#        mysh.c~                 simple_shell-original
mysh                    prog.txt                simple_shell-original.c
mysh.c                  simple_shell            simple_shell.c~

prompt : mv * ../2ndtemp2
mv: rename . to ../2ndtemp2/.: Invalid argument
mv: rename .. to ../2ndtemp2/..: Invalid argument

prompt : cd ../2ndtemp2
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2ndtemp2

prompt : ls
#simple_shell.c#        mysh.c~                 simple_shell-original
mysh                    prog.txt                simple_shell-original.c
mysh.c                  simple_shell            simple_shell.c~

prompt : rm *
rm: "." and ".." may not be removed

prompt : ls

prompt : exit
done.
```

ワイルドカード機能は実装されているとわかる。

7 プロンプト機能

7.1 仕様

指定した文字列にプロンプトを変更する。

7.2 処理の流れ

プロンプトの変数を用意し、常にその中にプロンプトの文字列を保有する。コマンドの入力待ちのときにプロンプトとしてこの文字列を表示する。

7.3 実装方法

“prompt” で初期化された char 型配列を用意し、このコマンドによって、第一引数の文字列に変更する。引数がない場合は、“prompt” とする。

7.4 テスト

7.4.1 テスト方法

文字列を指定して数回この機能を試してみる。

7.4.2 テスト結果

```
prompt : prompt ryo

ryo : prompt command

command : prompt

prompt :
```

引数に指定した文字列にプロンプトを変更していることがわかる。また、引数を指定しない場合は、“prompt”と変更されていることがわかる。

8 スクリプト機能

8.1 仕様

このプログラム実行時にリダイレクトで受けとったファイルから一行ずつ文字列を読み取り、コマンドとして実行する。

8.2 処理の流れ

関数 fgets で一行を読み取りコマンドラインとしてバッファに収める。その後はキーボード入力の場合と同じである。

8.3 実装方法

関数 isatty によって入力リダイレクトがあるかを判定し、あればプロンプトを表示しない。

8.4 テスト

8.4.1 テスト方法

テキストデータを示し、それを使用した結果を示す。

8.4.2　テスト結果

使用したテキストファイル

```
prog.txt

ls
pushd ..
ls
dirs
popd
pwd
```

テスト結果

```
exp175[81]% ./simple_shell < prog.txt
#simple_shell.c#        simple_shell            simple_shell.c
prog.txt                simple_shell-original   simple_shell.c~
prog.txt~               simple_shell-original.c
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
11                              sample-procreport.tex
12                              sample10-1
13                              sample10-1.c
1st                             sample10-1.c~
2nd                             sample10-2
a.out                           sample10-2.c
counter.c                       sample10-2.c~
counter.c~                      sample10-3
counter.exe                     sample10-3.c
data.txt                        sample8-3.c
ffproxy-1.6                     sample8-3.c~
ffproxy-1.6.tar.gz              sample8-3.exe
foo                             sample9-2
hoge.txt                        sample9-2_main.c
mori.txt                        sample9-2_main.c~
perl                            sample9-2_main.o
proC-3                          sample9-2_product.c
report1table.pl                 sample9-2_product.c~
sample-2-1.aux                  sample9-2_product.h
sample-2-1.dvi                  sample9-2_product.h~
sample-2-1.log                  sample9-2_product.o
sample-2-1.pdf                  unix_commands_private.txt
sample-2-1.tex                  unix_commands_shared.txt
sample-2-1.tex~                 unix_commands_shared.txt~
sample-procreport.log
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016
```

外部コマンドも内部コマンドも実行されていることがわかる。また exit コマンドがなくても正常に終了していることがわかる。

9 エイリアス機能

9.1 仕様

alias コマンドでコマンドを指定した文字列を入力しても実行されるようになる。unalias コマンドで alias コマンドで指定した文字列を解除する。

9.2 処理の流れ

alias コマンドは構造体の領域を動的に確保し、構造体に第一引数の文字列と第二引数のコマンドを保存する。unalias コマンドは第一引数で指定された文字列の構造体を探し、その構造体を削除する。

9.3 実装方法

構造体二分木を使用した。指定した文字列の辞書順になっている。削除の場合は指定した文字列を探し、その右側から一番小さいものを探し出し削除したいものと入れ替えることで削除を実現している。

9.4 テスト

9.4.1 テスト方法

いくつかのコマンドに対してこの機能を試してみる。

9.4.2 テスト結果

```
prompt : alias cdir pwd

prompt : alias h history

prompt : cdir
/.amd_mnt/home/exp/exp5/r-koyama/pro-c2016/2nd

prompt : h
 1 alias cc pwd
 2 alias h history
 3 pwd
 4 history

prompt : unalias h

prompt : h

prompt : unalias cdir

prompt : cdir
```

alias コマンドによって、文字列を第二引数のコマンドに対して設定していることがわかる。そのコマンドを打つと実際に置き換えられていることがわかる。unalias コマンドによって文字列の指定を解除していることがわかる。

10　自分で考えた機能

自分で考えた機能として cat コマンドを実装した。

10.1　仕様

第一引数で指定されたファイルの中身を出力する。

10.2　処理の流れ

第一引数で指定されたファイルを開く。その中身を一行ずつ読みとり出力する。ファイルを閉じる。

10.3　実装方法

ファイルポインタを宣言し、ファイルを操作することによって実装した。

10.4　テスト

10.4.1　テスト方法

8.4.2 章で用いたテキストファイルに対してこの機能を試してみる。

10.4.2　テスト結果

```
prompt : cat prog.txt
ls
pushd ..
ls
dirs
popd
pwd
```

ファイルの中身が表示されていることがわかる。

11　その他実装した機能

ディレクトリの管理機能のデバッグを簡単にするためにコマンド pwd を実装した。カレントディレクトリを取得し、それを出力するという方法で実装を実現した。

12　工夫点

- history 機能で![num] コマンドも実装した。
- alias コマンドのデータ構造として、二分木を使用したため、検索にかかる時間を抑えた。
- pushd コマンドでカレントディレクトリだけでなくパスを指定してスタックに保存して移動できるようにした。

13　考察

このプログラムは既存のシェルに機能の完成度が劣っている。既存のシェルにはそれぞれのコマンドにオプションを付けられる。このプログラムにも場合分けなどを使用してオプションを実装する余地がある。これからそのような機能を実装したいと思う。またプロンプト機能には現在の時刻やカレントディレクトリを表示させたりすることも改造しだいでは可能である。

14　感想

始めは自分でシェルを作ることに全く意味を見いだせなかったが、機能を実装していくうちに自分のシェルをもっとよくしていきたいという気持ちがわいてきた。13 章で述べた機能などを実装して、今のプログラムを機能がより充実したものにしたいと思う。また、子プロセス生成などの新しいコマンドは始めは理解に苦しんだが、触れることができてよかった。

A プログラムリスト

```
 1  /*
 2   *  インクルードファイル
 3   */
 4
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <sys/types.h>
 8  #include <unistd.h>
 9  #include <sys/wait.h>
10  #include <string.h>
11  #include <dirent.h>
12
13  /*
14   *  定数の定義
15   */
16
17  #define BUFLEN  1024     /* コマンド用のバッファの大きさ */
18  #define MAXARGNUM  256   /* 最大の引数の数 */
19
20  #define MAXHISTORY 32
21  #define PATHLEN 256
22  #define PROMPTLEN 32
23  #define COMMANDLEN 16
24  #define DIRLEN 64
25  #define WILDCARDNUM 64
26
27  /*
28   *      構造体の宣言
29   */
30
31  typedef struct dirs_stack{
32          char dirsname[PATHLEN];
33          struct dirs_stack *next;
34  } DIRS_STACK;
35
36  typedef struct alias_binary{
37          char alias_command[COMMANDLEN];
38          char original_command[BUFLEN];
39          struct alias_binary *left;
40          struct alias_binary *right;
41  } ALIAS_BINARY;
42
43  /*
44   *  ローカルプロトタイプ宣言
```

```
 45   */
 46
 47  int parse(char [], char *[]);
 48  void execute_command(char *[], int);
 49
 50  int execute_alias(ALIAS_BINARY *, char *, char *);
 51  void delete_alias(ALIAS_BINARY **, char *);
 52  void show_alias(ALIAS_BINARY *);
 53  void delete_binary(ALIAS_BINARY *);
 54
 55  /*--------------------------------------------------------------------------
 56   *
 57   *  関数名    : main
 58   *
 59   *  作業内容 : シェルのプロンプトを実現する
 60   *
 61   *  引数    :
 62   *
 63   *  返り値    :
 64   *
 65   *  注意    :
 66   *
 67   *-------------------------------------------------------------------------*/
 68
 69  int main(int argc, char *argv[])
 70  {
 71          char command_buffer[BUFLEN]; /* コマンド用のバッファ */
 72          char *args[MAXARGNUM];     /* 引数へのポインタの配列 */
 73          int command_status;               /* コマンドの状態を表す
 74
command_status = 0 : フォアグラウンドで実行
 75
command_status = 1 : バックグラウンドで実行
 76
command_status = 2 : シェルの終了
 77
command_status = 3 : 何もしない */
 78          char history[MAXHISTORY][BUFLEN];
 79          int history_num[MAXHISTORY] = {};
 80          int num = 0;
 81          int prev =0;
 82          char pathname[PATHLEN];
 83          char *command_buffer_p;
 84          char prompt[PROMPTLEN] = "prompt";
 85          int i;
 86          int history_x;
```

```
 87
 88          DIRS_STACK *head = NULL, *p, *access;
 89          ALIAS_BINARY *a_head = NULL, *a_p, *a_access, *a_temp;
 90          /*
 91           *  無限にループする
 92           */
 93
 94          for(;;) {
 95                  /*
 96                   *  プロンプトを表示する
 97                   */
 98
 99                  if(prev == 0){
100                          if(isatty(fileno(stdin)))
101                                  printf("\n%s : ", prompt);
102
103                  /*
104                   *  標準入力から１行を command_buffer へ読み込む
105                   *  入力が何もなければ改行を出力してプロンプト表示へ戻る
106                   */
107                          if(fgets(command_buffer, BUFLEN, stdin) == NULL) {
108
109                            if(isatty(fileno(stdin))){
110                              printf("\n");
111                                  continue;
112                            }else{
113                              exit(EXIT_SUCCESS);
114                            }
115                          }
116                          if(command_buffer[strlen(command_buffer) - 1]
== '\n'){
117                                  command_buffer[strlen(command_buffer) - 1]
= '\0';
118                          }
119                  }
120
121                  command_buffer_p = &command_buffer[0];
122
123                  while(*command_buffer_p == ' ' || *command_buffer_p == '\t') {
124                          command_buffer_p++;
125                  }
126
127                  if(strcmp(command_buffer_p, "") != 0){
128                          if(prev == 0)
129                                  for(i = 0; i<MAXHISTORY-1; i++){
130                                          strcpy(&history[i][0],
```

```
&history[i+1][0]);
   131                                          history_num[i] = history_num[i+1];
   132                                  }
   133                          strcpy(&history[MAXHISTORY-1][0], command_buffer);
   134                          history_num[MAXHISTORY-1] = ++num;
   135                  }
   136
   137                  prev = 0;
   138
   139                  /*
   140                   *  入力されたバッファ内のコマンドを解析する
   141                   *
   142                   *  返り値はコマンドの状態
   143                   */
   144
   145                  command_status = parse(command_buffer, args);
   146
   147                  /*
   148                   *  終了コマンドならばプログラムを終了
   149                   *  引数が何もなければプロンプト表示へ戻る
   150                   */
   151
   152                  if(command_status == 2) {
   153                          access = head;                                  //
スタック容量を開放
   154                          while(access != NULL){
   155                                  access = access->next;
   156                                  free(head);
   157                                  head = access;
   158                          }
   159                          delete_binary(a_head);                  //構造
体二分木容量開放
   160                          printf("done.\n");
   161                          exit(EXIT_SUCCESS);
   162                  } else if(command_status == 3) {
   163                          continue;
   164                  }
   165
   166                  /*
   167                   *  コマンドを実行する
   168                   */
   169
   170                  if(!strcmp(args[0], "history")){
   171                          for(i = 0; i<MAXHISTORY; i++)
   172                                  if(history_num[i] != 0)
   173                                          printf("%2d %s\n",history_num[i],
```

```
&history[i][0]);
   174
   175                  }else if(!strncmp(args[0], "!", 1)){
   176                          ++args[0];
   177                          if(!strcmp(args[0], "!")){
   178                                  printf("%s\n",&history[MAXHISTORY-2][0]);
   179                                  stpcpy(&command_buffer[0],
&history[MAXHISTORY-2][0]);
   180                                  --num;
   181                                  prev = 1;
   182                          }else{
   183                                  history_x = 0;
   184                                  for(i = 0; *(args[0]+i) != '\0'; i++)
   185                                          if(47 < *(args[0]+i) &&
*(args[0]+i) < 58){
   186                                                  history_x *= 10;
   187                                                  history_x +=
(int)*(args[0]+i) - 48;
   188                                          }
   189                                          else
   190                                                  break;
   191                                  if(*(args[0]+i) == '\0'){
   192                                          for(i = 0; i<MAXHISTORY &&
prev == 0; i++)
   193                                                  if(history_x ==
history_num[MAXHISTORY-1-i]){
   194                                                          printf(
"%s\n",&history[MAXHISTORY-1-i][0]);
   195                                                          strcpy(
command_buffer, &history[MAXHISTORY-1-i][0]);
   196                                                          --num;
   197                                                          prev = 1;
   198                                                  }
   199                                          if(prev == 0)
   200                                                  printf("%s: event not found\n",
   201                                  }else{
   202                                          for(i = 0; i<MAXHISTORY &&
prev == 0; i++)
   203                                                  if(!strncmp(args[0],
&history[MAXHISTORY-1-i][0], strlen(args[0]))){
   204                                                          printf("%s\n",
&history[MAXHISTORY-1-i][0]);
   205                                                          strcpy(
command_buffer, &history[MAXHISTORY-1-i][0]);
   206                                                          --num;
   207                                                          prev = 1;
```

```
   208                                                  }
   209                                          if(prev == 0)
   210                                                  printf(
"%s: event not found\n", args[0]-1);
   211                                  }
   212                          }
   213
   214                  }else if(!strcmp(args[0], "cd")){
   215                          if(chdir(args[1]))
   216                                  printf("%s: No such file or directory\n",
args[1]);
   217                          getcwd(pathname, PATHLEN);
   218                          printf("%s\n", pathname);
   219
   220                  }else if(!strcmp(args[0], "pwd")){
   221                          getcwd(pathname, PATHLEN);
   222                          printf("%s\n", pathname);
   223
   224                  }else if(!strcmp(args[0], "pushd")){
   225                          p = (DIRS_STACK*)malloc(sizeof(DIRS_STACK));
   226                          if(args[1] == NULL){
   227                                  getcwd(p->dirsname, PATHLEN);
   228                                  printf("%s\n",p->dirsname);
   229                                  p->next = head;
   230                                  head = p;
   231                          }else{
   232                                  if(chdir(args[1])){
   233                                          printf(
"%s: No such file or directory\n", args[1]);
   234                                          free(p);
   235                                  }else{
   236                                          getcwd(p->dirsname, PATHLEN);
   237                                          printf("%s\n",p->dirsname);
   238                                          p->next = head;
   239                                          head = p;
   240                                  }
   241                          }
   242
   243                  }else if(!strcmp(args[0], "dirs")){
   244                          if(head == NULL)
   245                                  printf("(null)\n");
   246                          else{
   247                                  access = head;
   248                                  while(access != NULL){
   249                                          printf("%s\n",access->dirsname);
   250                                          access = access->next;
```

```
   251                                  }
   252                          }
   253
   254                  }else if(!strcmp(args[0], "popd")){
   255                          if(head == NULL)
   256                                  printf("(null)\n");
   257                          else{
   258                                  chdir(head->dirsname);
   259                                  printf("%s\n",head->dirsname);
   260                                  p = head;
   261                                  head = p->next;
   262                                  free(p);
   263                          }
   264
   265                  }else if(!strcmp(args[0], "prompt")){
   266                          if(args[1] == NULL)
   267                                  strcpy(prompt, "prompt");
   268                          else
   269                                  stpcpy(prompt, args[1]);
   270
   271                  }else if(!strcmp(args[0], "alias")){
   272                          if(args[1] == NULL){
   273                                  show_alias(a_head);
   274                          }
   275                          else{
   276                                  a_access = a_head;
   277                                  while(a_access != NULL){
   278                                          a_temp = a_access;
   279                                          if(strcmp(a_access->alias_command,
args[1]) > 0)
   280                                                  a_access = a_access->left;
   281                                          else if(strcmp(a_access->alias_command,
args[1]) < 0)
   282                                                  a_access = a_access->right;
   283                                          else{
   284                                                  strcpy(
a_access->original_command, args[2]);
   285                                                  i = 3;
   286                                                  while(args[i] != NULL){
   287                                                          strcat(
a_access->original_command, " ");
   288                                                          strcat(
a_access->original_command, args[i]);
   289                                                          ++i;
   290                                                  }
   291                                                  break;
```

```
292                                             }
293                                     }
294                                     if(a_access == NULL){
295                                             a_p =
(ALIAS_BINARY *)malloc(sizeof(ALIAS_BINARY));
296                                             strcpy(a_p->alias_command, args[1]);
297                                             strcpy(a_p->original_command, args[2]);
298                                             i = 3;
299                                             while(args[i] != NULL){
300                                                     strcat(
a_access->original_command, " ");
301                                                     strcat(
a_access->original_command, args[i]);
302                                                     ++i;
303                                             }
304                                             a_p->left = NULL;
305                                             a_p->right = NULL;
306                                             if(a_head == NULL)
307                                                     a_head = a_p;
308                                             else{
309                                                     if(strcmp(
a_temp->alias_command, args[1]) > 0)
310                                                             a_temp->left = a_p;
311                                                     else if(strcmp(
a_temp->alias_command, args[1]) < 0)
312                                                             a_temp->right = a_p;
313                                             }
314                                     }
315                             }
316                     }else if(!strcmp(args[0], "unalias")){
317                             delete_alias(&a_head, args[1]);
318                     }else{
319                             if(execute_alias(a_head, args[0], command_buffer)){
320                                     --num;
321                                     prev = 1;
322                             }else
323                                     execute_command(args, command_status);
324                     }
325             }
326
327             return 0;
328     }
329
330     int execute_alias(ALIAS_BINARY *now, char *command, char *command_buffer)
331     {
332             ALIAS_BINARY *access;
```

```
333
334         access = now;
335         while(access != NULL){
336                 if(strcmp(access->alias_command, command) > 0){
337                         access = access->left;
338                 }else if(strcmp(access->alias_command, command) < 0){
339                         access = access->right;
340                 }else{
341                         strcpy(command_buffer, access->original_command);
342                         return(1);
343                 }
344         }
345                 return(0);
346 }
347
348 void delete_alias(ALIAS_BINARY **head, char *command)
349 {
350         ALIAS_BINARY *access1, *access2, *temp1, *temp2 = NULL;
351
352         access1 = *head;
353
354         while(access1 != NULL){
355                 if(strcmp(access1->alias_command, command) > 0){
356                         temp1 = access1;
357                         access1 = access1->left;
358                 }
359                 else if(strcmp(access1->alias_command, command) < 0){
360                         temp1 = access1;
361                         access1 = access1->right;
362                 }
363                 else
364                         break;
365         }
366
367         if(access1 == NULL)
368                 return;
369
370         if(access1->right == NULL){
371                 if(access1 == *head)
372                         *head = access1->left;
373                 else{
374                         if(strcmp(temp1->alias_command, command) > 0)
375                                 temp1->left = access1->left;
376                         else if (strcmp(temp1->alias_command, command) < 0)
377                                 temp1->right = access1->left;
378                 }
```

```
379                     free(access1);
380             }else{
381                     access2 = access1->right;
382                     while(access2->left != NULL){
383                             temp2 = access2;
384                             access2 = access2->left;
385                     }
386                     if(temp2 != NULL)
387                             temp2->left = access2->right;
388                     access2->left = access1->left;
389                     access2->right = access1->right;
390                     if(access1 == *head)
391                             *head = access2;
392                     else{
393                             if(strcmp(temp1->alias_command, command) > 0)
394                                     temp1->left = access2;
395                             else if (strcmp(temp1->alias_command, command) < 0)
396                                     temp1->right = access2;
397                     }
398                     free(access1);
399             }
400     }
401
402     void show_alias(ALIAS_BINARY *now)
403     {
404             if(now->left != NULL)
405                     show_alias(now->left);
406             printf("%s: %s\n",now->alias_command,now->original_command);
407             if(now->right != NULL)
408                     show_alias(now->right);
409     }
410
411     void delete_binary(ALIAS_BINARY *now)
412     {
413             if(now != NULL){
414                     delete_binary(now->left);
415                     delete_binary(now->right);
416                     free(now);
417             }
418     }
419
420     /*-------------------------------------------------------------------------
421      *
422      *  関数名    : parse
423      *
424      *  作業内容 : バッファ内のコマンドと引数を解析する
```

```
425   *
426   *  引数    :
427   *
428   *  返り値    : コマンドの状態を表す :
429   *                                0 : フォアグラウンドで実行
430   *                                1 : バックグラウンドで実行
431   *                                2 : シェルの終了
432   *                                3 : 何もしない
433   *
434   *  注意    :
435   *
436   *-------------------------------------------------------------------------*/
437
438  int parse(char buffer[],                /* バッファ */
439                    char *args[])                 /* 引数へのポインタ配列 */
440  {
441          int arg_index;   /* 引数用のインデックス */
442          int status;   /* コマンドの状態を表す */
443
444          char *args_temp[COMMANDLEN];
445          char wild_card[WILDCARDNUM][DIRLEN];
446
447          DIR *dir;
448          struct dirent *dp;
449          /*
450           *  変数の初期化
451           */
452
453          arg_index = 0;
454          status = 0;
455
456          /*
457           *  バッファが終了を表すコマンド（"exit"）ならば
458           *  コマンドの状態を表す返り値を 2 に設定してリターンする
459           */
460
461          if(strcmp(buffer, "exit") == 0) {
462                  status = 2;
463                  return status;
464          }
465
466          /*
467           *  バッファ内の文字がなくなるまで繰り返す
468           *  （ヌル文字が出てくるまで繰り返す）
469           */
470
```

```
471         while(*buffer != '\0') {
472
473                 /*
474                  *  空白類（空白とタブ）をヌル文字に置き換える
475                  *  これによってバッファ内の各引数が分割される
476                  */
477
478                 while(*buffer == ' ' || *buffer == '\t') {
479                         *(buffer++) = '\0';
480                 }
481
482                 /*
483                  * 空白の後が終端文字であればループを抜ける
484                  */
485
486                 if(*buffer == '\0') {
487                         break;
488                 }
489
490                 /*
491                  *  空白部分は読み飛ばされたはず
492                  *  buffer は現在は arg_index + 1 個めの引数の先頭を指している
493                  *
494                  *  引数の先頭へのポインタを引数へのポインタ配列に格納する
495                  */
496
497                 args[arg_index] = buffer;
498                 ++arg_index;
499
500                 /*
501                  *  引数部分を読み飛ばす
502                  *  （ヌル文字でも空白類でもない場合に読み進める）
503                  */
504
505                 while((*buffer != '\0') && (*buffer != ' ') && (*buffer != '\t')) {
506                         ++buffer;
507                 }
508         }
509
510         /*
511          *  最後の引数の次にはヌルへのポインタを格納する
512          */
513
514         args[arg_index] = NULL;
```

```
515
516         /*
517          *  最後の引数をチェックして "&" ならば
518          *
519          *  "&" を引数から削る
520          *  コマンドの状態を表す status に 1 を設定する
521          *
522          *  そうでなければ status に 0 を設定する
523          */
524
525         if(arg_index > 0 && strcmp(args[arg_index - 1], "&") == 0) {
526                 --arg_index;
527                 args[arg_index] = '\0';
528                 status = 1;
529         } else {
530                 status = 0;
531         }
532
533         /*
534          *  引数が何もなかった場合
535          */
536
537         if(arg_index == 0) {
538                 status = 3;
539         }
540
541         if(arg_index > 0){                              //ワイルドカー
ド機能
542
543                 int i=1;
544                 while(args[i] != NULL && strcmp(args[i], "*") != 0){
545                 i++;
546                 }
547                 if(args[i] != NULL){
548                         int j=0;
549                         while(args[i+j] != NULL){
550                                 args_temp[j] = args[i+1+j];
551                                 j++;
552                         }
553                         j = 0;
554                         arg_index = i;
555                         dir = opendir(".");
556                         while ((dp = readdir(dir)) != NULL) {
557                                 strcpy(&wild_card[j][0], dp->d_name);
558                                 args[arg_index] = &wild_card[j][0];
559                                 arg_index++;
```

```
560                                     j++;
561                             }
562                             j = 0;
563                             while(args_temp[j] != NULL){
564                                     args[arg_index] = args_temp[j];
565                                     arg_index++;
566                                     j++;
567                             }
568                             args[arg_index] = NULL;
569                             closedir(dir);
570                     }
571             }
572
573             /*
574              *  コマンドの状態を返す
575              */
576
577             return status;
578     }
579
580     /*------------------------------------------------------------------------
581      *
582      *  関数名    : execute_command
583      *
584      *  作業内容 : 引数として与えられたコマンドを実行する
585      *                          コマンドの状態がフォアグラウンドならば、コマンドを
586      *                          実行している子プロセスの終了を待つ
587      *                          バックグラウンドならば子プロセスの終了を待たずに
588      *                          main 関数に返る（プロンプト表示に戻る）
589      *
590      *  引数    :
591      *
592      *  返り値    :
593      *
594      *  注意    :
595      *
596      *------------------------------------------------------------------------*/
597
598     void execute_command(char *args[],      /* 引数の配列 */
599                                             int command_status)     /* コ
マンドの状態 */
600     {
601             int pid;          /* プロセスＩＤ */
602             int status;   /* 子プロセスの終了ステータス */
603             int wait;
604
```

```
605          /*
606           *  子プロセスを生成する
607           *
608           *  生成できたかを確認し、失敗ならばプログラムを終了する
609           */
610
611          /******** Your Program ********/
612          pid = fork();
613          if (pid < 0){
614                  printf("fork failure\n");
615                  exit(EXIT_FAILURE);
616          }
617
618          /*
619           *  子プロセスの場合には引数として与えられたものを実行する
620           *
621           *  引数の配列は以下を仮定している
622           *  ・第１引数には実行されるプログラムを示す文字列が格納されている
623           *  ・引数の配列はヌルポインタで終了している
624           */
625
626          /******** Your Program ********/
627          else if(pid == 0){
628                  execvp(args[0],args);
629                  exit(EXIT_SUCCESS);
630          }
631
632          /*
633           *  コマンドの状態がバックグラウンドなら関数を抜ける
634           */
635
636          /******** Your Program ********/
637          else{
638                  if(command_status == 1)
639                          return;
640
641          /*
642           *  ここにくるのはコマンドの状態がフォアグラウンドの場合
643           *
644           *  親プロセスの場合に子プロセスの終了を待つ
645           */
646
647          /******** Your Program ********/
648                  int wait = waitpid(pid, &status, 0); //子プロセスのプロ
セス ID を指定して、終了を待つ
649                  if(wait < 0){
```

```
650                         printf("wait failure\n");
651                         exit(EXIT_FAILURE);
652                 }
653         }
654
655         return;
656 }
657 /*-- END OF FILE -------------------------------------------------------------*/
```