Challenge 1

OverTheWire : Bandit

Bandit 00 Solution:

To access the first level, connect using SSH:

```
$ ssh bandit0@bandit.labs.overthewire.org -p 2220
```

Read the content of the readme file:

```
$ cat readme
```

```
NH2SXQwcBdpmTEzi3bvBHMM9H66vVXjL
```

Bandit 01 Solution: Connect to the server:

```
$ ssh bandit1@bandit.labs.overthewire.org -p 2220
```

Read the file named -:

```
$ cat ./-            // specify file pathname
```

```
rRGizSaX8Mk1RTb1CNQoXTcYZWU6lgzi
```

Bandit 02 Solution: SSH into the server:

```
$ ssh bandit2@bandit.labs.overthewire.org -p 2220
```

Read the file named spaces in this filename:

```
$ cat "spaces in this filename"
```

```
aBZ0W5EmUfAf7kHTQeOwd8bauFJ2lAiG
```

Bandit 03 Solution: Connect to the server:

```
$ ssh bandit3@bandit.labs.overthewire.org -p 2220
```

Access the inhere directory and read the .hidden file:

```
$ cd inhere/

$ ls -a    // -a will reveal hidden files

$ cat .hidden

2EW7BBsr6aMMoJ2HjW067dm8EgX26xNe
```

Bandit 04 Solution: SSH into the server:

```
$ ssh bandit4@bandit.labs.overthewire.org -p 2220
```

Navigate to the inhere directory and find the human-readable file:

```
$ cd inhere/

$ ls -la        // A lot of files

$ file ./-file0*  // will say ASCII text

$ cat ./-file07

lrIWWI6bB37kxfiCQZqUdOIYfr6eEeq
```

Bandit 05 Solution: Connect to the server:

```
$ ssh bandit5@bandit.labs.overthewire.org -p 2220
```

Use find command to locate the desired file based on given properties:

// Look for useful find commands on google - https://www.tecmint.com/35-practical-examples-of-linux-find-command/?source=post_page-----aa853b431c1d----------------------------------

```
$ find ./inhere/ -type f -readable ! -executable -size 1033c
```

```
$ cat /home/bandit5/inhere/maybehere07/.file2

P4L4vucdmLnm8I7Vl7jG1ApGSfjYKqJU
```

Bandit 06 Solution: SSH into the server:

```
$ ssh bandit6@bandit.labs.overthewire.org -p 2220
```

Utilize `find` command to locate the file with specified properties:

```
// Again look for find commands

$ find / -type f -size 33c -group bandit6 -user bandit7 2>/dev/null

// There will be a lot of ‘permission denied’ files. /dev/null will
nullify those permissions

$ cat /var/lib/dpkg/info/bandit7.password

z7WtoNQU2XfjmMtWA8u5rN4vzqu4v99S
```

Bandit 07 Solution: Connect to the server:

```
$ ssh bandit7@bandit.labs.overthewire.org -p 2220

// The password is next to the word millionth

$ grep millionth data.txt

TESKZC0XvTetK0S9xNwm25STk5iWrBvP
```

Bandit 08 Solution: SSH into the server:

```
$ ssh bandit8@bandit.labs.overthewire.org -p 2220
```

Sort the file, count unique lines, and filter out the line occurring only once:

```
$ sort data.txt | uniq -u  // -u will get only unique values

EN632PlfYiZbn3PhVK3XOGSlNInNE00t
```

Bandit 09 Solution:

1. Connect to the server using SSH.
2. Use the `strings` command to extract human-readable strings from `data.txt`.
3. Filter the strings beginning with several `=` characters using `grep`.

```
$ ssh bandit9@bandit.labs.overthewire.org -p 2220

$ strings data.txt | grep "^=="    // G7w8LIi6J3kTb8A7j9LgrywtEUlyyp6s
```

Bandit 10 Solution:

1. SSH into the server.
2. Read the contents of `data.txt`.
3. Decode the base64 encoded data using `base64 -d`.

```
$ ssh bandit10@bandit.labs.overthewire.org -p 2220

$ cat data.txt | base64 -d

6zPeziLdR2RKNdNYFNb6nVCKzphlXHBM
```

Bandit 11 Solution:

1. Log in to the server.
2. Read the contents of `data.txt`.
3. Use `tr` command to rotate lowercase and uppercase letters by 13 positions or ROT13 decoder available online can also be used.

```
$ ssh bandit11@bandit.labs.overthewire.org -p 2220

$ cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'
```

The password is revealed as `JVNBBFSmZwKKOP0XbFXOoW8chDz5yVRv`

Bandit 12 Solution:

1. SSH into the server.
2. Create a temporary working folder.
3. Copy data.txt to the working folder.
4. Convert the hexdump to binary using xxd.
5. Identify the compression algorithm using file command.
6. Decompress the file accordingly.
7. Repeat the process until the password is revealed.

```
$ ssh bandit12@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/ax

$ cp data.txt /tmp/ax

$ cd /tmp/ax

$ xxd -r data.txt data.out

$ file data.out           // data.out will turn out to be a gzip file

$ mv data.out data.gz     // need to move to decompression

$ gzip -d data.gz

$ file data               // data will be a bzip2 file

$ bzip2 -d data           // no need to move

$ file data.out

$ mv data.out data.gz

$ gzip -d data.gz

$ file data

$ tar -xf data

$ file data5.bin

$ tar -xf data5.bin
```

```
$ file data6.bin

$ bzip2 -d data6.bin

$ file data6.bin.out

$ tar -xf data6.bin.out

$ file data8.bin

$ mv data8.bin data8.gz

$ gzip -d data8.gz

$ file data8

$ cat data8
```

The password is `wbWdlBxEir4CaE8LaPhauuOo6pwRmrDw.`

Bandit 13 Solution:

1. SSH into the server.
2. List files in the home directory to find `sshkey.private`.
3. Exit the server and download `sshkey.private` to local machine using `scp`.
4. Change permissions of the private key.
5. SSH into the next level using the private key.

```
$ ssh bandit13@bandit.labs.overthewire.org -p 2220

$ ls -la

$ exit

$ scp -P 2220 bandit13@bandit.labs.overthewire.org:sshkey.private .

$ chmod 400 sshkey.private

$ ssh -i sshkey.private bandit14@bandit.labs.overthewire.org -p 2220
```

Bandit 14 Solution:

1. SSH into the server using the private key obtained in the previous level.
2. Read the password of the next level.
3. Use `nc` command to submit the current level's password to port 30000 on localhost.

```
$ ssh -i sshkey.private bandit14@bandit.labs.overthewire.org -p 2220
```

```
$ cat /etc/bandit_pass/bandit14 | nc localhost 30000
```

The correct response confirms the password as `fGrHPx402xGC7U7rXKDaxiWFTOiF0ENq`.

Bandit 15 Solution:

1. SSH into the server.
2. Read the password of the next level.
3. Use `openssl` to submit the current level's password to port 30001 on localhost with SSL encryption.

```
$ ssh bandit15@bandit.labs.overthewire.org -p 2220
```

```
$ cat /etc/bandit_pass/bandit15 | openssl s_client -connect
localhost:30001 -quiet
```

```
JQttfApK4SeyHwDlI9SXGR50qclOAil1
```

Bandit 16 Solution:

1. Connect to the server using SSH.
2. Use a loop to scan ports from 31000 to 32000 to find which ports have a server listening.
3. Use OpenSSL to check if the server on each port speaks SSL.
4. Identify the port that gives back the next credentials.

```
$ ssh bandit16@bandit.labs.overthewire.org -p 2220
```

```
$ for i in {31000..32000} ; do
```

```
>  SERVER="localhost"

>  PORT=$i

>  (echo  > /dev/tcp/$SERVER/$PORT) >& /dev/null &&

>  echo "Port $PORT open"

> done
```

```
$ cat /etc/bandit_pass/bandit16 | openssl s_client -connect
localhost:PORT -quiet
```

The password is transmitted back - VwOSWtCA7lRKkTfbr2IDh6awj9RNZM5e.

Bandit 17 Solution:

1. SSH into the server.
2. Use the diff command to compare the contents of passwords.old and passwords.new.
3. Identify the line that has been changed.

```
$ ssh -i sshkey bandit17@bandit.labs.overthewire.org -p 2220
```

```
$ diff passwords.old passwords.new
```

The changed line reveals the password hga5tuuCLF6fFzUpnagiMN8ssu9LFrdg

Bandit 18 Solution:

1. SSH into the server.
2. Use the command directly in SSH to read the contents of readme.

```
$ ssh bandit18@bandit.labs.overthewire.org -p 2220 "cat readme"
```

The password is displayed awhqfNnAbc1naukrpqDYcF95h7HoMTrC

Bandit 19 Solution:

1. Connect to the server using SSH.
2. Execute the setuid binary without arguments to understand its functionality.
3. Use the binary to read the password from `/etc/bandit_pass/bandit20`.

```
$ ssh bandit19@bandit.labs.overthewire.org -p 2220
```

```
$ ./bandit20-do
```

```
$ ./bandit20-do cat /etc/bandit_pass/bandit20
```

The password for the next level is revealed
`VxCazJaVykI6W36BkBU0mJTCM8rR95XT`

Bandit 20 Solution:

1. SSH into the server.
2. Start a listener on a specific port using `nc`.
3. Run the `suconnect` binary with the port as an argument.
4. The password will be transmitted through the listener.

```
# Terminal 1
```

```
$ nc -lp 31337 < /etc/bandit_pass/bandit20
```

```
# Terminal 2
```

```
$ ./suconnect 31337
```

The password is received in Terminal 1
`NvEJF7oVjkddltPSrdKEFOllh9V1IBcq`

Bandit 21 Solution:

1. Log in to the server.
2. Check the `/etc/cron.d/` directory for cron job configurations.
3. Identify the script being executed by the cron job.
4. Read the script to find out where the password is stored.

```
$ ssh bandit21@bandit.labs.overthewire.org -p 2220
```

```
$ ls -la /etc/cron.d/
```

```
$ cat /etc/cron.d/cronjob_bandit22

$ cat /usr/bin/cronjob_bandit22.sh
```

The password is stored as described in the script `WdDozAdTM2z9DiFEQ2mGlwngMfj4EZff`.

Bandit 22 Solution:

1. SSH into the server.
2. Look in `/etc/cron.d/` for the cron job configuration.
3. Check the script executed by the cron job to understand how the password is stored.

```
$ ssh bandit22@bandit.labs.overthewire.org -p 2220

$ ls -la /etc/cron.d/

$ cat /etc/cron.d/cronjob_bandit23

$ cat /usr/bin/cronjob_bandit23.sh

$ echo "I am user bandit23" | md5sum

$ cat /tmp/<hash>
```

Retrieve the password from the hashed file `QYw0Y2aiA672PsMmh9puTQuhoz8SyR2G`

Bandit 23 Solution:

1. SSH into the server.
2. Check the `/etc/cron.d/` directory for the cron job configuration.
3. Identify the script being executed by the cron job.
4. Create a script that copies the password to a location you have access to, such as `/tmp`.
5. Make the script executable and copy it to the directory where the cron job executes.

```
$ ssh bandit23@bandit.labs.overthewire.org -p 2220

$ ls -la /etc/cron.d/
```

```
$ cat /etc/cron.d/cronjob_bandit24

$ cat /usr/bin/cronjob_bandit24.sh

$ mkdir /tmp/alex1234

$ cd /tmp/alex1234

$ vi script.sh

# Enter the script contents to copy the password

$ chmod 777 script.sh

$ cp script.sh /var/spool/bandit24

$ chmod 777 /tmp/alex1234/
```

The password will be copied to `/tmp/alex1234/bandit24pass`

```
VAfGXJ1PBSsPSnvsjI8p759leLZ9GGar
```

Bandit 24 Solution:

1. SSH into the server.
2. Use the for loop for brute force.

```
$ for i in {0000..9999}; do echo “VAfGXJ1PBSsPSnvsjI8p759leLZ9GGar
$i”; done | nc localhost 30002

p7TaowMYrmu23Ol8hiZh9UvD0O9hpx8d
```

Bandit 25 & 26 Solution:

1. SSH into the server.
2. Identify the shell being used by bandit26.
3. Understand how the shell works and how to break out of it.
4. Use VI editor to execute commands and retrieve the password for bandit27.

```
$ ssh bandit25@bandit.labs.overthewire.org -p 2220

$ cat /usr/bin/showtext
```

```
$ ls

$ ssh -i bandit26.sshkey bandit26@localhost

# Follow the steps to break out of the shell and retrieve the password
for bandit27
```

The password for bandit27 will be displayed
c7GvcKlw9mC7aUQaPx7nwFstuAIBw1o1

Bandit 27 Solution:

1. SSH into the server.
2. Clone the git repository.
3. Retrieve the password from the README file.

```
$ ssh bandit27@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/repo123

$ cd /tmp/repo123

$ git clone ssh://bandit27-git@localhost/home/bandit27-git/repo.git/

$ cat repo/README
```

The password for bandit28 will be displayed
YnQpBuifNMas1hcUFk70ZmqkhUU2EuaS

Bandit 28 Solution:

1. SSH into the server.
2. Clone the git repository.
3. Checkout an older commit to reveal the password.

```
$ ssh bandit28@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/repo1337

$ cd /tmp/repo1337
```

```
$ git clone ssh://bandit28-git@localhost/home/bandit28-git/repo

$ cd repo

$ git log

$ git checkout <commit-hash>

$ cat README.md
```

The password revealed AVanL161y9rsbcJIsFHuw35rjaOM19nR

Bandit 29 Solution:

1. SSH into the server.
2. Clone the git repository.
3. Checkout the dev branch to reveal the password.

```
$ ssh bandit29@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/plop123

$ cd /tmp/plop123

$ git clone ssh://bandit29-git@localhost/home/bandit29-git/repo

$ cd repo

$ cat README.md

$ git branch -r

$ git checkout dev

$ cat README.md
```

The password revealed tQKvmcwNYcFS6vmPHIUSI3ShmsrQZK8S.

Bandit 30 Solution:

1. SSH into the server.
2. Clone the git repository.
3. Retrieve the password using the `git show` command with the tag.

```
$ ssh bandit30@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/plop1234

$ cd /tmp/plop1234

$ git clone ssh://bandit30-git@localhost/home/bandit30-git/repo

$ cd repo

$ git tag

$ git show secret
```

The password revealed `xbhV3HpNGlTIdnjUrdAlPzc2L6y9EOnS`.

Bandit 31 Solution:

1. SSH into the server.
2. Clone the git repository.
3. Follow the instructions in the README.md to push a file to the remote repository.
4. Retrieve the password from the response after pushing the file.

```
$ ssh bandit31@bandit.labs.overthewire.org -p 2220

$ mkdir /tmp/plop12345

$ cd /tmp/plop12345

$ git clone ssh://bandit31-git@localhost/home/bandit31-git/repo

$ cd repo

$ echo "May I come in?" > key.txt

$ git add -f key.txt
```

```
$ git commit -m key.txt

$ git push origin master
```

The password revealed is `OoffzGDlzhAlerFJ2cAiz1D41JW1Mhmt`

Bandit 32 Solution:

1. SSH into the server.
2. Follow the instructions in the welcome message to get an interactive shell.
3. Use Vim to read the password for the next level.

```
$ ssh bandit32@bandit.labs.overthewire.org -p 2220

$ $0

$ vim

# In Vim, enter the command:

# :r /etc/bandit_pass/bandit33
```

The password revealed is `rmCBvG56y58BXzv98yZGdO7ATVL5dW8y.`

Bandit 33 Solution (The End):

1. SSH into the server.
2. Check the README.txt file for the congratulatory message.

```
$ ssh bandit33@bandit.labs.overthewire.org -p 2220

$ ls

$ cat README.txt
```

Challenge 2.1

Russian Roulette game

Python

```python
import random

def russian_roulette():
    chambers = [False, False, False, False, False, True]
    random.shuffle(chambers)

    def take_turn(player):
        if chambers.pop():
            print(f"{player} pulls the trigger... BANG! {player} dies!")
            return True
        else:
            print(f"{player} pulls the trigger.. Click. {player} survives.")
            return False

    print("Welcome to the terrific game of Russian Roulette!")
    print("Toh maar diya jaye ki chor diya jaaye")
    player = input("Would care for giving your name?\n")

    while True:

        if take_turn(player):
            break
        print("Seem to have a good luck!")
        if take_turn("Boss"):
            break
    print("Game over!")
russian_roulette()
```

Challenge 2.2

Selection Sort Algorithm

C/C++

```
#include<stdio.h>
void SelectionSort(int n, int* arr){
int iOM;
int i;
int temp,flag=0;
for(i=0; i<n-1; i++){
   iOM = i;
   for(int j= i+1; j<n; j++){
    if(arr[iOM]>arr[j]){
     iOM=j; }}
   flag++;
   temp = arr[i];
       arr[i]= arr[iOM];
       arr[iOM]= temp;  }
printf("\nRunning Selection Sort...\n");}
int PrintArray(int n, int *arr){
    for(int i=0; i<n; i++){
       printf("%d ",arr[i]);}
}
int main(){
   int n;
   printf("Enter the size of the array: \n");
   scanf("%d",&n);
   printf("Enter the elements of the array: \n");
   int arr[n];
   for (int i=0 ; i<n; i++){
       scanf("%d",&arr[i]);}
SelectionSort(n,arr);
PrintArray(n,arr);
return 0; }
```

Challenge 2.3

a. Single user TO-DO list (C language)

C/C++

```
#include<stdio.h>
#include<string.h>
#define MAX_TASK_LENGTH 100
#define FILENAME "todo.txt"

void create_new_todo_list();
void add_todo_item();
void view_todo_list();
int main()
{
   printf("Hey there! this is your to-do list organizer.\n");
   int choice;
   printf("\nWhat would you like to do?\n");
   printf("1. Add a to-do item\n");
   printf("2. Create a new to-do list\n");
   printf("3. View your to-do list\n");
   printf("4. Exit\n");
   printf("Enter your choice: ");
   scanf("%d",&choice);

   switch(choice){
   case 1:
    add_todo_item();
   break;
   case 2:
   create_new_todo_list();
   break;
   case 3:
   view_todo_list();
```

```
   break;
   case 4:
   printf("Goodbye! See you again.");
   break;
   default:
   printf("Invalid choice. Please try again.\n");
}}


void create_new_todo_list(){
   FILE *file = fopen(FILENAME, "w");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
   char task[MAX_TASK_LENGTH];
   printf("Add your tasks. Enter 'OK' when done.\n");
   getchar();
   int i = 1;
   while (1) {
       printf("%d. ", i);
       fgets(task, MAX_TASK_LENGTH, stdin);
       task[strcspn(task, "\n")] = '\0';

       if (strcmp(task, "OK") == 0) {
           break;
       }

       fprintf(file, "%s\n", task);
       i++;
   }
   fclose(file);
   printf("New to-do list created successfully!\n");
}
```

```
   void add_todo_item(){
       FILE *file = fopen(FILENAME, "a");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
   char task[MAX_TASK_LENGTH];
   printf("Enter your to-do item: ");

   getchar();
   fgets(task, MAX_TASK_LENGTH, stdin);
   task[strcspn(task, "\n")] = '\0';
   fprintf(file, "%s\n", task);
   fclose(file);
   printf("To-do item added successfully!\n");

   }

void view_todo_list(){
   FILE *file = fopen(FILENAME, "r");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
char task[MAX_TASK_LENGTH];
printf("\nYour to-do list:\n");
   while (fgets(task, MAX_TASK_LENGTH, file)) {
       printf("- %s", task);
   }

   fclose(file);
   printf("\n"); }
```

b. Multiple user single file TO-DO list

1. C language

C/C++

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX_TASK_LENGTH 100
#define FILENAME "todo.txt"
#define NAME 20
void create_new_todo_list(const char *name);
void add_todo_item(const char *name);
void view_todo_list(const char *name);
int main()
{
   printf("Hey there! this is your to-do list organizer.\n");
   int choice;
   char name[NAME];
   printf("Tell me your name: ");
   fgets(name,NAME, stdin);
   name[strcspn(name, "\n")] = '\0';
   create_new_todo_list(name);
   printf("\nHey %s!",name);
   while(1){
   printf("\nWhat would you like to do?\n");
   printf("1. Add a to-do item\n");
   printf("2. View your to-do list\n");
   printf("3. Exit\n");
   printf("Enter your choice: ");
   scanf("%d",&choice);

   switch(choice){
   case 1:
    add_todo_item(name);
```

```
   break;
   case 2:
   view_todo_list(name);
   break;
   case 3:
    printf("Goodbye! See you again.");
    exit(0);
   break;
   default:
   printf("Invalid choice. Please try again.\n");

}}
}

void create_new_todo_list(const char *name ){
   FILE *file = fopen(FILENAME, "a");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
   fprintf(file, "%s's list: \n", name);
   fclose(file);
}

   void add_todo_item(const char *name){
       FILE *file = fopen(FILENAME, "a");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
   char task[MAX_TASK_LENGTH];
   printf("Add your tasks. Press 'OK' when done. \n");
   getchar();
   while(1){
```

```
   fgets(task, MAX_TASK_LENGTH, stdin);
   task[strcspn(task, "\n")] = '\0';
   if(strcmp(task,"OK")== 0){
       break;
   }
   fprintf(file, "- %s\n", task);
   }
   fclose(file);
   printf("To-do item added successfully!\n");

   }

void view_todo_list(const char *name){
   FILE *file = fopen(FILENAME, "r");
   if (file == NULL) {
       perror("Could not open file");
       return;
   }
char line[MAX_TASK_LENGTH];
fgets(line, MAX_TASK_LENGTH, file);
while (fgets(line, MAX_TASK_LENGTH, file)) {
       printf("%s", line);
   }

   fclose(file);
   printf("\n");
}
```

2. Python

Python

```python
import sys
FILENAME = "to_do.txt"

def create_new_todo_list(user_name):
    with open(FILENAME, "a") as file:
        file.write(f"{user_name}'s TO-DO List: \n")
    print("Add your tasks. Enter 'OK' when done.")
    while True:
        task = input(f"{len(open(FILENAME).readlines())}. ")
        if task.casefold() == 'ok':
            break
        with open(FILENAME,"a") as file:
            file.write(f"- {task}\n")
    print("New to-do list created successfully!")


def add_todo_item(user_name):
    task = input("Enter your to-do item: ")
    with open(FILENAME, "a") as file:

        file.write(f"{user_name}: {task}\n")
    print("TO-DO item added successfully!")

def view_todo_item(user_name):
    with open(FILENAME, "r") as file:
        tasks = file.readlines()
    for task in tasks:
        if task.startswith(user_name):
```

```
            print(task.strip())
    print("\n")


def main():
    print("Hey there! this is your to-do list organizer.\n")
    user_name = input("Tell me your name: ")
    while True:
        print(f"\nHey {user_name}!")
        print("\nWhat would you like to do?\n")
        print("1. Create a new to-do list\n")
        print("2. Add a to-do item\n")
        print("3. View your to-do list\n")
        print("4. Exit\n")
        choice  = int(input("Enter your choice: "))

        if choice == 1:
            create_new_todo_list(user_name)
        elif choice == 2:
            add_todo_item(user_name)
        elif choice == 3:
            view_todo_item(user_name)
        elif choice == 4:
            print("Goodbye! See you again.")
            sys.exit()
        else:
            print("Invalid choice. Please try again.\n")

if __name__ == "__main__":
    main()
```

Challenge 2.4

Percussion programming language

```
Unset
Bang “Hello World!”                          // print statement
Tan “Why so serious?”                       // prompt user for input
Sing a                                      // Declare variable ‘a’
Sing b                                      // Declare variable ‘b’
Rock a to b                                 // calculate a + b
Drum a on b                                 // calculate a - b
Pluck a with b                              // calculate a*b
Strum a over b                              // calculate a/b
Chord a and b                               // concatenate ‘a’ and ‘b’
Tune a to b                                  // Assignment : a = b
Beat a times                                 // Loop ‘a’ times
Jam                                          // Pause or return
```

l05t programming language

```
Unset
Lonely A happy                              // positive variable declaration ‘A’
Lonely B sad                                // negative variable declaration ‘B’
Together A B happy                                   // calculate A + B
Together A B sad                                     // calculate A - B
Together A B bittersweet                             // calculate A * B
Together A B fighting                                // calculate A / B
Going A forever                                      // while (1)
A B Imperfect                    // returns No if ( A == B) and Yes if ( A!= B)
Left A Right B                                      // Assignment : A = B
Tata A                                               // Decrement : A–
Titi B                                              // Increment : B++
Found Lost “Hey Beautiful!”                         // print statement
Got Lost  “Give a pet name!”                        // prompt user for input
Domino                                               // Program exits
```