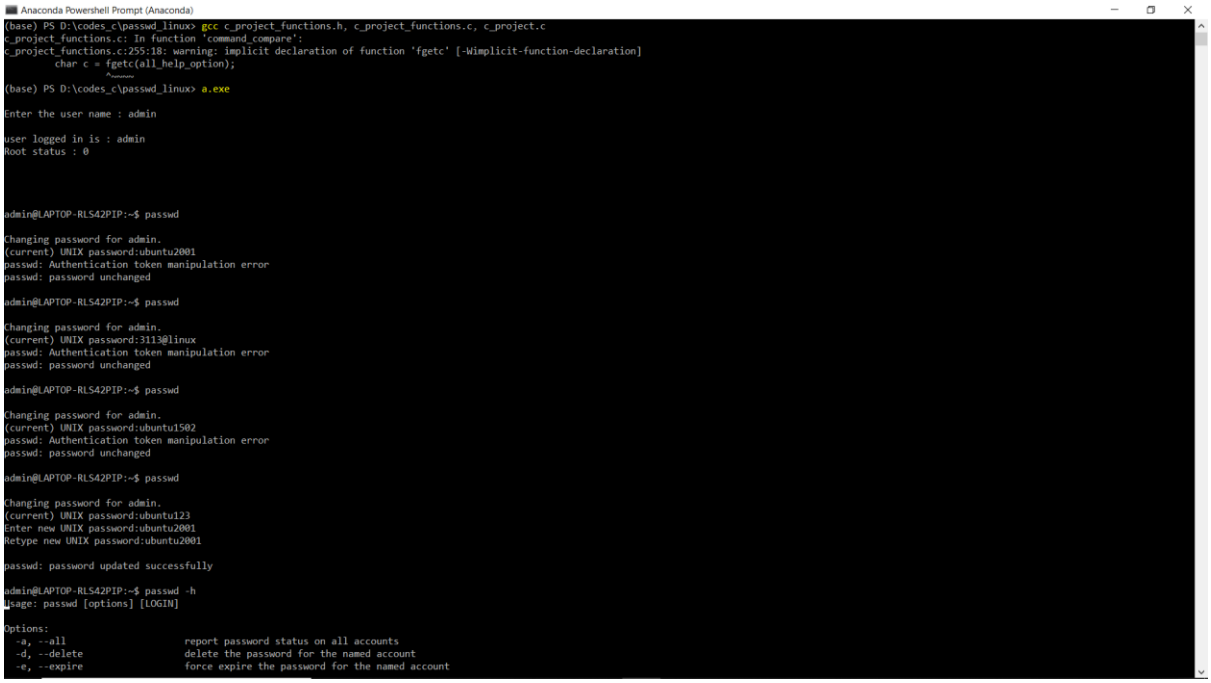
Output:


Anaconda Powershell Prompt (Anaconda)
(base) PS D:\codes_c\passwd_linux> gcc c_project_functions.h, c_project_functions.c, c_project.c
c_project_functions.c: In function 'command_compare':
c_project_functions.c:255:18: warning: implicit declaration of function 'fgetc' [-Wimplicit-function-declaration]
char c = fgetc(all_help_option);
(base) PS D:\codes_c\passwd_linux> a.exe
Enter the user name : admin
user logged in is : admin
Root status : 0
admin@LAPTOP-RLS42PIP:~$ passwd
Changing password for admin.
(current) UNIX password:ubuntu2001
passwd: Authentication token manipulation error
passwd: password unchanged
admin@LAPTOP-RLS42PIP:~$ passwd
Changing password for admin.
(current) UNIX password:3113@linux
passwd: Authentication token manipulation error
passwd: password unchanged
admin@LAPTOP-RLS42PIP:~$ passwd
Changing password for admin.
(current) UNIX password:ubuntu1502
passwd: Authentication token manipulation error
passwd: password unchanged
admin@LAPTOP-RLS42PIP:~$ passwd
Changing password for admin.
(current) UNIX password:ubuntu123
Enter new UNIX password:ubuntu2001
Retype new UNIX password:ubuntu2001
passwd: password updated successfully
admin@LAPTOP-RLS42PIP:~$ passwd -h
Usage: passwd [options] [LOGIN]
Options:
-a, --all report password status on all accounts
-d, --delete delete the password for the named account
-e, --expire force expire the password for the named account


Anaconda Powershell Prompt (Anaconda)
admin@LAPTOP-RLS42PIP:~$ passwd -h
Usage: passwd [options] [LOGIN]
Options:
-a, --all report password status on all accounts
-d, --delete delete the password for the named account
-e, --expire force expire the password for the named account
-h, --help display this help message and exit
-k, --keep-tokens change password only if expired
-i, --inactive INACTIVE set password inactive after expiration
to INACTIVE
-l, --lock lock the password of the named account
-n, --mindays MIN_DAYS set minimum number of days before password
change to MIN_DAYS
-q, --quiet quiet mode
-r, --repository REPOSITORY change password in REPOSITORY repository
-R, --root CHROOT_DIR directory to chroot into
-S, --status report password status on the named account
-u, --unlock unlock the password of the named account
-w, --warndays WARN_DAYS set expiration warning days to WARN_DAYS
-x, --maxdays MAX_DAYS set maximum number of days before password
change to MAX_DAYS
admin@LAPTOP-RLS42PIP:~$

Code:

c_project.c

```c
/*
This project mimics the linux command passwd which changes the password of the
 logged in user.
This project also mimics some of the options provided by the passwd command.
The project assumes the entire command length is less than 256 characters.
The project also assumes that the username length is lesser than or equal to 3
2 characters and the password length is lesser than or equal to 64 characters
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "c_project_functions.h"

int main()
{
    char *command = (char *)malloc(256 * sizeof(char)),
         *user = (char *)malloc(32 * sizeof(char)),
         *user_attributes = (char *)malloc(256 * sizeof(char)), *ref = (char *
)malloc(32 * sizeof(char));
    int root = login(user, user_attributes);
    strcpy(ref, user);
    strcpy(command, "\0");
    printf("\n\n");
    while (1)
    {
        printf("\n\n%s@%s:~$ ", ref, getenv("COMPUTERNAME")); //gets the syste
m name on which the project is running
        gets(command);
        strcpy(user, ref);
        command_compare(command, user, root, ref);
    }
    free(command);
    free(user);
    free(user_attributes);
    free(ref);
    return 0;
}
/*
The only warning this project is getting is

"c_project_functions.c: In function 'command_compare':
```

```
c_project_functions.c:255:18: warning: implicit declaration of function 'fgetc
' [-Wimplicit-function-declaration]
         char c = fgetc(all_help_option);
                  ^~~~~"
*/
```

c_project_functions.h

```
void delchar(char *x, int a);
void command_compare(char *command, char *user, int root, char *ref);
int login(char *user, char *user_attributes);
```

c_project_functions.c

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

int klen; //global variable used for encryption

void delchar(char *x, int a) //function used to delete given number of charact
ers from a given string
{
    if ((a - 1) <= strlen(x))
        strcpy(&x[-1], &x[a - 1]);
}

//encryption functions
int shift(char c)
{
    int newpos = c;
    if (c <= 'z' && c >= 'a')
        newpos -= 97;
    else if (c <= 'Z' && c >= 'A')
        newpos -= 65;
    return newpos;
}

int check(int k)
{
    if (k == klen - 1)
        k = 0;
    else
        k++;
```

```
    return k;
}

char *vigenere(char *word)
{
    char keyword[] = "linux";
    int key, length = strlen(word), j = 0;
    klen = strlen(keyword);
    for (int i = 0; i < length; i++)
    {
        if (j <= klen - 1)
            key = shift(*(keyword + j));
        while (key > 25)
            key -= 26;
        if (*(word + i) <= 'z' && *(word + i) >= 'a')
        {
            if (key > 'z' - word[i])
            {
                int tkey = key;
                int gap = 'z' - *(word + i) + 1;
                *(word + i) = 'a';
                tkey = tkey - gap;
                *(word + i) += tkey;
                j = check(j);
            }
            else
            {
                *(word + i) += key;
                j = check(j);
            }
        }
        else if (*(word + i) <= 'Z' && *(word + i) >= 'A')
        {
            if (key > 'Z' - word[i])
            {
                int tkey = key;
                int gap = 'Z' - *(word + i) + 1;
                *(word + i) = 'A';
                tkey = tkey - gap;
                *(word + i) += tkey;
                j = check(j);
            }
            else
            {
                *(word + i) += key;
                j = check(j);
            }
        }
```

```
    }
    return word;
}

//update function used to update the passwd.txt and shadow.txt files
void update(int password_len, int root, char *ref, char *user, char *ref_shado
w)
{
    char *new_password = (char *)malloc(64 * sizeof(char)), *retype_password =
 (char *)malloc(64 * sizeof(char)), shadow_ref[256];
    printf("Enter new UNIX password:");
    gets(new_password);
    printf("Retype new UNIX password:");
    gets(retype_password);
    if (strcmp(new_password, retype_password) == 0)
    {
        vigenere(new_password); //encrypting the password
        delchar(ref, password_len + 1);
        delchar(ref_shadow, password_len + 1);
        strcat(user, ":");
        strcat(new_password, ":");
        strcat(user, new_password);
        strcpy(shadow_ref, user);
        strcat(shadow_ref, ref_shadow);
        strcat(user, ref);
        FILE *fptr1, *fptr2;
        int linectr = 0;
        char str[256], pname[] = "passwd.txt", temp[] = "temp.txt";

        fptr1 = fopen(pname, "r");
        fptr2 = fopen(temp, "w");
        if (fptr1 == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        if (fptr2 == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        while (!feof(fptr1))
        {
            strcpy(str, "\0");
            fgets(str, 256, fptr1);
            if (!feof(fptr1))
            {
                linectr++;
```

```
                if (linectr != root + 1)
                    fprintf(fptr2, "%s", str);
                else
                    fprintf(fptr2, "%s", user);
            }
        }
        fclose(fptr1);
        fclose(fptr2);
        remove(pname);
        rename(temp, pname);

        linectr = 0;
        strcpy(pname, "shadow.txt");
        strcpy(temp, "temp.txt");
        fptr1 = fopen(pname, "r");
        fptr2 = fopen(temp, "w");
        if (fptr1 == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        if (fptr2 == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        while (!feof(fptr1))
        {
            strcpy(str, "\0");
            fgets(str, 256, fptr1);
            if (!feof(fptr1))
            {
                linectr++;
                if (linectr != root + 1)
                    fprintf(fptr2, "%s", str);
                else
                    fprintf(fptr2, "%s", shadow_ref);
            }
        }
        fclose(fptr1);
        fclose(fptr2);
        remove(pname);
        rename(temp, pname);

        printf("\npasswd: password updated successfully");
    }
    else
```

```
        printf("passwd: Authentication token manipulation error\npasswd: passw
ord unchanged");
}

//command_compare function used to select the option to mimic
void command_compare(char *command, char *user, int root, char *ref)
{
    //changing the password
    if ((strcmp(command, "passwd") == 0) || (strcmp(command, "passwd -
k") == 0) || (strcmp(command, "passwd --keep-tokens") == 0))
    {
        printf("\nChanging password for %s.\n(current) UNIX password:", ref);
        char *password = (char *)malloc(64 * sizeof(char)), *ref = (char *)mal
loc(256 * sizeof(char)), *ref_shadow = (char *)malloc(256 * sizeof(256));
        strcpy(password, "\0");
        strcpy(ref, "\0");
        gets(password);
        vigenere(password); //encrypting the password
        int password_len = strlen(password), total_line_num = 1, user_len = 0,
 line_num = 0;
        FILE *passwd = fopen("passwd.txt", "r"), *shadow = fopen("shadow.txt",
 "r");
        if (passwd == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        if (shadow == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        while (fgets(ref_shadow, 256, shadow) != NULL)
        {
            while (*(ref_shadow + user_len) != ':')
                user_len++; //counting the number of characters in the usernam
e
            if (strncmp(ref_shadow, user, user_len) == 0)
                break;
            else
                strcpy(ref_shadow, "\0");
        }

        while (fgets(ref, 256, passwd) != NULL)
        {
            if (strncmp(ref, user, user_len) == 0)
                break;
            else
```

```
                strcpy(ref, "\0");
        }
        fclose(passwd);
        fclose(shadow);

        if (passwd == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        passwd = fopen("passwd.txt", "a");
        fputs("\0", passwd);
        fclose(passwd);

        if (shadow == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        shadow = fopen("shadow.txt", "a");
        fputs("\0", shadow);
        fclose(shadow);

        delchar(ref, user_len + 1);
        delchar(ref_shadow, user_len + 1);
        if (strncmp(ref, password, password_len) == 0)
            update(password_len, root, ref, user, ref_shadow);
        else
            printf("passwd: Authentication token manipulation error\npasswd: p
assword unchanged");

        free(ref);
        free(password);
    }
    //--help of -h option
    else if ((strncmp(command, "passwd -
h", 9) == 0) || (strncmp(command, "passwd --help", 13) == 0))
    {
        FILE *all_help_option = fopen("all_help_option.txt", "r");
        if (all_help_option == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        char c = fgetc(all_help_option);
        while (c != EOF)
        {
            printf("%c", c);
```

```
            c = fgetc(all_help_option);
        }
        fclose(all_help_option);
    }
    //showing the man page of the passwd
    else if ((strncmp(command, "man passwd", 10) == 0))
    {
        FILE *man_passwd = fopen("man_passwd.txt", "r");
        if (man_passwd == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        char c = fgetc(man_passwd);
        while (c != EOF)
        {
            printf("%c", c);
            c = fgetc(man_passwd);
        }
        fclose(man_passwd);
    }
    //showing the man page of the shadow file
    else if ((strncmp(command, "man shadow", 10) == 0))
    {
        FILE *man_shadow = fopen("man_shadow.txt", "r");
        if (man_shadow == NULL)
        {
            printf("\n The file cannot be opened");
            exit(0);
        }
        char c = fgetc(man_shadow);
        while (c != EOF)
        {
            printf("%c", c);
            c = fgetc(man_shadow);
        }
        fclose(man_shadow);
    }
    //to clear the console
    else if ((strcmp(command, "clear") == 0))
        system("cls");

    //though this bolck doesn't mimic any option it just says whether to execu
te this option you need to be root user or not
    else if ((strncmp(command, "passwd -
d", 9) == 0) || (strncmp(command, "passwd -
e", 9) == 0) || (strncmp(command, "passwd -i", 9) == 0) ||
```

```
            (strncmp(command, "passwd -
l", 9) == 0) || (strncmp(command, "passwd -
q", 9) == 0) || (strncmp(command, "passwd -u", 9) == 0) ||
            (strncmp(command, "passwd -
n", 9) == 0) || (strncmp(command, "passwd -
w", 9) == 0) || (strncmp(command, "passwd -x", 9) == 0) ||
            (strncmp(command, "passwd --
delete", 15) == 0) || (strncmp(command, "passwd --expire", 15) == 0) ||
            (strncmp(command, "passwd --
inactivate", 19) == 0) || (strncmp(command, "passwd --lock", 13) == 0) ||
            (strncmp(command, "passwd --
mindays", 16) == 0) || (strncmp(command, "passwd --quiet", 14) == 0) ||
            (strncmp(command, "passwd --
unlock", 15) == 0) || (strncmp(command, "passwd --warndays", 17) == 0) ||
            (strncmp(command, "passwd --maxdays", 16) == 0))
    {
        if (!root)
            printf("\nYou are the root user");
        else
            printf("\nPermission denied");
    }

    //default case
    else
        printf("Command '%s' not found", command);
}

//login function used to get the details of the user. Return root status
int login(char *user, char *user_attributes)
{
    printf("\nEnter the user name : ");
    gets(user);
    FILE *shadow = fopen("shadow.txt", "r");
    if (shadow == NULL)
    {
        printf("\n The file cannot be opened");
        exit(0);
    }
    int root = 0;
    while (fgets(user_attributes, 256, shadow) != NULL)
    {

        int user_len = 0;
        while (*(user_attributes + user_len) != ':')
            user_len++;
        if (strncmp(user_attributes, user, user_len) == 0)
        {
            break;
```

```
        }
        else
        {
            root++;
            strcpy(user_attributes, "\0");
        }
    }
    fclose(shadow);
    printf("\nuser logged in is : %s", user);
    printf("\nRoot status : %d\n", root); //if root value is 0 then the logged
 in user is the root user
    return root;
}
```

makefile.mk

```
a.out : c_project.o c_project_functions.o
    gcc c_project.o c_project_functions.o
c_project.o : c_project.c c_project_functions.h
    gcc c_project.c
c_project_functions.o : c_project_functions.c c_project_functions.h
    gcc c_project_functions.c
```