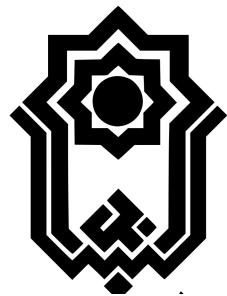


به نام خدا



دانشکده فنی و مهندسی

برنامه سازی پیشرفته

گزارش کار

Mini Project: txt-wiz-cli

نام و نام خانوادگی : حسین حقیقت

شماره دانشجویی : 40012358010

استاد

دکتر مرتضی یوسف صنعتی

1401 خرداد

فهرست

1	مقدمه	3
2	پیاده سازی پروژه	4
2.1	پیاده سازی کلاس های پایه و پیش نیاز ها	5
2.1.1	: Command	5
2.1.2	: App	5
2.1.3	: Book	6
2.1.4	: My input file	6
2.2	پیاده سازی ویرایشگر های رشته	8
2.2.1	ویرایشگر capitalize	9
2.2.2	ویرایشگر reverse	9
2.2.3	ویرایشگر replace	10
2.2.4	ویرایشگر split	10
2.2.5	ویرایشگر evaluate	11
2.2.6	ویرایشگر find	12
2.3	پیاده سازی ویرایشگر های فایل	13
2.3.1	Indent	13
2.3.2	Txtwrap	14
2.3.3	Diff	14
2.3.4	slide-show	15
2.4	Main	17
3	کتابخانه های استفاده شده	18
3.1	Iostream	18
3.2	Vector	18
3.3	Fstream	18
4	گیت	19
5	تحقیق بیشتر	20
5.1	سه کتابخانه cpp برای استفاده از CLI	20
5.2	سه ابزار مفید خط فرمان لینوکس	20

۱ مقدمه

رابط گرافیکی یا خط فرمان؟ مساله این است!

عام مردم بر این عقیده اند که رابط گرافیکی برنامه یا (GUI) بهتر از رابط خط فرمان یا (CLI) میباشد.

ولی این عقیده با نظر برنامه نویس ها و متخصصان کامپیوتر کاملا در تضاد است.

شاید علاقه مردم عادی به رابط گرافیکی ساده و قابل فهم تر بودن آن میباشد . به هر حال کار کردن با برنامه ای رنگارنگ که فیلد هایی برای وارد کردن اطلاعات دارد و همچنین با زدن دکمه هایی که برای کاربر در صفحه طراحی شده ، کار هایی را میتوان به ساده ترین صورت انجام داد و سوشه انگیز میباشد ولی دلیل برنامه نویس ها و متخصصان کامپیوتر برای استفاده از رابط خط فرمان چیست؟

استفاده از خط فرمان در کامپیوتر باعث ایجاد یک رابطه عمیق بین کاربر و کامپیوتر میشود . در این حالت کاربر نه تنها به بخش های وسیع ، خاص و امنیتی کامپیوتر دسترسی دارد بلکه میتواند ارتباط عاطفی نیز با کامپیوتر خود برقرار کند .

کار کردن با رابط خط فرمان مانند صحبت کردن با کامپیوتر میباشد . اگر کاربر به زبان انگلیسی مسلط باشد میتواند کار های خود را سریع تر و حتی ساده تر از رابط گرافیکی انجام دهد ، در حالی که بعد از گذشت زمان کمی میتوان متوجه رابطه عمیق و احساسی ایجاد شده بین کاربر و کامپیوتر شد . همه ای انسان ها به یار و یاوری نیاز دارند که در صورت نیاز به آنها کمک کند و نیاز آنها را بطرف کنند و چه یاری بهتر از کامپیوتر شخصی که تنها درخواسته اش از کاربر اتصال به برق و اینترنت میباشد .



و اما طراحی این پروژه که هدفش در راستای تقویت کار با رابط خط فرمان و همچنین استفاده از رشته ها در این رابط میباشد میتواند به کاربر کمک کند تا در صورت نیاز به برنامه ای که از قبل طراحی نشده خودش دست به کد شده و برنامه را طراحی و برنامه نویسی کند .

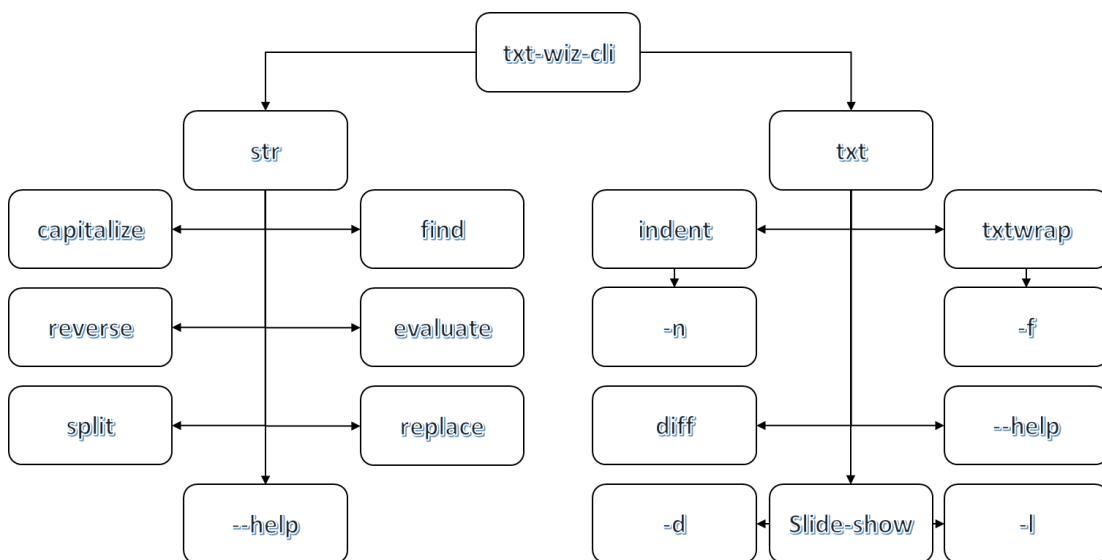
2 پیاده سازی پروژه

قابل ذکر است که این پروژه به صورت مازولار و تکه ای طراحی شده که شامل بخش های مختلف میباشد . با این کار نه تنها فهم پروژه ساده تر میشود بلکه برای توسعه ی نرم افزار فقط کافیست بخش جدید را به پروژه اضافه کرد و برنامه قبلی نیاز چندان زیادی به تغییر ندارد .

این نکته نیز حائز اهمیت است که ممکن است برای طراحی و پیاده سازی کلاس ها و توابع ، روش های بهینه و کوتاه تری وجود داشته باشد ولی این پروژه سعی در تمرکز بر نکات اصلی آموزشی (رابط کاربری خط فرمان و رشته ها) دارد .

در هر مرحله در صورت اشتباہ بودن کامند ارسالی از کاربر اطلاعات کمکی نمایش داده میشود ولی در صورتی که اطلاعات نمایش داده شده نیاز شما را برطرف نکرد میتوانید از تگ (--help) برای نمایش اطلاعات بیشتر استفاده کنید .

دستورات برنامه به صورت کلی در دو بخش فایل ها (txt) و رشته ها (str) طراحی شده و به صورت درختی رشد کرده است .



2.1 پیاده سازی کلاس های پایه و پیش نیاز ها

: Command 2.1.1

```
#ifndef _COMMAND_HPP
#define _COMMAND_HPP

#include <iostream>

class Command
{
protected:
    std::string mainText;
    Command(std::string);
};

#endif // _COMMAND_HPP
```

این کلاس که ریشه اکثر کلاس های این برنامه میباشد دارای یک متغیر از نوع رشته میباشد که توسط متدهای سازنده مقدار دهی میشود . این متغیر برای ذخیره سازی اصلی متن هر کامند طراحی شده .

: App 2.1.2

```
#ifndef _APP_HPP
#define _APP_HPP

#include <iostream>
#include <vector>

#include "capitalize.hpp"
#include "revers.hpp"
#include "replace.hpp"
#include "split.hpp"
#include "evaluate.hpp"
#include "find.hpp"
#include "txt.hpp"
#include "indent.hpp"
#include "txtwrap.hpp"
#include "diff.hpp"
#include "slide_show.hpp"
```

```

class App
{
private:
    static float version;
public:
    static float getVersion();
    static void help();
    static void moreHelp();
};

#endif // _APP_HPP

```

در این کلاس اطلاعات کلی همچون ورژن برنامه ذخیره شده و استفاده میشود.

: Book 2.1.3

```

#ifndef _BOOK_HPP
#define _BOOK_HPP

#include <iostream>
#include <vector>

class Book
{
private:
    std::vector<std::string> page;
public:
    void push(std::string);
    void open();
    int getPagesNumber();
    std::string getPage(int n);
    void print(int);
};

#endif // _BOOK_HPP

```

این کلاس برای slide show طراحی شده که ابتدا صفحه های کتاب را دریافت میکند سپس آن صفحات کتاب نمایش داده و به کاربر اجازه حرکت در بین صفحات را میدهد.

: My input file 2.1.4

```

#ifndef _MYINPUTFILE_HPP
#define _MYINPUTFILE_HPP

```

```
#include <iostream>
#include <fstream>

class MyInputFile : public std::ifstream
{
public:
    MyInputFile(std::string);
    ~MyInputFile();
    std::string getWord();
};

#endif // _MYINPUTFILE_HPP
```

این کلاس که از کلاس ifstream اطلاعاتش را به ارث برده تمام ویژگی های آن را دارا میباشد ولی علاوه بر آن ویژگی هایی را دارا میباشد :

1. متدهای سازنده برای باز کردن فایل
2. متدهای خرب برای بستن فایل
3. متدهای دریافت اطلاعات فایل به صورت کلمه به کلمه استفاده میشود .

2.2 پیاده سازی ویرایشگر های رشته

```
#ifndef _STR_HPP
#define _STR_HPP

#include <vector>

#include "command.hpp"

class Str : public Command
{
public:
    static void help();
    static void moreHelp();
protected:
    std::string result;
    Str(std::string);
    void print();
};

#endif // _STR_HPP
```

در کلاس (str) که از کلاس (command) ارث برده ویژگی های یک دستور از نوع اول (رشته ای) وجود دارد که عبارت است از :

1. MainText : این متغیر از نوع رشته ای در کلاس پدر (command) ذخیره میشود که در

هنگام ساخته شدن شی از کلاس (str) مقدار دهی میشود . قابل ذکر است که تمام کلاس های زیر مجموعه یک متن اصلی دارند که آن متن از ورودی دریافت و در این متغیر ذخیره میشود .

2. Help() : این تابع اطلاعات ابتدایی از استفاده درست از این بخش برنامه را به کاربر نمایش میدهد .(این تابع در کلاس های فرزند تغییر کرده و ویژگی های مختص همان کلاس ها را نمایش میدهد)

3. moreHelp() : این تابع اطلاعات بیشتر و توضیحات را همراه با مثال های صحیح از اجرای دستور و نتیجه آن را نمایش میدهد .(این تابع در کلاس های فرزند تغییر کرده و ویژگی های مختص همان کلاس ها را نمایش میدهد)

4. result : در این متغیر نتیجه تغییرات اعمال شده روی رشته اولیه ذخیره میشود .

.5 Print() : این تابع نتیجه را در خروجی چاپ میکند .

2.2.1 ویرایشگر capitalize

```
#ifndef _CAPITALIZE_HPP
#define _CAPITALIZE_HPP

#include "str.hpp"

class Capitalize : public Str
{
public:
    static void help();
    static void moreHelp();
    Capitalize(std::string);
private:
    void conversion();
};

#endif // _CAPITALIZE_HPP
```

در هر کدام از کلاس های طراحی شده در این برنامه مازولار بودن و سلسله مراتب اجراب متدها رعایت شده است ولی به دلیل سهولت کار و تمرکز بیشتر بر مطالب مرتبط تمام متدهای طراحی شده در متدهای سازنده فراخوانی شده .

```
Capitalize::Capitalize(string main) : Str(main){
    conversion();
    print();
}
```

در این حالت هنگام ساختن یک شی جدید از هر کلاس اطلاعات مربوطه را ورودی به آن میدهیم و نتایج را در همان لحظه دریافت میکنیم . به دلیل (private) تعریف شدن متدهایی که عملیات انجام میدهند در خارج از متدهای سازنده قابل دسترسی نمیباشند . ولی متدهای (print) برای نمایش نتیجه در آینده نیز در دسترس کاربر قرار دارد .

2.2.2 ویرایشگر reverse

```
void Reverse::conversion(){
    for (int i = mainText.size() - 1; i >= 0; i--)
        result.push_back(mainText[i]);
}
```

برای برعکس کردن متن فقط کافیست از آخر به اول متن ورودی را در `result` اضافه کنیم و سپس چاپ کنیم.

2.2.3 ویرایشگر replace

```
void Replace::conversion(){
    int index;
    while((index = mainText.find(before)) != string::npos) {
        mainText.erase(index, before.size());
        mainText.insert(index, after);
    }
    result = mainText;
}
```

برای جایه جایی یک کلمه در کل رشته کافیست ابتدا از متد استاندارد `[find()]` استفاده کرده و کلمه قدیمی را در رشته بیابیم.

سپس کلمه قدیمی را پاک کرده `[erase()]` و پس از آن کلمه جدید را در آن اضافه میکنیم `[insert()]`.

2.2.4 ویرایشگر split

```
void Split::conversion(){
    string word;
    result = "{}";
    for (size_t i = 0; i < mainText.size(); i++){
        if (mainText[i] == delimiter){
            result.append(word + ", ");
            word = "";
        }
        else
            word.push_back(mainText[i]);
    }
    result.append(word + "}");
}
```

در صورتی که کاربر از ما جدا کردن متن با استفاده از یک کاراکتر را درخواست کرد ، فقط کافیست در یک حلقه حرف به حرف ورودی را در متغیر `word` ذخیره و در هنگام رسیدن به کلمه مورد نظر با استفاده از متغیر استاندارد `(append())` آن کلمه را به نتیجه اضافه کنیم .

2.2.5 ویرایشگر evaluate

```

void Evaluate::conversion(){
    int result = 0;
    int value = 1;
    for (int i = mainText.size() - 1; i >= 0; i--)
    {
        int digit = mainText[i];
        if (digit >= '0' && digit <= '9')
            digit -= '0';

        else if (digit >= 'A' && digit <= 'Z')
            digit -= ('A' - 10);

        else if (digit >= 'a' && digit <= 'z')
            digit -= ('a' - 10);

        else
            throw;

        if (digit >= base)
            throw;

        result += (digit * value);
        value *= base;
    }
    Str::result = to_string(result);
}

void Evaluate::print(){
    cout << endl;
    cout << "Number : " << mainText << endl;
    cout << "base : " << base << endl << endl;

    cout << "Number : ";
    Str::print();
    cout << "base : " << "10";
    cout << endl;
}

```

در این کلاس علاوه بر متده تبدیل ، متده print() هم بازنویسی شده و علاوه بر چاپ کردن result اطلاعات دیگری را نیز در نتیجه چاپ میکند .

2.2.6 ویرایشگر find

```
void Find::conversion(){
    int result_counter = 0;
    string word = match;
    // remove '--' char from word
    while (word.find('--') != string::npos)
        word.pop_back();

    // find word in mainText
    int index;
    if((index = mainText.find(word)) != string::npos) {
        result = mainText.substr(index, match.size());
    }
    else{
        cout << "Not found!" << endl;
        throw;
    }
}
```

برای پیدا کردن کلمه مورد نظر ابتدا کلمه را در متغیر word ذخیره کرده و "--" ها را از آن حذف میکنیم.

پس از آن میتوان با متد find() که یک متد استاندارد برای رشته است کلمه را در متن اصلی پیدا کنیم.

در صورت یافت نشدن عبارت "Not found!" در result ریخته میشود.

2.3 پیاده سازی ویرایشگر های فایل

```
#ifndef _TXT_HPP
#define _TXT_HPP

#include <vector>

#include "command.hpp"
#include "myinputfile.hpp"

class Txt : private Command
{
public:
    static void help();
    static void moreHelp();
protected:
    Txt(std::string);
    MyInputFile file;
};

#endif // _TXT_HPP
```

کلاس (txt) که ریشه تمام کلاس های مرتبط با فایل است .

متغیری از نوع (myinputFile) در آن وجود دارد که وظیفه مدیریت فایل را بر عهده دارد و در هنگام ساخت یک شی از این کلاس و زیر مجموعه هایش فایل مربوطه را باز میکند .

Indent 2.3.1

```
Indent::Indent(string main, string indent) : Txt(main), ind(indent),
n(false)
{
    string line;
    while (!file.eof()){
        getline(file, line);
        cout << ind << " " << line << endl;
    }
}

Indent::Indent(string main, string indent, int num) : Txt(main),
ind(indent), n(true), number(num)
{
    string line;
```

```

while (!file.eof()){
    getline(file, line);
    cout << number << " " << ind << " " << line << endl;
    number++;
}

```

این کلاس شامل دو متده سازنده میباشد که دو نوع خروجی برای کاربر تهیه و چاپ میکند :

1. خروجی ساده

2. خروجی به همراه شماره خط در اول همان خط

Txtwrap 2.3.2

```

#ifndef _WRAP_HPP
#define _WRAP_HPP

#include "txt.hpp"

class Wrap : public Txt
{
private:
    int width;
    void conversion();
    void conversion_f();
    bool f_flag;
public:
    static void help();
    static void moreHelp();
    Wrap(std::string, int, bool);
};

#endif // _WRAP_HPP

```

در کلاس (txtwrap) دو متده سازنده تعریف شده که سازنده اول در حالت عادی و سازنده دوم در هنگام فعال بودن تگ (f) فراخوانی میشود .

Diff 2.3.3

```

#ifndef _DIFF_HPP
#define _DIFF_HPP

#include "txt.hpp"

```

```

class Diff : public Txt
{
private:
    MyInputFile secondFile;
public:
    static void help();
    static void moreHelp();
    void conversion();
    Diff(std::string, std::string);
};

#endif // _DIFF_HPP

```

این کلاس علاوه بر دارا بودن فایل اول که در کلاس پدر وجود دارد ، فایل دومی هم نیاز دارد .

متدهم هر دو فایل را خط به خط اجرا کرده و به دو صورت نمایش میدهد .

slide-show 2.3.4

```

#ifndef _SLIDE_SHOW_HPP
#define _SLIDE_SHOW_HPP

#include "txt.hpp"
#include "book.hpp"
#include <stack>

class SlideShow : public Txt
{
private:
    Book book;
    int len = 0;
    char del = 0;
public:
    static void help();
    static void moreHelp();
    void show_information();
    void show_page(int);

    void setAllPages_d();
    void setAllPages_l();

    SlideShow(std::string, int);
    SlideShow(std::string, char);
};

#endif // _SLIDE_SHOW_HPP

```

در این کلاس ابتدا یک (Book) که هر صفحه آن توسط خواسته کاربر جدا شده اند ساخته و تنظیم میشود .

سپس کتاب را باز کرده و صفحه اول آن را به کاربر نشان میدهد .
پس از آن کاربر میتواند از خود کلاس (Book) برای حرکت بین صفحه ها استفاده کند .

Main 2.4

در مین برنامه ورودی ها ابتدا وارد یک وکتور میشود و سپس به صورت درختی تمام حالات بررسی میشود . اگر در مرحله ای مشکل در دستور ورودی یافت شود متدهای help() و moreHelp() آن بخش فراخوانی میشود تا کاربر متوجه اشتباه خود شود . در هر مرحله غیر از متدهای help() و moreHelp() هم طراحی شده است که در صورت کافی نبودن اطلاعات داده شده کاربر میتواند از تگ help- برای دریافت راهنمایی و مثال های بیشتر استفاده کند .

3 کتابخانه های استفاده شده

Iostream 3.1

استفاده از این کتابخانه در ابتدای هر کد سی پلاس پلاس تا حدودی الزامی میباشد ، چرا که توابع جامع و کاملی را برای شروع برنامه نویسی در خود دارا میباشد .

Vector 3.2

کلاس وکتور در کلاس Book به ارث برده شده و همچنین در main برنامه برای ورودی های کاربر استفاده شده است .

Fstreame 3.3

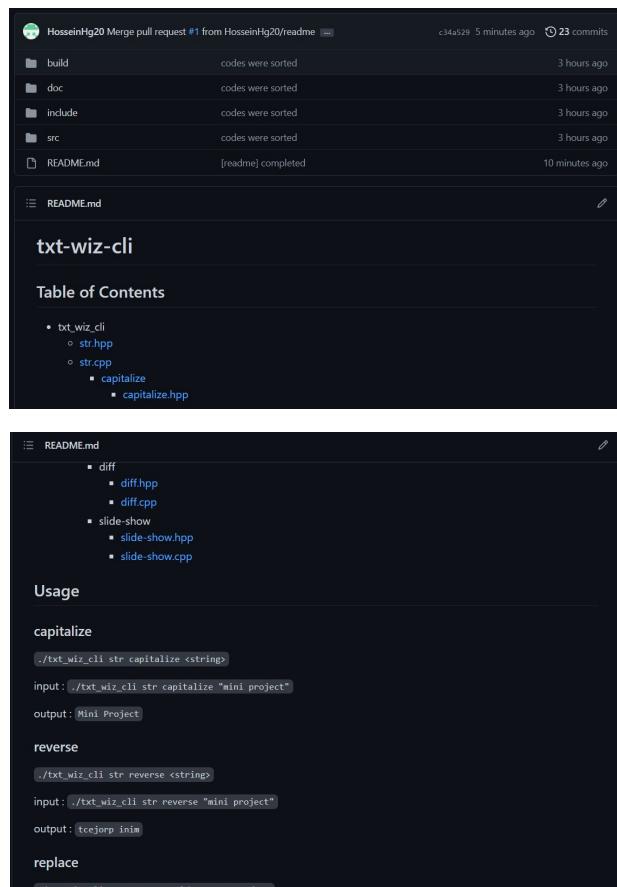
کلاس شخصی سازی شده ی ifstreame از کلاس myInputfile به ارث برده و تمام ویژگی های این کلاس را دارا میباشد .

در پروژه ازین کلاس برای باز کردن فایل ها و خواندن اطلاعات درون آنها استفاده شده .

4 گیت

در این پروژه از ابتدا از گیت استفاده شده و همچنین پروژه در سایت <https://github.com> بارگزاری شده است (پروژه خصوصی و برای عموم غیر قابل دسترس میباشد).

سعی بر این بود که با هر تغییر و رسیدن به خروجی مناسب در قسمت خاصی از برنامه ، در گیت ثبت شود که در اکثر مواقع این اتفاق صورت گرفت و در حال حاضر میتوان هم به تاریخچه ساخت پروژه دسترسی داشت و بدون دردسر آن را توسعه داد .



5 تحقیق بیشتر

5.1 سه کتابخانه `cpp` برای استفاده از CLI

کار با CLI (Command line interface) یا همان رابط کاربری خط فرمان تاریخچه طولایی دارد و اولین استفاده از آن را میتوان برابر دانست با اولین کامپیوتر های ساخته شده در تاریخ.

همانطور که گفته شد این رابط کاربری سن زیادی دارد پس میتوان نتیجه گرفت که مطمئناً افراد زیادی برای راحت تر و کار آمد تر شدن این رابط سعی و تلاش بسیاری کرده اند.

کتابخانه های زیادی در این باره ساخته شده و در سایت های مرتبط به اشتراک گذاشته شده اند که میتوان از معروف ترین آنها به سه مورد اشاره کرد:

1. <https://github.com/daniele77/cli>
2. <https://github.com/CLIUtils/CLI11>
3. <https://github.com/GrossoMoreira/shpp>

علاوه بر این سه کتابخانه کتابخانه های ریز و درشت، معروف و ناشناس دیگری نیز وجود دارند که کار با این رابط کاربری دوست داشتنی را دوست داشتنی تر میکنند.

برای اطلاع از کتابخانه های بیشتر میتوانید از سایت گیتهاب یا موتور جست و جو گوگل استفاده کنید.

5.2 سه ابزار مفید خط فرمان لینوکس

اولین ابزاری که به ذهن هر کاربر لینوکس برای باز کردن فایل های متنی میرسد دستور (`cat`) میباشد.

علاوه بر (cat) ویرایشگر (wim) هم کاربردیست که قابلیت های زیادی را پشتیبانی میکند ولی نیاز است قبل از استفاده آن را نصب کنیم .

همچنین دستور های پرکاربرد دیگری نیز وجود دارد که به شرح زیر است :

Diff .1 : برای مشاهده تفاوت دو فایل متنی استفاده میشود .

Head .2 : برای مشاهده 10 خط اول فایل متنی (که میتوان با استفاده از تگ n- تعداد خطوط را مشخص کرد) .

Tail .3 : با این دستور میتوان 10 خط آخر فایل متنی را مشاهده کرد (همچنین میتوان با استفاده از تگ n- تعداد خطوط را مشخص کرد) .

Wc .4 : درصورتی که کاربر میخواهد تعداد خطوط ، کلمات و حروف یک فایل متنی را استخراج کند میتواند از این دستور استفاده کند .