به نام خدا

درس مبانی برنامه‌سازی

فاز اول پروژه

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

نیم سال اول ۰۲ ـ ۰۱

استاد:

دکتر محمدامین فضلی

مهلت ارسال:

فاز اول: ۳۰ دی
ساعت ۲۳:۵۹:۵۹

مسئول پروژه:

امیرمهدی کوششی

مسئول فاز اول:

سیاوش رحیمی شاطرانلو

طراحان فاز اول:

محسن قاسمی، امید دلیران، امیرحسین عزیزی، نیکان واسعی، علیرضا کاظمینی، مهدی تیموری انار، علی آقایاری

مسئولین تنظیم مستند:

امیرمهدی کوششی، آرمان بابایی

فهرست

نکات قابل توجه

- پس از اتمام این فاز، در گیت خود یک تگ با ورژن "v1.0.0" بزنید. در روز تحویل حضوری این tag بررسی خواهد شد و کدهای پس از آن نمره‌ای نخواهد گرفت. برای اطلاعات بیش‌تر در مورد شیوه ورژن‌گذاری، می‌توانید به این لینک مراجعه کنید. البته برای این پروژه صرفا رعایت کردن همان ورژن گفته شده کافیست، اما خوب است که با منطق ورژن‌بندی هم آشنا بشوید.
- در صورت کشف تقلب، برای بار اول منفی نمرهٔ آن فاز برای آن فرد ثبت می‌شود و برای بار دوم، نمرهٔ منفی کل پروژه برای فرد لحاظ خواهد شد.

مقدمه

اهداف پروژه

- هدف این پروژه طراحی یک ابزار ویرایش فایل مشابه vim است. احتمالا در کارگاه کامپیوتر و یا جاهای دیگر با این ابزار کار کرده‌اید. در غیر این صورت می‌توانید از طریق این لینک نحوه کار با این ابزار را ببینید.
- در این فاز از پروژه باید قابلیت‌هایی که برای این ابزار آمده است را پیاده سازی کنید.
- در این پروژه نحوه پیاده‌سازی اجزای مختلف از اهمیت بسیاری برخوردار است و تنها خروجی نهایی مهم نیست. از این رو برای تمیزی کد خود ارزش قائل شوید.
- آشنایی با سیستم مدیریت نسخه Git و کار بر روی پروژه بر بستر یک مخزن Github، یکی از اهداف مهم پروژه است. در این مورد توصیه می‌شود تغییرات خود را در دوره‌های کوتاه مدت commit کنید.

کلیات پروژه

در این فاز، منطق پروژه باید پیاده‌سازی شود. نحوهٔ ارتباط با کاربر نیز از طریق واسط کاربری کنسول است. همچنین توجه کنید که در فاز دوم قرار است برای منطق پروژه که در این فاز طراحی شده است، رابط کاربری تحت کنسول (Command Line Interface) طراحی کنید. در نتیجه جداسازی درست منطق پروژه از رابط کاربری آن باعث می‌شود که در فاز ۲ نیاز به بازنویسی کدهای منطق به شدت کاهش یابد.

در ادامهٔ مستند، موجودیت‌ها، نمای کلی رابط کاربری سیستم، نقش‌ها و دستورات لازم شرح داده‌شده است.

نکته ۱: در هر جایی از پروژه می‌توانید هرگونه خلاقیتی را به‌کار ببرید. با این حال توجه کنید که خواسته‌های واضح پروژه بایستی انجام شوند و سیستم ورودی گرفتن و خروجی دادن شما باید مطابق جزییات گفته شده در این مستند باشد.

نکته ۲: در فاز ۱ شما باید از طریق کنسول با برنامه خود ارتباط برقرار کنید، به همین دلیل نیاز به یک سری دستورات مشخص می‌باشد که شما از طریق آن دستورات می‌توانید پروژه را جلو ببرید. به همین منظور دستورات پیشنهادی در کنار هر قابلیت آمده است که می‌توانید از آن‌ها استفاده کنید (دقت کنید این دستورات پیشنهادی هستند و شما می‌توانید دستورات خود را به سلیقه خود پیاده سازی کنید، اما نباید دستورات شما از یک چهارچوب معین خارج شود، در ادامه نیز بیشتر با این موضوع آشنا خواهید شد.)

نکته ۳: در مستند بعضی از دستورهایی که مشاهده می‌کنید فرمتی به شکل زیر دارند:

دستور

compare file1 file2

همچنین توجه کنید که اگر دستور نامعتبری وارد شد که در قسمت مربوطه، برای آن خطای به خصوصی در نظر گرفته نشده بود، پیام زیر را چاپ کنید:

پیغام به کاربر

invalid command

راه‌اندازی مخزن گیت‌هاب

همانطور که می‌دانید برای پروژه لازم است بر روی یک مخزن (repository) گیت فعالیت کنید. برای ساختن این مخزن، کافیست وارد این لینک شوید.

ابتدا با لیستی مواجه می‌شوید که شماره دانشجویی تمام افراد در آن موجود است. شماره دانشجویی خود را بیابید و بر روی آن کلیک کنید. سپس مدتی صبر کنید و صفحه را refresh کنید، حال می‌توانید مخزن گیت‌هاب خود را مشاهده کنید.

معرفی ابزار

همانطور که قبلا توضیح دادیم، شما باید یک ابزار ویرایش متن مانند vim طراحی کنید. در این فاز از پروژه هیچ نیازی به طراحی گرافیکی نیست اما در فاز بعد شما باید مانند خود ابزار vim یک طراحی گرافیکی داشته باشید. در این فاز دستورات از طریق کنسول مانند تمارینی که تا به حال زدید به شما داده می‌شود و شما باید دستورات را بگیرید و پردازش کنید و تغییرات را روی فایل‌ها اعمال کنید و در نهایت خروجی مناسب آن دستور را به کاربر نشان دهید.

توضیح بخش‌های مختلف پروژه

موارد عمومی و قابلیت‌های ابزار

create file

شما یک فولدر root باید برای پروژه خود داشته باشید که تمامی مسیرها از این فولدر شروع خواهند شد. اسم این فولدر دلخواه هست اما پیشنهاد می‌کنیم که اسم آن را همان root بگذازید. دقت کنید که تمامی آدرس دهی‌ها از آن فولدر خواهد بود.

دستور

```
createfile –file <file name and address>
```

مثال

```
createfile –file /root/dir1/dir2/file.txt
```

در صورتی که فولدرهای آمده در آدرس وجود نداشتند شما باید آن‌ها را بسازید. در مثال بالا فولدرهای dir۱ و dir۲ هست.

در صورتی که فایل موردنظر از قبل وجود داشت، خطای مناسب چاپ کنید.

نکته بسیار مهم

در دستورهایی که ورودی می‌گیرند، به دو صورت ورودی داده می‌شود. یا ورودی بین دو ” ” می‌آید و یا بدون این دوعلامت (گیومه) می‌آید. فرق آن‌ها بدین معناست:

- اگر ورودی بین دو ” ” آمد به این معنی است که یک عبارت است، یعنی بین کلمات آن فاصله وجود دارد.
- اگر ورودی بین دو ” ” نیامد، به این معنی است که در آن عبارت بین کلمات فاصله‌ای وجود ندارد.

به مثال‌های زیر برای فهم موضوع دقت کنید.

مثال

```
–file /root/dir1/dir2/file.txt OR –file ”/root/dir1/temp file 1.txt”
–str salam OR – str ”salam khobi ?”
```

دقت کنید این مورد در تمامی دستوراتی که ورودی می‌گیرند صادق است.

insert

با زدن دستور زیر شما باید متن مورد نظر را در جایگاه مورد نظر قرار دهید.

دستور

insertstr –file <file name> –str <str> —pos <line no>:<start position>

مثال

insertstr –file /root/dir1/dir2/file.txt –str Salam –pos 2:5

در صورتی که فایل مورد نظر وجود نداشت خطای مناسب چاپ کنید.

دقت کنید شماره خطوط از ۱ و شماره کاراکترها از ۰ شروع می‌شود. در واقع pos 1:0– به معنی خط اول و همان ابتدای خط است.

به موارد زیر دقت کنید:

- در صورتی که در رشته ورودی "\n" آمد، به معنی خط جدید است و شما باید در فایل به خط بعدی روید.
- در صورتی که بعد از "\" "\" دیگری آمد، به این معنی است که شما باید دقیقا همان رشته را در فایل بگذارید.

به مثال‌های زیر توجه کنید تا بهتر متوجه شوید:

```
Salam\n khobi?
```

در فایل می‌شود:

```
Salam
khobi?
```

به مثال زیر نیز توجه کنید:

```
Salam \\n khobi?
```

در فایل می‌شود:

```
Salam \n khobi?
```

دقت کنید بین دو گیومه، می‌تواند گیومه‌های دیگری نیز بیاید و شما باید حواستان به

این مورد نیز باشد، برای مثال ورودی زیر نیز درست است:

مثال

–str ”He said: ”Hello World!”; just this.”

درواقع عبارت بالا به این معنی است که شما باید در فایل نوعی خود عبارت زیر را بنویسید:

```
//file.txt
He said: "Hello World!"; just this.
```

یعنی گیومه‌های داخلی جزوی از متن هستند و برای مشخص کردن عبارت **نیستند** .

cat

با این دستور شما محتوای فایل را در خروجی نشان می‌دهید.

دستور

cat –file <file name>

به عنوان مثال فرض کنید شما یک فایل دارید که در آن محتوای زیر نوشته شده است.

```
//file.txt
Salam khobi?
```

مثال

cat –file file.txt

خروجی

Salam khobi?

remove

دستور

removestr –file <file name> –pos <line no>:<start position> -size <number of characters to remove> -f -b <forward or backward>

مثال

removetstr –file /root/file1.txt –pos 2:1 -size 3 -b

فرض کنید در یک فایل قبل از اجرای دستور بالا داریم:

```
Salam Khobi?
```

بعد از اجرای دستور بالا محتوای فایل عبارت است از:

```
Salahobi?
```

کاراکترهای "m" و "\n" و "K" پاک شده‌اند.

فلگ‌های f- و b- به این معنی است که رو به جلو پاک شود یا رو به عقب.

copy

دستور

copystr –file <file name> –pos <line no>:<start position> -size <number of characters to copy> -f -b <forward or backward>

با این دستور شما رشته موردنظر را در clipboard کپی می‌کنید.

cut

دستور

cutstr –file <file name> –pos <line no>:<start position> -size <number of characters to cut> -f -b <forward or backward>

این دستوری ترکیبی است از دو دستور کپی کردن و حذف کردن. در واقع با زدن این دستور شما رشته موردنظر را در clipboard کپی می‌کنید و آن را از از فایل حذف می‌کنید.

paste

دستور

pastestr –file <file name> –pos <line no>:<start position>

با این دستور رشته‌ای که در clipboard است را در جایگاه موردنظر الصاق می‌کنید.

find

این دستور، آدرس یک فایل را می‌گیرد و یک واژه نیز می‌گیرد و آن را درون فایل می‌جوید و می‌گوید که نخستین بار در کجای فایل آمده است. به عنوان مثال اگر در فایل مورد نظر (a.txt) واژه‌ی vajhe که جستجو شده، نخستین بار از کارکتر ۱۰۰ام فایل آغاز شده، دستور find –str vajhe –file a.txt باید ۱۰۰ را بدهد.

دستور

```
find –str <str> –file <file name>
```

چنانچه بخواهیم در یک فایل دنبال یک عبارت بگردیم که در میان آن ممکن است فاصله (white-space) وجود داشته‌باشد، باید آن عبارت را درون گیومه بگذاریم مثلاً

مثال

```
find –str ”this is an expression containing white-space.” –file a.txt
```

همچنین باید بتوان واژگان یا عبارات را به صورت wildcard هم جست. برای سادگی تنها الگوهای wildcard ای که باید پیاده‌سازی کنید، دو الگوی *a و a* هستند. یعنی یا در آغاز و یا در پایان علامت * می‌آید و به این معنا است که آنجا هر دنباله‌ای از کاراکترها از جمله دنباله‌ی تهی (به جز کارکتر فاصله یا 0\یا EndOfFile) ممکن است بیاید. به عنوان مثال fi* می‌تواند با fire و fine و five و ... منطبق شود و our* می‌تواند با our و four و your و ... منطبق شود ولی مثلاً our* با y our مطابق نیست و فقط با بخش our اش مطابق است.

چنانچه کاربر بخواهد در عبارت جستجویش واقعاً دنبال کارکتر * در متن بگردد، باید پیش از این کارکتر در دستور ورودی‌اش، \بگذارد پس مثلاً الگوی fi* با رشته‌ی five منطبق نیست امّا با رشته‌ی fi* منطبق است.

مثال

```
find –str ”an expression with \* but not wildcard” –file a.txt
```

اگر عبارت یا الگوی جستجوشده وجود نداشته‌باشد، این دستور باید به ما ۱ ـ برگرداند.

همانگونه که گفته‌شد، اگر در فایل مورد نظر چندین جا عبارت یا الگوی جستجوشده وجود داشته‌باشد، این دستور به طور پیش‌فرض، اندیس نخستین جا را بر حسب شماره‌ی کارکتر می‌دهد. امّا می‌توان با دادن ورودی‌های بیشتر به این دستور، این مقادیر را تغییر داد.

اگر فایل آدرس مورد نظر وجود نداشت، باید پیغام خطای مناسبی بدهد.

ویژگی count

چنانچه پس از این دستور عبارت count- را بنویسیم، این دستور به ما تعداد تکرار الگو یا عبارت جستجوشده را در فایل مورد نظر می‌دهد. طبیعتاً اگر چنین چیزی یافت نشود باید ۰ را برگرداند.

مثال

```
find –str fire –file a.txt -count
```

ویژگی at

پس از این ویژگی یک عدد n می‌آید که می‌گوید امینn باری را که آن عبارت یا الگو در فایل مورد نظر آمده در نظر بگیرد. اگر n بیشتر از تعداد تکرار آن عبارت باشد، باید به ما ۱ ـ برگرداند.

مثال

```
find –str fire –file a.txt -at 3
```

ویژگی byword

همانگونه که گفته‌شد، این دستور شماره‌ی کارکتر آغاز عبارت را می‌داد. اگر بخواهیم به جای آن، بگوید عبارت از چندمین واژه (منظور از واژه هر زیررشته‌ای از متن فایل است که شامل space نشود ولی کاراکتر بعد و قبل آن (در صورت وجود) space باشد) آغاز شده.

مثال

```
find –str ”salam khubi*” –file a.txt -byword
```

ویژگی all

در صورت وجود این ویژگی در دستور داده‌شده، باید شماره‌ی همه‌ی تکرارهای یافته‌شده را بدهیم. یعنی مثلاً اگر در فایل مورد نظر، عبارت difficult project در واژه‌های سوم، دهم و بیستم آمده (یعنی واژه‌های سوم، دهم و بیستم difficult هستند و چهارم و یازدهم و بیست‌ویکم project) دستور زیر باید خروجی زیر را بدهد.

مثال

```
find –str ”difficult project” –file a.txt -all -byword
```

خروجی

```
3, 10, 20
```

دقّت‌کنید که ترتیب ویژگی‌ها ممکن است به هر ترتیبی عوض شود و حتی لزومی ندارد که همگی بیایند امّا چنانچه ترکیب نامعتبری از آن‌ها بیاید، باید پیغام خطای مناسبی نشان‌داده‌شود (مثلاً at و byword می‌توانند با هم در یک دستور بیایند ولی at و all منطقاً نمی‌توانند همزمان در یک دستور داده‌شوند و اگر داده‌شوند، برنامه باید پیغام خطای مناسبی بدهد).

replace

دستور replace تا حدّی همانند find است فقط با این تفاوت که پس از عبارت مورد جستجو، عبارت دیگری برای جایگذاری داده‌می‌شود. سپس باید به جای این که عبارت یافته‌شده را به کاربر نمایش‌دهیم، با عبارت دوم جایگزین کنیم و فایل را ذخیره کنیم. چنانچه مشکلی وجود داشت (مثلاً این که عبارت مورد نظر پیدا نشد) باید مشکل را به کاربر گزارش دهیم و چنانچه عملیات موفقیت‌آمیز انجام‌شد، باید پیغام موفّقیت به کاربر داده‌شود.

دستور

```
replace –str1 <str> –str2 <str> –file <file name> [-at <num> | -all]
```

همانند دستور find ویژگی‌های at و all را داریم که all یعنی باید همه‌ی جاهایی که عبارت مورد نظر پیدا می‌شود آن را جایگذاری کنیم و at یعنی باید مثلاً سومین باری که عبارت وجود دارد را جایگذاری کنیم. اگر هیچ کدام داده‌نشود، به طور پیشفرض at 1 در نظر می‌گیریم. توجّه بفرمایید که all و at منطقاً نمی‌توانند به همراه هم بیایند و اگر کاربر آنها را با هم وارد کند، باید پیغام خطای مناسب به او بدهید. توجّه بفرمایید که در این دستور

برخلاف find ویژگی‌های count و byword نداریم. توجّه کنید که همچنان عبارت مورد جستجو می‌تواند به صورت wildcard داده‌شود امّا طبیعتاً عبارت دوم که باید جایگزین عبارت نخست شود، نمی‌تواند wildcard باشد. اگر فایل آدرس مورد نظر وجود نداشت، باید پیغام خطای مناسبی بدهد.

مثال

replace –str1 ”salam khubi?” –str2 ”Dorud! Che Khabar?” –file a.txt -all

مثال

replace –str ”fi* five” –file a.txt -at 4

grep

از دستورهای معروف لینوکس است که برای جستجو کردن محتوای درون فایل‌ها از آن استفاده می‌شود. در این قسمت هدف پیاده‌سازی یک نسخه‌ی ساده از این قابلیت است.

دستور

grep [options] –str <pattern> –files [<file1> <file2> <file3> …]

برای مثال میخواهیم پترن proj را در فایل های main.txt ، text.txt ، project.txt پیدا کنیم.

فرض کنید درون فایل‌ها به صورت زیر باشد:

```
  // main.txt
project name : vim
author : someone
version : 12.5

  // project.txt
project status : in progress
proj phase : 1

  // text.txt
random text
example
proj
```

دستور ما به شکل زیر خواهد بود:

مثال

```
grep –str ”proj” –files main.txt text.txt project.txt
```

خروجی این دستور باید تمام خط هایی از فایل های ورودی باشد که استرینگ proj زیرمجموعه آن خط است.

خروجی

```
main.txt: project name : vim
project.txt: project status : in progress
project.txt: proj phase : 1
text.txt: proj
```

علاوه بر این، پروژه شما باید آپشن های c و l را پشتیبانی کند.

آپشن C

در صورتی که کاربر از آپشن c استفاده کند ، دیگر نیاز نیست که تمامی خطها چاپ شوند بلکه کافی است تعداد خطهای خروجی نمایش داده شوند.

مثال

```
grep -c –str ”proj” –files main.txt text.txt project.txt
```

خروجی

```
4
```

آپشن l

در صورتی که کاربر از آپشن l استفاده کند ، دیگر نیاز نیست که تمامی خطها را پرینت کنید بلکه کافی است نام فایل‌هایی که عبارت در آن‌ها پیدا شده چاپ شوند.

مثال

```
grep -l –str ”proj” –files main.txt text.txt project.txt
```

خروجی

main.txt
text.txt
project.txt

undo

این دستور مانند دستور Undo در word یا دیگر ویرایشگرها عمل می‌کند و با زدن این دستور فایل شما باید به مرحله قبل از آخرین تغییر موفقیت‌آمیز برگردد. توجه کنید که اجرای هر دستوری اعم از undo یک تغییر محسوب می‌شود.

دستور

undo –file <file>

در اثر اجرای این دستور، فایلی <file> به حالت قبل از ایجاد آخرین دستور برمی‌گردد.

مثال

```
  // file.txt
This is an example for vim project
```

```
  // file.txt
This is an example for vim project
```

سپس شما کلمه is را به was تغییر می دهید:

```
  // file.txt
This was an example for vim project
```

با اجرای دستور زیر، فایل باید مشابه زیر به حالت اولیه برگردد.

مثال

undo –file file.txt

با اعمال این تغییر، محتوای فایل ما عبارت است از:

```
  // file.txt
This is an example for vim project
```

دقت کنید که فقط یک مرحله undo کافی است و بیشتر از آن مورد نیاز نیست.

closing pairs

این ویژگی را حتما در codeblocks یا ویرایشگرهای (editor) دیگری که با آن‌ها کار کرده‌اید دیده‌اید. طرز کار این قابلیت این گونه است که وقتی شما در ویرایشگر خود خطی مانند "int main{something}" را می‌نویسید، ویرایشگر خودبه‌خود آن را به صورت زیر می‌نویسد:

```
int main {
    something
}
```

این قابلیت از وجود چند ویژگی مطمئن می‌شود:

- "{" و "}" آخرین کاراکتر در خط خود باشند.
- در صورتی که قبل از آکولاد باز، کاراکتر غیرسفیدی (non-whitespace character) قرار دارد، آکولاد باز باید دقیقا یک کاراکتر فاصله (space) با آخرین کاراکتر غیرسفید فاصله داشته باشد.
- آکولاد بسته به جز فاصله‌ی شروع خط تنها کاراکتر در خط خود باشد.
- بین "{" و "}" یک خط فاصله می‌اندازد.
- محتویات بین "{}" ها را اگر موجود باشد در خط خالی بین آنها می نویسد به صورتی که شروع خط یک tab یا چهار space نسبت به اولین خط قبل آن که آکولاد باز قرار دارد، جلوتر باشند.

دستور

auto-indent <file>

در نتیجه‌ی این دستور فایل باید به گونه‌ای تغییر کند که بدون تغییر در کاراکترهای غیرسفید شرط‌های بالا رعایت شده باشند. در صورتی که آکولادگذاری در فایل ورودی صحیح نباشد، می‌توانید شرط‌ها را رعایت نکنید، اما همچنان محتویات غیرسفید فایل نباید تغییر کنند.

مثال

```
  // before
int main()  {  if (1 == 1){printf(“Hello world”)};  }

  // after
int main() {
    if (1 == 1) {
      printf(“Hello world”);
    }
}
```

مثال دوم

```
  // before
{{}       }

  // after
{
    {
    }
}
```

text comparator

ابزار text comparator ابزاری برای مقایسه‌ی خط‌به‌خط محتوای دو فایل متنی است. این دستور خط به خط محتوای دو فایل را مقایسه می‌کند و خطوطی در دو فایل که با هم متفاوت هستند را نشان می‌دهد.

دستور

compare <file1> <file2>

در نتیجه‌ی این دستور هر دو خط که متفاوت باشند، سه خط به طریقی مشابه زیر باید نمایش داده شود:

```
  //output
============ #<line number> ============
<line from file1>
<line from file2>
```

علاوه بر این اگر یکی از فایل‌ها طولانی‌تر از دیگری است، در ادامه‌ی خروجی خطوط اضافه درج می‌شوند. همچنین باید ذکر شود که کدام یک از فایل‌ها بلندتر است. می‌توانیم قرارداد کنیم که اگر فایل اول یا دوم طولانی‌تر بود ادامه‌ی خروجی به یکی از صورت‌های زیر (به ترتیب) نمایش داده می‌شود.

```
  //output
<<<<<<<<<<<< #<start line number> - #<end line num-
ber> <<<<<<<<<<<< <content from file1>
```

و یا

```
  //output
>>>>>>>>>>>> #<start line number> - #<end line num-
ber> >>>>>>>>>>>> <content from file2>
```

مثال

فرض کنید دو فایل a.txt و b.txt را در اختیار داشته باشیم و دستور a.txt compare b.txt را اجرا کنیم.

```
  // a.txt
Salam
Khubi?
```

```
  //b.txt
Salam
Khubam.
Khubi?
```

در این صورت خروجی باید مشابه زیر باشد:

```
  compare a.txt b.txt
============ #2 ============
Khubi?
Khubam.
>>>>>>>>>>>> #3 - #3 <<<<<<<<<<<<
Khubi?
```

امتیازی

در صورتی که تفاوت دو خط تنها در یک کلمه جدا شده با فاصله بود، مشابه زیر آن را نشان دهید.

```
  //output
============ #<line number> ============
Dars >>FOP<< shirin ast
Dars >>ryazi1<< shirin ast
```

directory tree

پروژه شما باید قابلیت نمایش نمودار درختی آدرس‌ها را داشته باشد. به این معنی که فایل‌ها و فولدرهای موجود در فولدری اجرای دستور را به صورت بازگشتی نمایش دهد.

دستور

tree <depth>

نمودار درختی با عمق <depth> باید در خروجی بیاید. برای پیاده‌سازی این قسمت نیاز به توابع بازگشتی دارید. به ازای آدرس dir۱ شما باید روی همه فایل‌ها و فولدرهای این آدرس پیمایش کنید و در صورتی که به فایل رسیدید، نام آن فایل و در صورتی که به فولدری رسیدید، نام آن فولدر و محتویات داخل آن (که عمق آن‌ها از عمق dir۱ یک واحد بیشتر است) را باید چاپ کنید. همچنین می‌بایست محتویات آدرس‌ها و زیرآدرس‌ها را به صورت متمایز نشان دهید. (متفاوت بودن عمق آن‌ها محسوس باشد.) توجه کنید که اگر depth = -1 آنگاه می‌بایست تمامی زیرآدرس‌ها را نمایش دهید.

در صورتی که عدد کوچک‌تر از -1 به ورودی داده شود، شما می‌بایست خطای invalid

depth نمایش دهید.

مثال

یک نمونه از نمودار درختی آدرس‌ها را می‌توانید در تصویر زیر مشاهده کنید.

```
tree 2
example_file_d1.tst
example_sub_dir_1:
├────file1.tst
└────file2.tst
example_sub_dir2_1:
├────file1_on_depth2.tst
└────example_sub_dir_2:
example_sub_dir3_1:
example_file2_d1.tst
```

arman

این قسمت امکان انتقال خروجی یک دستور به عنوان یکی از ورودی‌های دستور بعدی را فراهم می‌کند. به این ترتیب خروجی دستور قبلی به عنوان ورودی str– دستور بعدی در صورت وجود استفاده می‌شود. برای استفاده از این قابلیت بین دو دستور علامت =D قرار می‌گیرد.

مثال

فرض کنید قبل از اجرای دستور، فایل a.txt یک فایل خالی است.

مثال

```
tree 2 =D insertstr –file a.txt –pos 1:0
```

بعد از اجرای این دستور محتوای فایل a.txt عبارت است از:

```
// a.txt
├── example_file2_d1.tst
├── example_file_d1.tst
├── example_sub_dir_1
│   ├── file1.tst
│   └── file2.tst
├── example_sub_dir2_1
│   ├── example_sub_dir_2
│   └── file1_on_depth2.tst
└── example_sub_dir3_1
```

دقت کنید که به عنوان دستور اول قبل از علامت arman همه دستوراتی که خروجی می‌دهند می‌توانند قرار گیرند. این دستورات عبارتند از tree find، cat، grep، و ...

همچنین به عنوان دستور دوم بعد از علامت arman تمامی دستوراتی که ورودی رشته می‌گیرند و یا به عبارت دیگر در آن دستورات str– مشهود است می‌تواند قرار گیرد. در واقع این دستورات به عنوان دستور دوم می‌آیند و اما دیگر str– در آن دستور نمی‌آید و برنامه شما باید به صورت خودکار خروجی دستور قبلی را به عنوان ورودی رشته به دستور جدید بدهد. در مثال بالا این موضوع مشهود است.