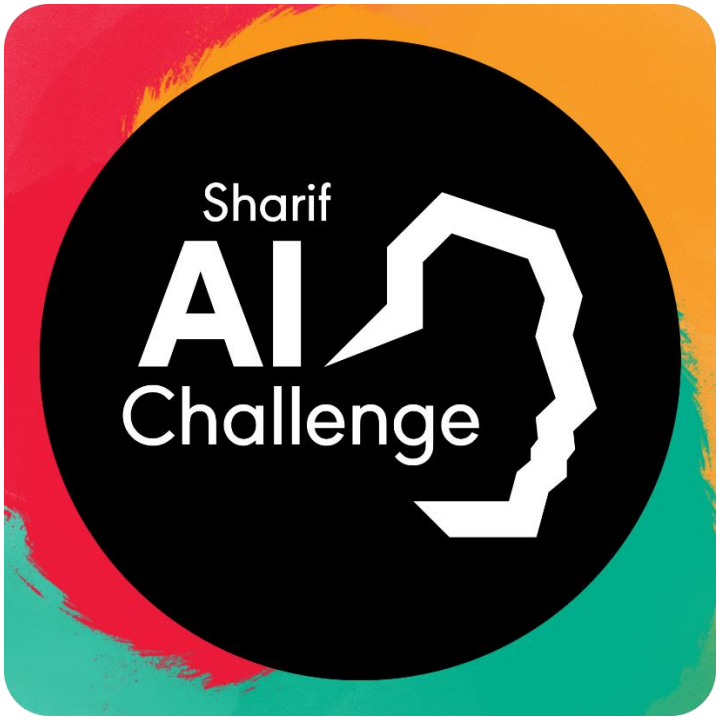
«به نام خدا»



یازدهمین نبرد هوش مصنوعی شریف
SHARIF AI CHALLENGE 2019
دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

زمستان ۱۳۹۷

محتوای فنی یازدهمین نبرد هوش مصنوعی شریف

«بخش دوم: راهنمای Debug»

سید سجاد کاهانی – سید علی هاشمی

فهرست مطالب

بخش اول : کار با IDE ها

دیباگ، احتمالاً قدمتی به عمرِ کامپیوتر یا حتی به عمرِ مهندسی دارد و به طورِ خلاصه می‌شود عیب‌یابی پس از ساختِ یک نمونه از یک چیز، که این‌جا چیزِ ما همان کدِ ماست ! Debugging به فرایند مشکل یابی یک نرم‌افزار یا اپلیکیشن گفته می شود.

زمانی که ما کدنویسی می کنیم، معمولاً در حین کدنویسی برخی خطاها را مرتکب می‌شویم که در نهایت منجر به این می‌شوند تا برنامه ی ما آن طور که باید و شاید کار نکند.تعبیرِ دیگری وجود دارد که راهگشای شیوه این عیب‌یابی‌ست، ما با فرض‌هایی که در خصوصِ داده‌های ورودی و عملکردِ سیستم‌های دیگر (مثل کتاب‌خانه‌ها) می‌کنیم، برنامه‌ای می‌نویسیم که حدس می‌زنیم با توجه به فرض‌ها خروجیِ مناسبی تولید کند، با اجرای آن روی کامپیوتر، مثالِ نقضی را می‌بینیم که یا به‌معنیِ اشتباه در فرض‌ها و یا به معنیِ اشتباه در فرایندِ اثبات است، حالا به بحث و مجادله با کامپیوتر می‌پردازیم تا بالاخره با تغییراتی در اثبات، آن را بپذیرد. این شیوه بی‌شباهت به روش سقراطی در مباحثه نیست.

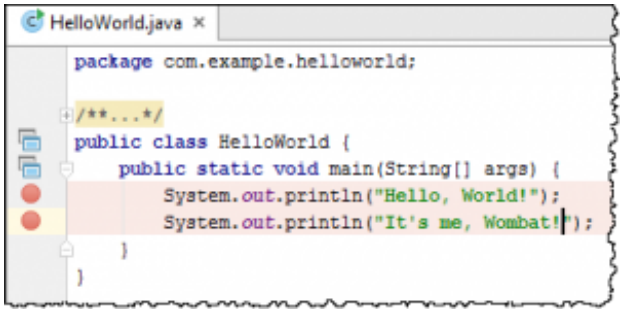
به طور کلی، ما معمولاً چند نوع مشکل در برنامه‌های خود داریم که عبارتند از:

اورورهای سینتکسی :به نوشتار کدهای یک زبان برنامه نویسی Syntax گفته می شود. گاهی اوقات برنامه نویسان در حین نوشتن برخی دستورات، غلط املایی مرتکب می شوند. مثلاً به جای نوشتن دستورprint ، می‌نویسند .pritn در برخی زبان‌ها مثل HTML و CSS مرتکب شدن چنین خطاهایی خیلی مشکل ساز نیست اما برخی از دیگر زبان‌ها مثل PHP و Python با اورورهای سنتکسی خیلی مشکل دارند و برنامه هرگز اجرا نخواهد شد.

اورورهای سمنتیک :این دست اورورها زمانی رخ می‌دهند که کدهای شما درست است اما نتیجه ی مطلوب را دریافت نمی‌کنید مثل زمانی که عددی را بر صفر تقسیم کنید (در ریاضیات چنین عملی امکان پذیر نیست!)

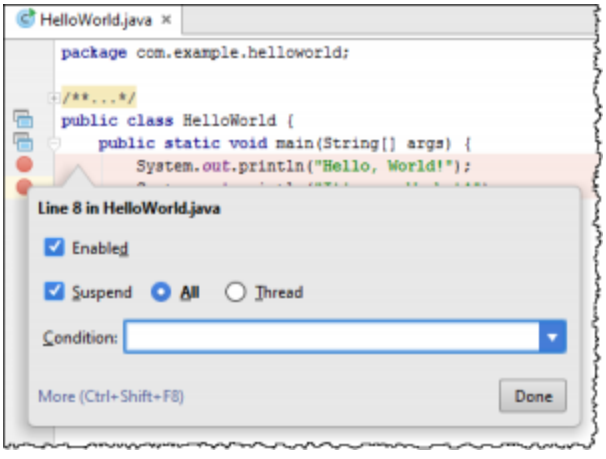
ارورهای منطقی :این دست از ارورها یا مشکلات جزو ارورهای سخت هستند و شاید یک برنامه نویس روزها و شاید هفته‌ها برای یافتن آن‌ها می بایست زمان بگذارد. سینتکس برنامه درست است و برنامه می بایست همان طور که انتظار می‌رود اجرا شود اما واقعیت این گونه نیست! فرض کنیم که یک فروشگاه آنلاین داریم و زمانی که مشتری به سبد خرید خود می رود، برنامه‌ای که ما نوشته‌ایم جمع سبد خرید را اشتباه در معرض دید وی قرار می دهد.

قرار دادن نقطه توقف(breakpoint): برای شروع دیباگ ابتدا باید یک breakpoint در دستوراتی که شما میخواهید در آن ها اجرای برنامه خود را متوقف کنید ، قرار دهید. در مثال زیر و کد موجود ما breakpoint را در ابتدای دو دستور قرار میدهیم.

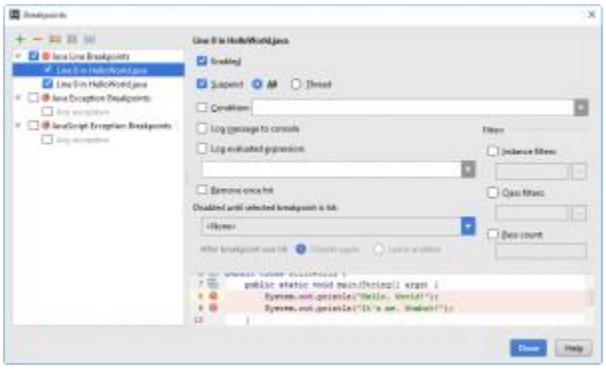


رنگ خط مشخص شده تغییر میکند. اگر ماوس را روی نقطه قرمز قرار دهید، خصوصیت آن در tooltip دیده میشود.

اگر میخواهید ویژگی های این breakpoint را تغییر دهید روی آن کلیک راست میکنید

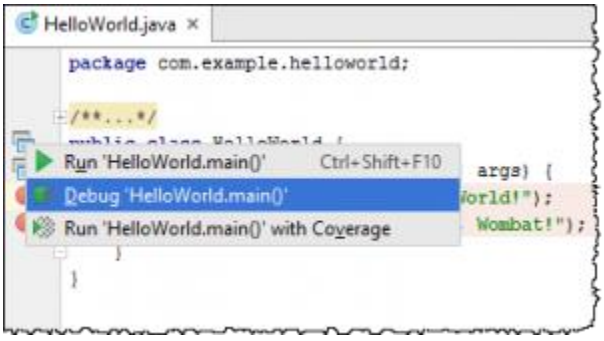


برای دیدن همه نقاط توقف و تغییر ویژگی های آنها کافیست Ctrl + Shift + F8 را فشار دهید



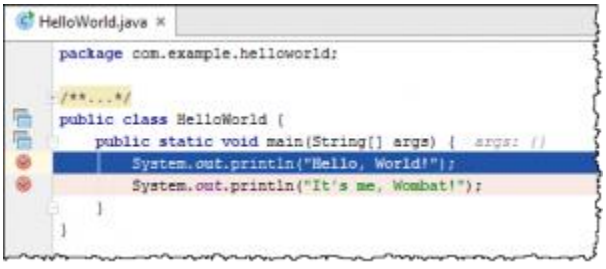
شروع دیباگ

با کلیک روی breakpoint منویی باز میشود که از طریق آن میتوانید کد را run یا debug کنید

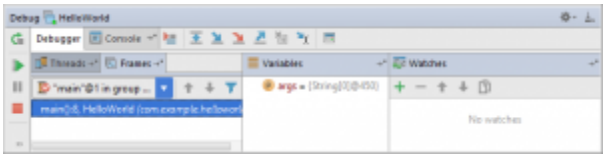


کد شما کامپایل میشود (ممکن است طول بکشد!!) و سپس برنامه در اولین breakpoint متوقف میشود. تغییراتی مشاهده

میشود از جمله تغییر رنگ خط breakpoint



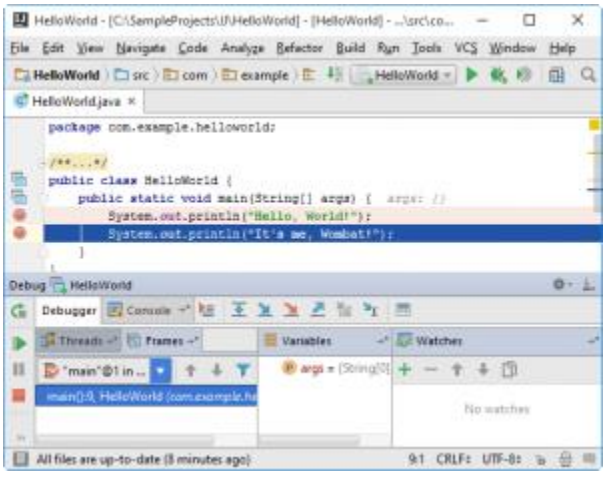
به این معنی است که برنامه تا قبل این خط اجرا شده و در این دستور معلق شده است. در قسمت پایین صفحه intellij یک پنجره به نام debug tool window ظاهر میشود که تمام ابزارهای مورد نیاز برای انجام دیباگ در آن قابل مشاهده است.



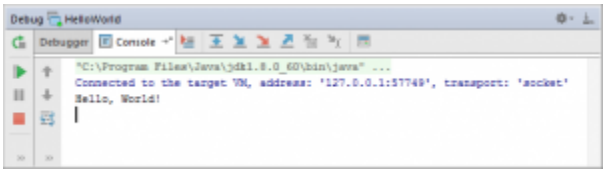
حال دیباگ را ادامه میدهیم. کافیست روی فلش آبی رو به پایین نوار ابزار در پنجره debug tool window کلیک کنید یا F8 را فشار دهید. اکنون خط بعدی آبی شده و تغییرات زیر اعمال میشود.

1- در کادر frames شماره خط بعدی نمایش داده میشود.

2- در کادر console یک آیکون زرد رنگ اضافه شده که نشان میدهد برنامه دارای یک خروجی جدید است.

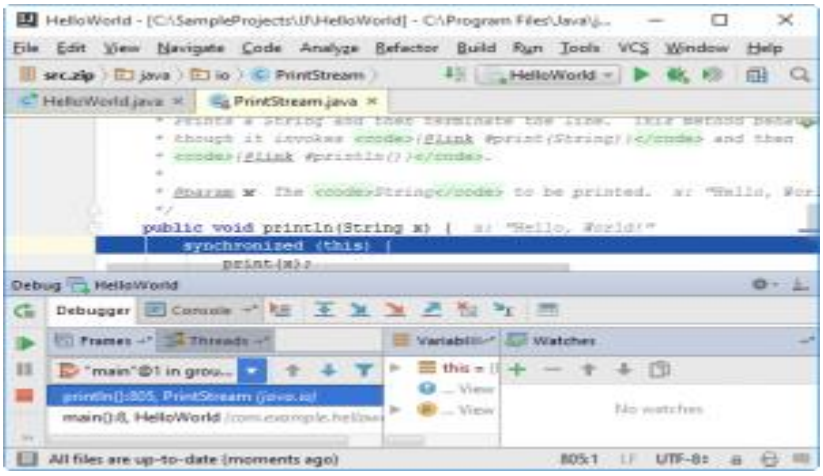


روی زبانه console (tab)!! کلیک کنید تا خروجی را ببینید



با فشردن F8 یا کلیک روی فلش آبی رو به پایین دستور بعدی هم اجرا شده و با تکرار این کار برنامه پایان میابد.

حال روشی با اندکی پیچیدگی بیشتر معرفی میکنیم! کافیست پس از شروع دیباگ و رسیدن به breakpoint دکمه های Shift+Alt+F7 را فشار دهید تا به درون تابع println در کلاس PrintStream برویم.



حال با فشردن Shift + F8 به نقطه توقف بعدی میرویم.

خب این یک جلسه خییلیی ساده از دیباگ بود! قطعا کدهای شما اینقدر آسان نیست :)

کارهای مختلفی که میتوان انجام داد :

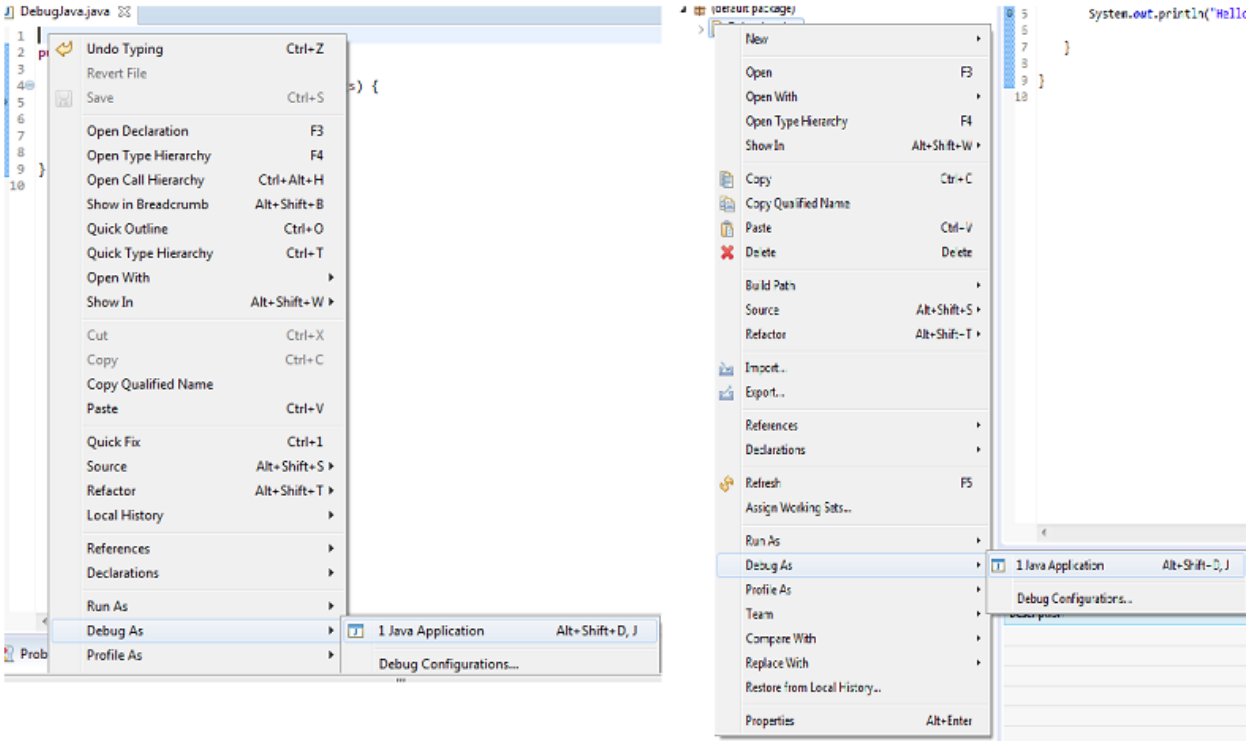
متوقف کردن دیباگ: از طریق منوی اصلی گزینه های **Run | Pause Program** را انتخاب کنید. / در نوار ابزار دیباگ، ⏸ را کلیک کنید.

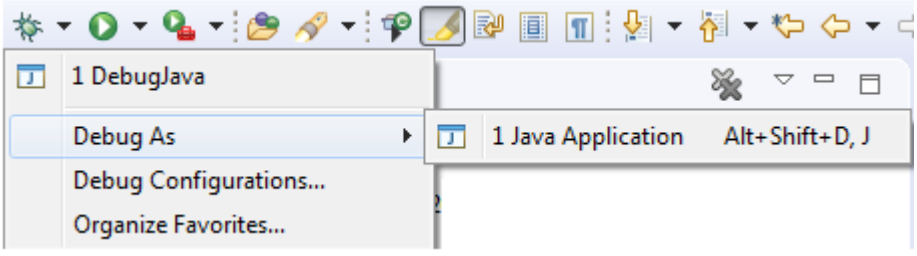
ادامه دادن دیباگ:از طریق منوی اصلی گزینه های **Run | Resume Program** را انتخاب کنید. / در نوار ابزار ▶ را کلیک کنید. / دکمه F9 را فشار دهید.

Eclipse

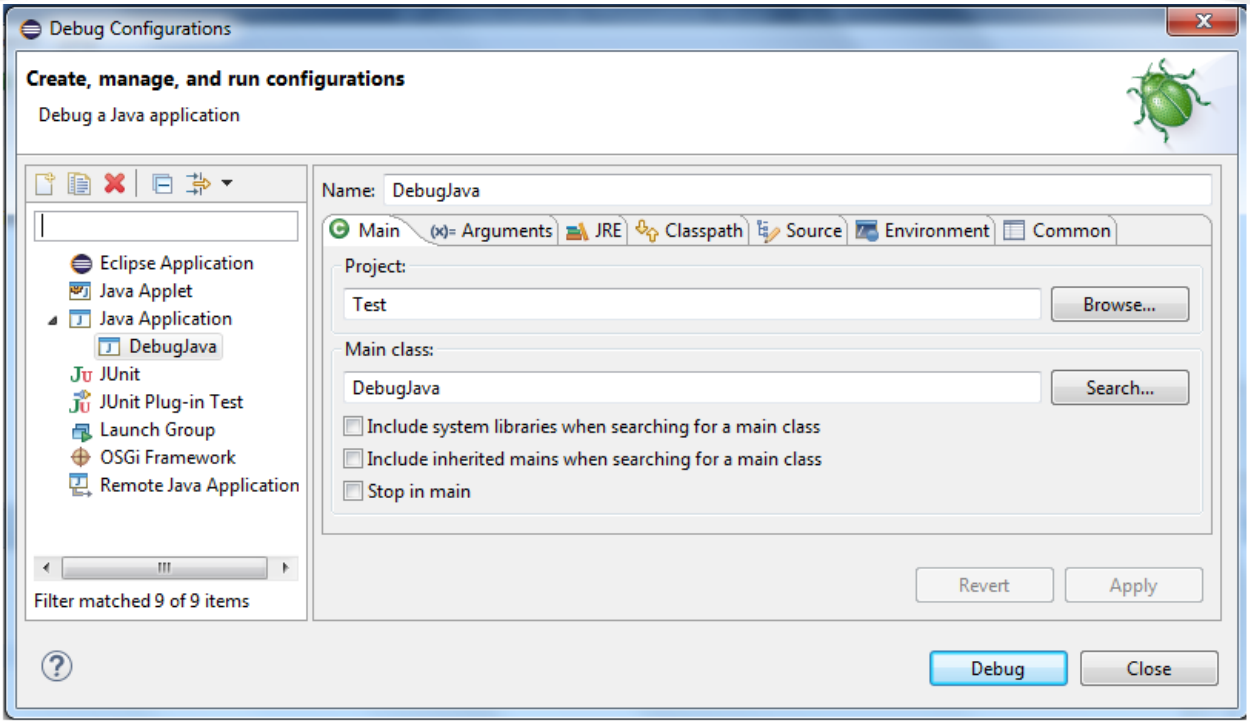
برای شروع دیباگ کافیست روی نام کلاس مدنظر در قسمت Package explorer راست کلیک کنید و گزینه

Debug As → Java Application یا Alt + Shift + D, J را فشار دهید.

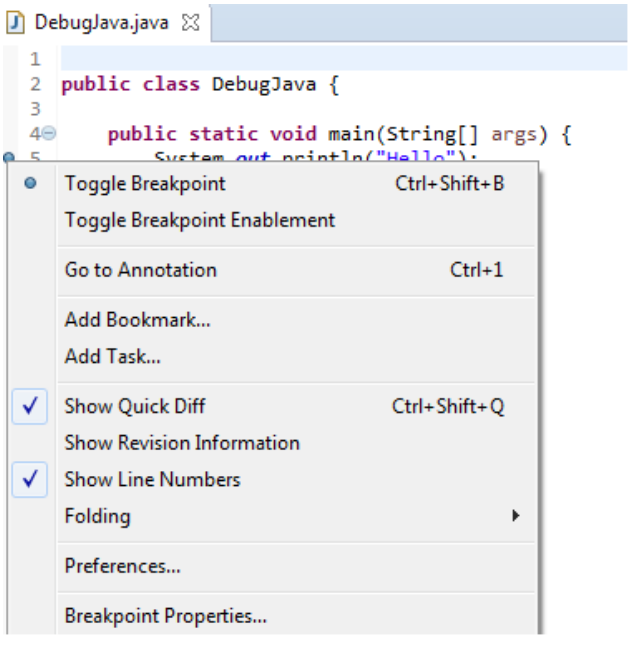




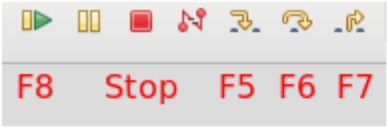
هریک از کارهای فوق یک **Debug Launch Configuration** جدید ساخته و با استفاده ازآن میتوان دیباگ را شروع کرد.



حال باید breakpoint بسازیم. (مانند jetbrains ها !) با دوبار کلیک کردن بر خط موردنظر یا کلیک راست و انتخاب Toggle Breakpoint میتوان روی خط مورد نظر breakpoint ساخت.



ابزارهای زیر در eclipse برای دیباگ کردن تعبیه شده اند.



Shortcut	Toolbar	Description
F5 (Step Into)		Steps into the call
F6 (Step Over)		Steps over the call
F7 (Step Return)		Steps out to the caller
F8 (Resume)		Resumes the execution
Ctrl + R (Run to Line)		Run to the line number of the current caret position
Drop to Frame		Rerun a part of your program
Shift + F5 (Use Step Filters)		Skipping the packages for Step into
Ctr + F5 / Ctrl + Alt + Click		Step Into Selection

NetBeans

بعید میدونم تعداد افرادی که از این ide استفاده میکنن زیاد باشه! درکل ide خوبی هست و امکانات گرافیکی خوبی داره. صرفا

لینک یوتیوب زیر رو معرفی میکنیم که خیلی کوتاه دیباگ در NetBeans رو بررسی کرده. (باشد که رستگار شوید!)

https://www.youtube.com/watch?v=2Z9B8wYhKWw

بخش دوم : کار با GDB

GNU Debugger

یک ابزارِ مشهور برای دیباگ کردن است که از قضا از قدیمی‌ترین ابزارها نیز هست. از این ابزار می‌توان برای گستره وسیعی از کارها از دیباگ تا مهندسیِ معکوس و disassemble استفاده کرد.

بگذارید یک‌باره کار با این ابزار را شروع کنیم.

یک کدِ سی/سی++ در اختیار داریم که آن‌را با دستورِ gcc به این شکل کامپایل می‌کنیم

```
gcc -o output file1.c file2.c …
```

با اضافه کردنِ آرگومان

```
-g
```

پس از gcc، این‌بار پروژه‌مان در وضعیتِ دیباگ کامپایل می‌شود.

کامپایل شدنِ کد در وضعیت دیباگ، اطلاعاتِ کد، نظیر نام متغیرها و خطوط کد و نگاشتِ آن‌ها به دستوراتِ زبانِ ماشین نیز داخلِ فایلِ باینریِ output قرار می‌گیرد و باعث می‌شود دیباگ کردنِ آن راحت‌تر شود. (پس یعنی بدونِ این‌کار نیز دیباگ ممکن است)

حالا برای شروعِ دیباگ می‌توانیم توی ترمینال بنویسیم

```
gdb output
```

یا آدرسِ هر باینریِ دیگری.

حالا واردِ محیطِ gdb شدیم، همان‌طور که احتمالاً تجربه کرده‌اید، فرایندِ دیباگ اغلب به این صورت است که چند breakpoint در کد قرار می‌دهیم تا در آن‌جا متوقف شوند، سپس کد را اجرا می‌کنیم، کد به خطوط مذکور می‌رسد و از آن‌جا می‌توانیم خط‌به‌خط کد را اجرا کنیم و بعد از هر خط مقادیرِ حافظه را بسنجیم.

برای انجامِ این فرایند در این محیط، ابتدا با دستورِbreakpoint یا به اختصار b، نقاطِ توقف را مشخص می‌کنیم

برای مثال

```
breakpoint file1.c:16
b file2.c:11
```

و اگر تنها یک فایل داشتیم

```
b 12
```

حالا، باید کد را اجرا کنیم تا در نقاط مذکور متوقف شود، این کار با دستور زیر انجام می‌شود.

```
run
```

پس از توقفِ دیباگر، دوباره می‌توانیم دستورهایی را وارد کنیم، مثلاً

```
print arr[2] / v[i]
```

این دستور، همان‌طور که قابلِ حدس است، حاصلِ عبارتِ مذکور را محاسبه و چاپ می‌کند. با این‌روش می‌توانیم داده‌های درونِ حافظه را به شکلی که می‌خواهیم بررسی کنیم.

مرحله بعدی جابه‌جا شدن در کد است،

دستورِ

```
next
```

برنامه را در خطِ بعدی در همین فایل متوقف می‌کند

و دستورِ

step

برنامه را تا اولین خط از سورس در این فایل یا فایلِ دیگر از پروژه متوقف می‌کند، به این معنی که این دستور بیشتر عملکردی مشابهِ

Step In

در ابزارهای گرافیکی دارد و شما را به درونِ تابع‌هایی که فراخوانی می‌شوند می‌برد (درصورتی که تابع جزوی از پروژه باشد).
تا این‌جا چیزهایی که روی همه دیباگرهای معمول وجود داشت را بررسی کردیم، اما gdb به همین‌جا منتهی نمی‌شود. در ادامه دو موردِ کاربردیِ دیگر را از gdb بررسی می‌کنیم.

کار با دامپ

وقتی یک برنامه دچارِ خطای حینِ اجرا بشود معمولاً سیستم‌عامل همه اطلاعاتِ موجود در حافظه را ذخیره می‌کند تا بعداً بتوان برای عیب‌یابی از آن استفاده کرد. به این اطلاعاتِ ذخیره شده core dump می‌گویند.
برای مثال، یک برنامه را که با gcc با فلگِ دیباگ ((-gکامپایل شده اجرا می‌کنیم و در حین اجرا دچار خطایی نظیر بیرون زدن از آرایه یا تقسیم بر صفر می‌شود. در سیستم‌های جدیدی که احتمالاً شاملِ همه سیستم‌های لینوکسیِ اطراف ما می‌شوند می‌توانیم با استفاده از دستورِ زیر به core dump مذکور دسترسی داشته‌باشیم

coredumpctl gdb output

که در آن output نامِ فایلِ اجرایی برنامه است

یا می‌توانیم از طریق دیگری آن را مشخص کنیم

coredumpctl gdb 6653

که در آن عدد 6653، pid یا شناسه برنامه حین اجرا بوده که معمولاً در هنگام بروز خطا برنامه آن را چاپ می‌کند.

با زدنِ این دستور به محیط gdb وارد می‌شوید، اما این‌بار دیگر نمی‌توان از دستورات run و next و step استفاده کرد.

ابتدا با دستور backtrace یا bt، می‌توانیم بفهمیم در لحظه خطا، لیستِ فراخوانی‌های تودرتو چه بوده‌است یا به زبانِ ساده‌تر، خطا در کدام خط از کدام تابع روی داده و چه تابعی در کدام خط این تابع را فراخوانی کرده و... همین‌طور تا آخر.
حالا با استفاده از دستور frame به شکل زیر، یکی از این تابع‌های فراخوانی‌شده از لیست bt را انتخاب می‌کنیم

frame 2

که در آن عدد شماره تابع در bt است.
حالا با دستور list می‌توانیم کدِ چندخط مجاور را ببینیم، با دستور

info locals

متغیرهای محلی در تابع مذکور را ببینیم و با دستور

print x

مقدار متغیری مانند x را ببینیم.

مفهوم Watchpoint

یکی از قابلیت‌های دیگرِ gdb، توقف داده، watchpoint یا data breakpoint است. با این قابلیت می‌توان برنامه را هنگامِ نوشتن و تغییر دادنِ آن بخش از حافظه متوقف کرد. این کار را می‌توان با دستور زیر انجام داد

```
watch x
```

برنامه را هربار که مقدار x متغییر می‌کند متوقف می‌کند.

لینک ها و منابع بیشتر

https://www.aparat.com/v/bkGuK/قسمت_اول_-_gdb_آموزش_کار_با

https://www.eclipse.org/community/eclipse_newsletter/2017/june/article1.php

https://www.jetbrains.com/help/idea/debugging-code.html

https://stackify.com/java-debugging-tips/