آزمون نرم افزار – بخش۲–۴

# پوشش گراف برای عناصر طراحی

صدیقه خوشنویس
دانشگاه آزاد اسلامی – واحد شهرقدس

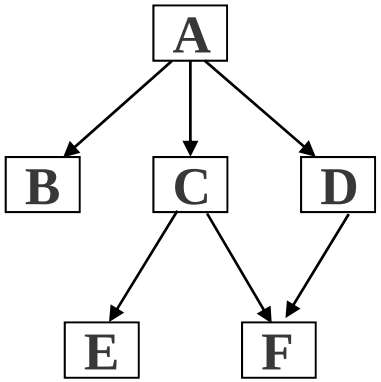
# نرم افزار شیء گرا و آزمون

- آزمایش نرم افزار بر اساس عناصر طراحی
- تأکید شیءگرایی بر پیمانه‌بندی موجب ایجاد پیچیدگی در اتصال عناصر طراحی می‌شود.
- اتصال عناصر، مربوط به آزمایش یکپارچه سازی (integration testing) است.
- بر اساس عناصر و اتصال میان آنها، یک گراف ایجاد می کنیم.

# پوشش گراف ساختاری برای عناصر طراحی

- بر اساس عناصر و اتصال میان آنها، یک گراف ایجاد می کنیم.
- اتصال (coupling) = ارتباط وابستگی بین دو واحد نرم افزاری

- گراف فراخوانی: رایج ترین گراف برای آزمایش عناصر طراحی
  - گره های این گراف: واحدهای نرم افزاری (مثلاً متدها)
  - یال های این گراف: فراخوانی این واحدها از یکدیگر



نمونه ای از گراف فراخوانی

پوشش گره: هر واحد را حداقل یک بار فراخوانی کنید (پوشش متد).

پوشش یال: هر فراخوانی را حداقل یک بار اجرا کنید (پوشش فراخوانی).

# گراف فراخوانی برای کلاسها

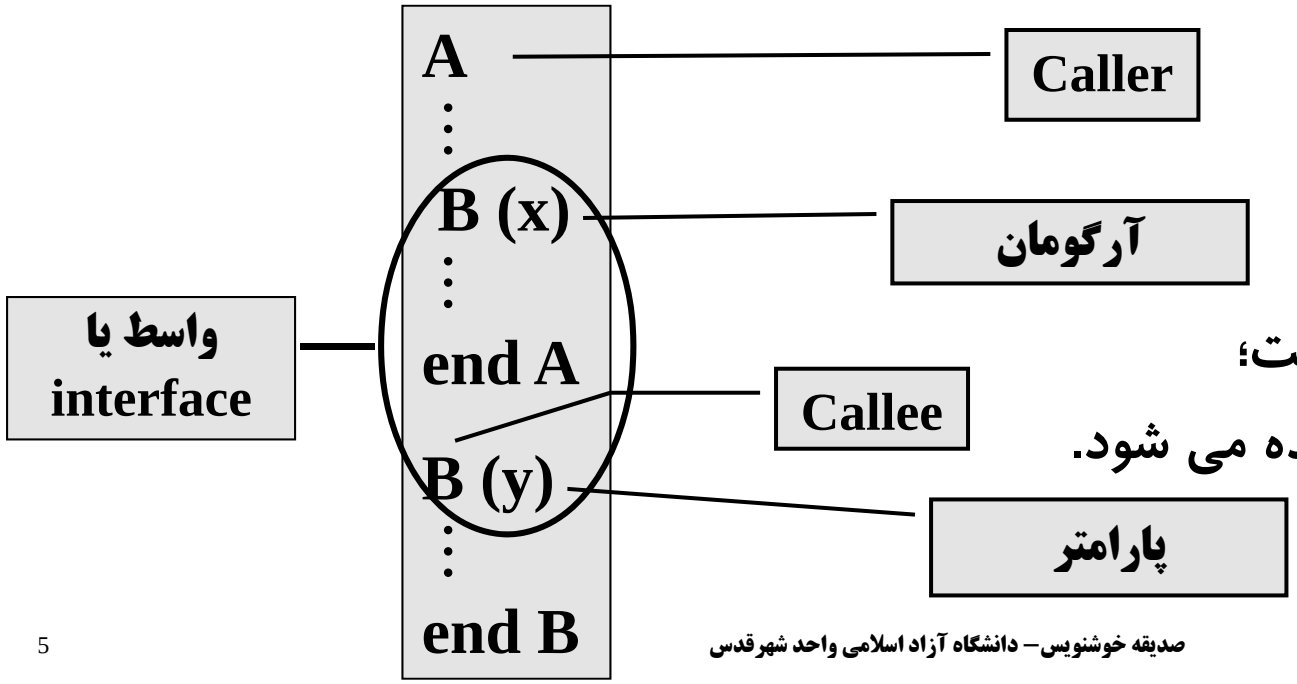
- ماژول مجموعه‌ای است از واحدهای مرتبط به هم (مثلاً کلاسها یا متدهای مرتبط به هم)
- پوشش گره و یال روی گراف فراخوانی کلاسها معمولاً کارآمد نیست.
- زیرا ممکن است لزوماً برخی متدها فراخوانی نشوند.

- آزمایش جریان داده (DU) مناسب تر است.

- آزمایش DU روی گراف فراخوانی، نسبت به گراف CFG پیچیده تر است.
  - وقتی متغیری از یک متد به متد دیگر پاس می شود، تغییر نام می دهد.
  - تعیین defها و useها سخت تر است (در متدهای مختلف هستند نه در یک متد).
  - باید تعیین کنیم کدام defها به کدام useها می رسند؟ (reach)
- اما درست وقتی نرم افزار پیچیده می شود، آزمونگر باید کنجکاوتر باشد! زیرا نقص احتمالاً همانجاست.

# آزمایش DU در سطح طراحی

- فراخوانی کننده (Caller): واحدی که واحد دیگر را فراخوانی می کند.
- فراخوانی شونده (Callee): واحدی که فراخوانی می شود.
- محل فراخوانی (Call Site): دستور یا گره ای که فراخوانی در آن واقع شده است.
- آرگومان (پارامتر واقعی): متغیر در caller
- پارامتر (پارامتر رسمی): متغیر در callee




- ما در آزمایش یکپارچه سازی هستیم و فقط واسط برایمان مهم است؛ وگرنه این تست خیلی پیچیده می شود.

# زوج های DU بین متدی

- وقتی بر واسط متمرکز می شویم، فقط به دو مورد زیر نیاز داریم:
  - آخرین **def** های متغیر قبل از فراخوانی ها و **return**ها
  - اولین **use** های متغیر داخل متد و بعد از فراخوانی ها

- **Last-Def (آخرین def)**: مجموعه گره‌هایی که در آنها، **x**، **def** شده و از **call-site** به یک **use** در متد دیگر مسیر **def-clear** دارند.
  - می تواند از **caller** به **callee** باشد (با استفاده از پارامتر یا متغیر سراسری یا مشترک)
  - می تواند از **callee** به **caller** باشد (به عنوان مقدار **return** شده)

- **First-Use (اولین use)**: مجموعه گره‌هایی که در آنها، **x**، **use** شده و از **call-site** به آن گره یک مسیر **def-clear** و **use-clear** وجود دارد.

# مسیرهای DU اتصال (Coupling DU paths)

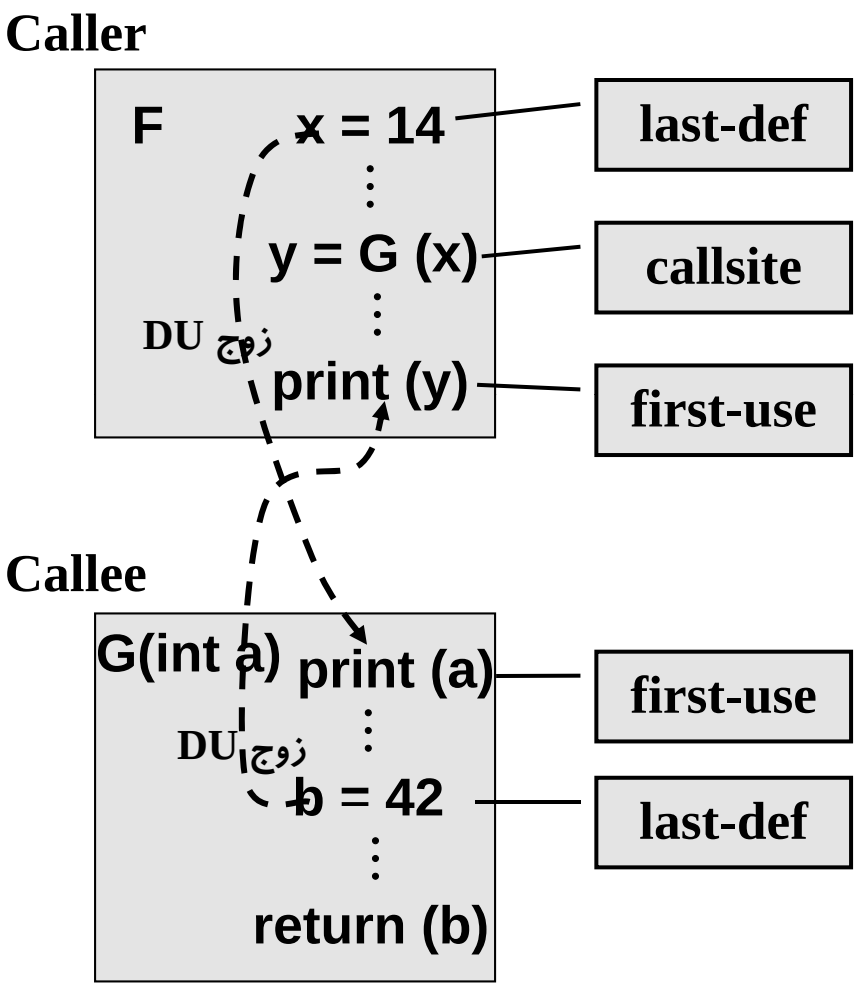
- مسیر DU اتصال برای متغیر x = مسیر از last-def به first-use متغیر x
  - یادآوری: ممکن است متغیر، بین caller و callee همنام نباشد.
  - حالا مثل مسیرهای DU عادی، می توانیم معیارهای پوشش DU را روی این مسیرها داشته باشیم.

- پوشش all-defs که نام جدید آن را all-coupling-defs می گذاریم: نیازمند آن است که یک مسیر از هر last-def به حداقل یک first-use اجرا شود.

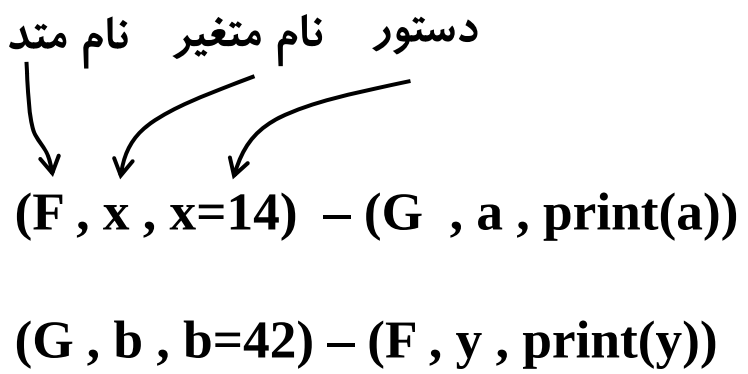
- پوشش all-uses که نام جدید آن را all-coupling-uses می گذاریم: نیازمند آن است که یک مسیر از هر last-def به هر first-use اجرا شود.

- پوشش all-du-paths که نام جدید آن را all-coupling-du-paths می‌گذاریم: نیازمند آن است که هر مسیر از هر last-def به هر first-use اجرا شود.

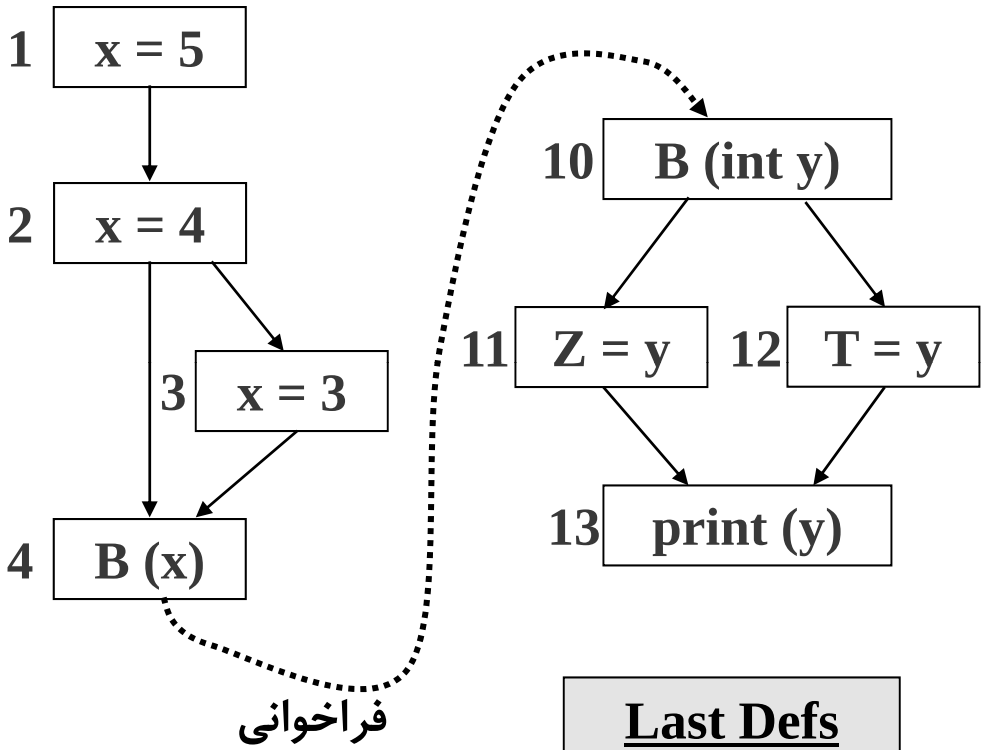
# مثال (۱)




زوجهای DU اتصال:



# مثال (۲)




**Last Defs**
گره های 2, 3

**First Uses**
گره های 11, 12

زوجهای DU اتصال:

(A , x , 2) – (B , y , 11)
(A , x , 2) – (B , y , 12)
(A , x , 3) – (B , y , 11)
(A , x , 3) – (B , y , 12)

# مثال (۳) – برنامه ریشه دوم (quadratic)

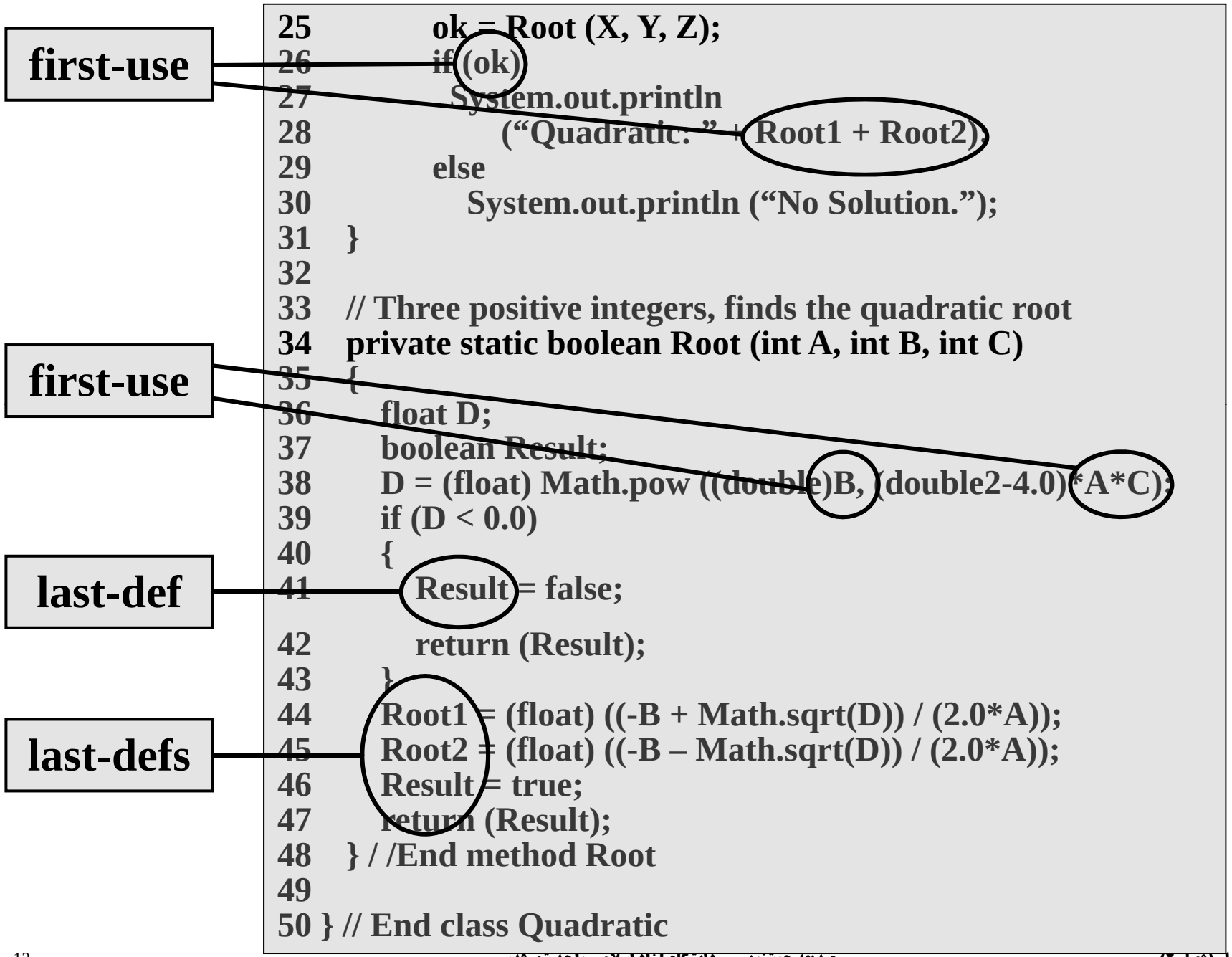
```
1 // Program to compute the quadratic root for
two numbers
2 import java.lang.Math;
3
4 class Quadratic
5 {
6  private static float Root1, Root2;
7
8  public static void main (String[] argv)
9  {
10     int X, Y, Z;
11    boolean ok;
12    int controlFlag = Integer.parseInt (argv[0]);
13    if (controlFlag == 1)
14    {
15        X = Integer.parseInt (argv[1]);
16        Y = Integer.parseInt (argv[2]);
17        Z = Integer.parseInt (argv[3]);
18     }
19    else
20    {

21         X = 10;
22         Y = 9;
23         Z = 12;
24     }
25         ok = Root (X, Y, Z);
26         if (ok)
27           System.out.println
28               (“Quadratic: ” + Root1 + Root2);
29         else
30            System.out.println (“No Solution.”);
31   }
32
33 // Three positive integers, finds quadratic root
34   private static boolean Root (int A, int B, int C)
35   {
36      float D;
37      boolean Result;
38      D = (float) Math.pow ((double)B,

             (double2-4.0)*A*C );
39      if (D < 0.0)
40      {
41         Result = false;

42         return (Result);
43      }
44      Root1 = (float) ((-B + Math.sqrt(D))/(2.0*A));
45      Root2 = (float) ((-B – Math.sqrt(D))/(2.0*A));
46      Result = true;
47      return (Result);
48   } / /End method Root
49
50   } // End class Quadratic
```

# مثال(۳) – برنامه ریشه دوم (quadratic)

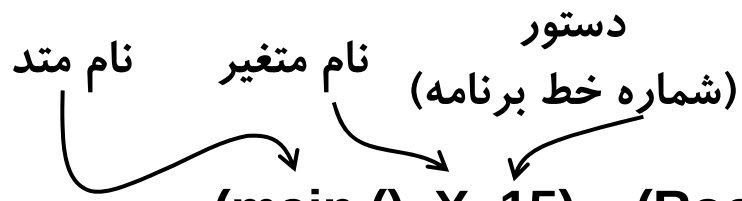
```
1 // Program to compute the quadratic root for two numbers
2 import java.lang.Math;
3
4 class Quadratic
5 {
6  private static float Root1, Root2;
7
8  public static void main (String[] argv)
9  {
10     int X, Y, Z;
11    boolean ok;
12    int controlFlag = Integer.parseInt (argv[0]);
13    if (controlFlag == 1)
14    {
15       X = Integer.parseInt (argv[1]);
16       Y = Integer.parseInt (argv[2]);
17       Z = Integer.parseInt (argv[3]);
18     }
19    else
20    {
21       X = 10;
22       Y = 9;
23       Z = 12;
24     }
```

متغیرهای مشترک
(به دلیل استاتیک بودن)

last-defs


first-use
first-use
last-def
last-defs
25 ok = Root (X, Y, Z);
26 if (ok)
27 System.out.println
28 (“Quadratic: ” + Root1 + Root2);
29 else
30 System.out.println (“No Solution.”);
31 }
32
33 // Three positive integers, finds the quadratic root
34 private static boolean Root (int A, int B, int C)
35 {
36 float D;
37 boolean Result;
38 D = (float) Math.pow ((double)B, (double2-4.0)*A*C);
39 if (D < 0.0)
40 {
41 Result = false;
42 return (Result);
43 }
44 Root1 = (float) ((-B + Math.sqrt(D)) / (2.0*A));
45 Root2 = (float) ((-B – Math.sqrt(D)) / (2.0*A));
46 Result = true;
47 return (Result);
48 } / /End method Root
49
50 } // End class Quadratic

# مثال(۳) – زوج های DU اتصال در quadratic




(main (), X, 15) – (Root (), A, 38)
(main (), Y, 16) – (Root (), B, 38)
(main (), Z, 17) – (Root (), C, 38)
(main (), X, 21) – (Root (), A, 38)
(main (), Y, 22) – (Root (), B, 38)
(main (), Z, 23) – (Root (), C, 38)
(Root (), Root1, 44) – (main (), Root1, 28)
(Root (), Root2, 45) – (main (), Root2, 28)
(Root (), Result, 41) – ( main (), ok, 26 )
(Root (), Result, 46) – ( main (), ok, 26 )

# خلاصه نکات پوشش DU اتصال

- **نکته۱: نقاط مهم برنامه : ۱- call site ها در caller ۲- returnها در callee**
- **نکته۲: متغیرهای مهم:**
  - **۱- آرگومانها در call site (که در callee دچار تغییر نام می شوند)**
  - **۲- متغیرهای return (که در caller دچار تغییر نام می شوند)**
  - **۳- متغیرهای سراسری یا استاتیک (که دچار تغییر نام نمی شوند)**
- **نکته۳: برای هر متغیر مهم به شکل زیر عمل می کنیم:**
  - **۱- قبل از هر callsite یا return، last-defهای آن را پیدا می کنیم.**
  - **۲- بعد از هر callsite یا return، first-useهای آن را پیدا می کنیم.**
  - **۳- با last-defها و first-useهای مرتبط به هم (با در نظر گرفتن تغییر نام های احتمالی) زوج du اتصال تشکیل می دهیم.**
  - **۴- TR را با توجه به پوشش خواسته شده و زوج های مرحله ۳ (مرحله بالا) تشکیل می دهیم.**
- **نکته۴: پوشش ها مانند du است:**
  - **پوشش all-defs اتصال: از هر last-def به یک first-use از یک مسیر**
  - **پوشش all-uses اتصال: از هر last-def به هر first-use از یک مسیر**
  - **پوشش all-du اتصال: از هر last-def به هر first-use از همه مسیرها**

# سایر نکات مهم...

- فقط متغیرهایی را در نظر بگیرید که در callee، def یا use شده.

- تراگذاری را در آزمون DU اتصال بررسی نمی کنیم زیرا بسیار گران است.
  - اگر A، B را فراخوانی کند؛ و B نیز C را فراخوانی کند، آنگاه متغیری، در A، def شده و در C، use می شود (زوج DU در اثر خاصیت تراگذاری)

- دسترسی به یک عنصر از یک آرایه، چه def باشد و چه use، دسترسی به کل آرایه محسوب می شود.

# پایان جلسه چهارم