پاسخ تمرین سری هفتم مبانی سیستم‌های نهفته و بیدرنگ

۱. مسئله‌ی ۲۸:

a.
```
x = a + b;
if (x < 20) proc1();
else {
    y = c + d;
    while (y < 10)
        y = y + e;
}
```

b.
```
r = 10;
s = a - b;
for (i = 0; i < 10; i++)
    x[i] = a[i] * b[s];
```

c.
```
x = a - b;
y = c - d;
z = e - f;
if (x < 10) {
    q = y + e;
    z = e + f;
}
if (z < y) proc1();
```

مسئله‌ی ۲۹:

a)
```
x = a + b;
if (x < 20) proc1();
```

برای اینکه شرط برقرار شود، x باید کمتر از ۲۰ باشد، برای مثال مقادیر a = 1 و b = 2 مناسب هستند.

```
y = c + d;
while (y < 10)
    y = y + e;
```

برای اینکه حلقه حداقل یک بار اجرا شود، y باید حداقل یک بار کمتر از ۱۰ باشد، مقادیر c = 1 و d = 2 مناسب هستند.

b)

```
r = 10;
s = a - b;
for (i = 0; i < 10; i++)
    x[i] = a[i] * b[s];
```

برای s: باید در رنج مقادیر ممکن آرایه‌ی b قرار داشته باشد. یعنی باید سایز b برابر s + 1 باشد و s از صفر بزرگتر باشد. بنابراین با فرض اینکه سایز b بزرگتر از ۲ است، ، مقادیر b = 1 و a = 2 مناسب هستند.

برای i: با توجه به اینکه مقدار اولیه‌ی i برابر صفر است، عبارت داخل حلقه حداقل یک بار اجرا می‌شود (به شرطی که سایز x و a حداقل یک باشد).

c)

```
x = a - b;
y = c - d;
z = e - f;
if (x < 10) {
    q = y + e;
    z = e + f;
}
if (z < y) proc1();
```

برای x:

برای اینکه شرط برقرار شود، x باید کمتر از ۱۰ باشد، برای مثال مقادیر a = 1 و b = 2 مناسب هستند.

برای y و z: برای اینکه عبارت q = y + e اجرا شود، x باید کمتر از ۱۰ باشد، برای مثال مقادیر a = 1 و b = 2 مناسب هستند.

اینکه شرط z < y برقرار باشد به مقدار y بستگی دارد که آن نیز به مقدار x بستگی دارد. به طور کلی حالات زیر برقرار است:

- اگر x < 10 باشد، خط z = e + f اجرا می‌شود، پس برای اینکه z < y برقرار باشد داریم:

z < y => e + f < c – d

بنابراین مقادیر a = 1 و b = 2 و e = 1 و f = 1 و d = 1 و c = 5 مناسب هستند.

- اگر x < 10 نباشد، خط z = e + f اجرا نمی‌شود مقدار z برابر e – f می‌ماند، پس برای اینکه z < y برقرار باشد داریم:

z < y => e - f < c – d

بنابراین مقادیر a = 1 و b = 2 و e = 2 و f = 1 و d = 1 و c = 5 مناسب هستند.

۲.

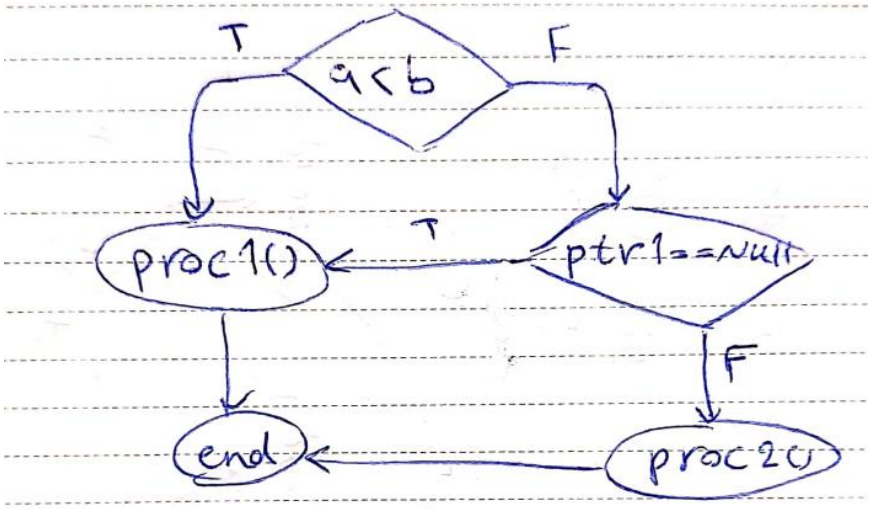
SF = SCC + global * 5 + SLOC/20

a.
```
if (a < b || ptr1 == NULL) proc1();
else proc2();
```

b.
```
switch (x) {
case 0: proc1(); break;
case 1: proc2(); break;
case 2: proc3(); break;
case 3: proc4(); break;
default; dproc(); break;
}
```

c.
```
if (a <5 && b > 7) proc1();
else if (a < 5) proc2();
else if (b > 7) proc3();
else proc4();
```

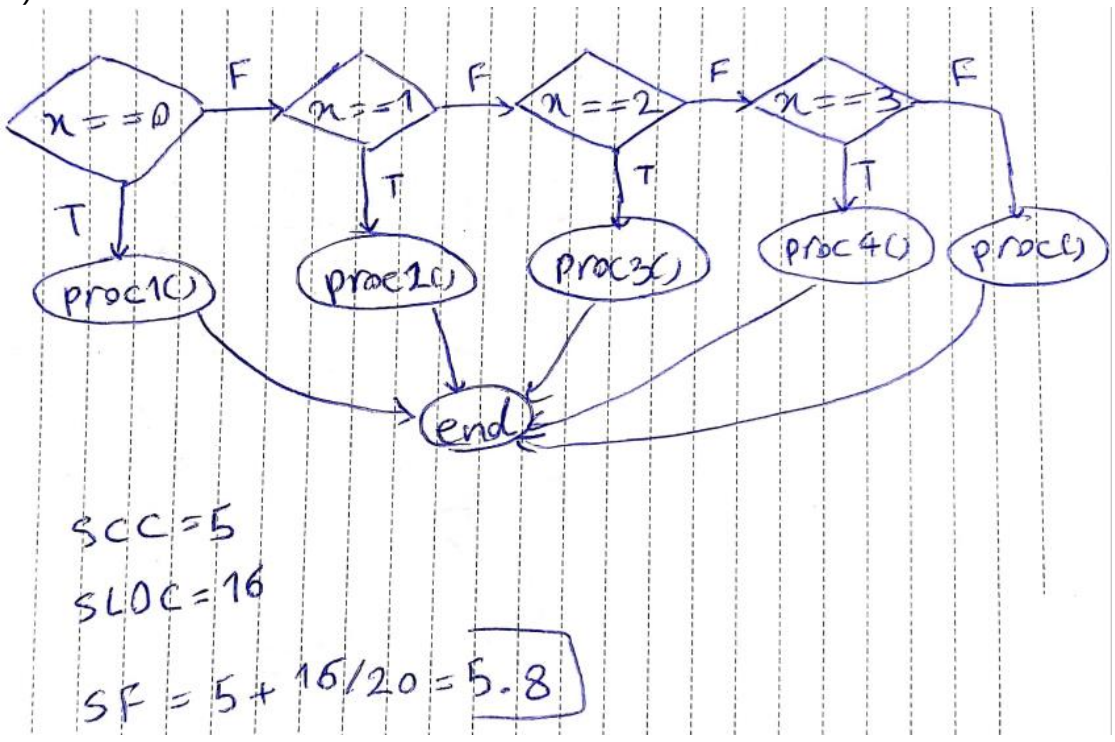
a)



scc = 2 + 1 = 3 sLOC = 4

⇒ SF = 3 + 0x5 + 4/20 = 3.2

b)

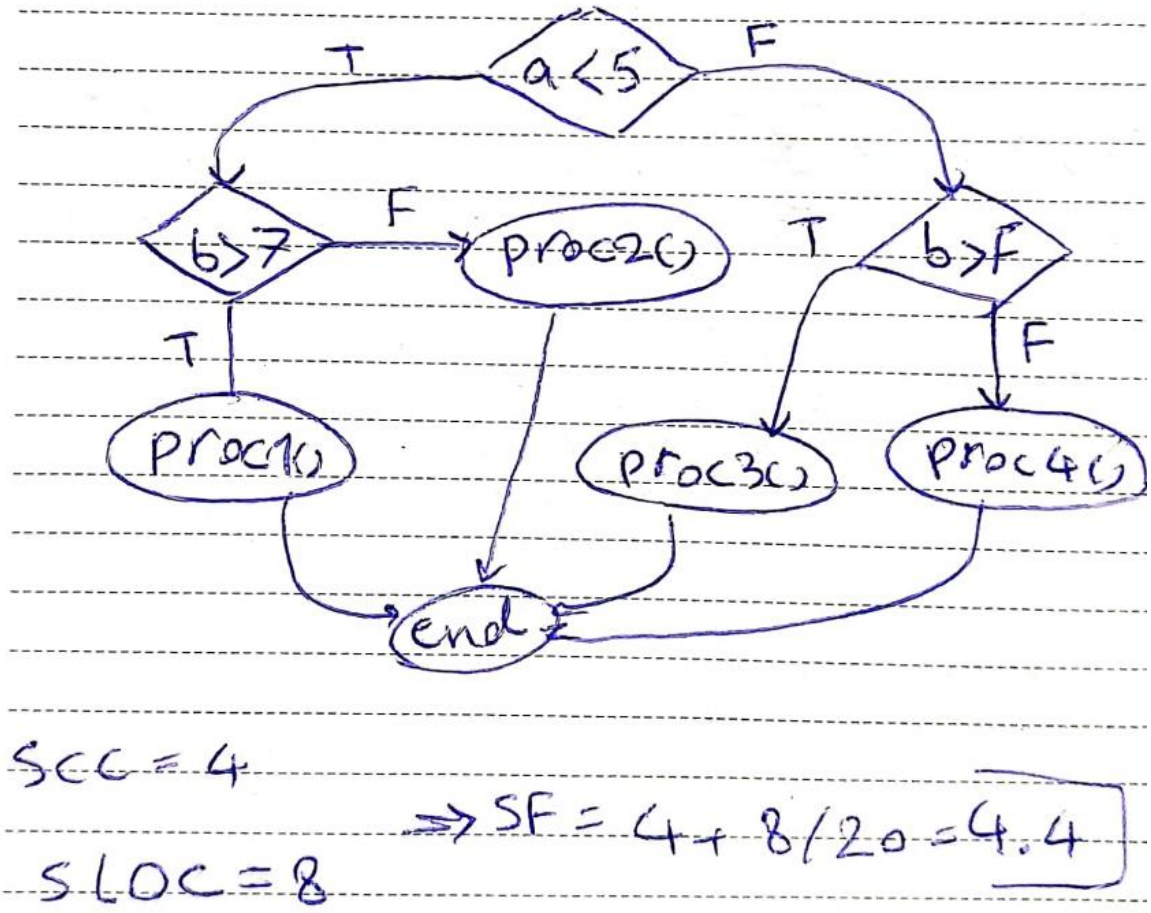


SCC = 5

SLOC = 16

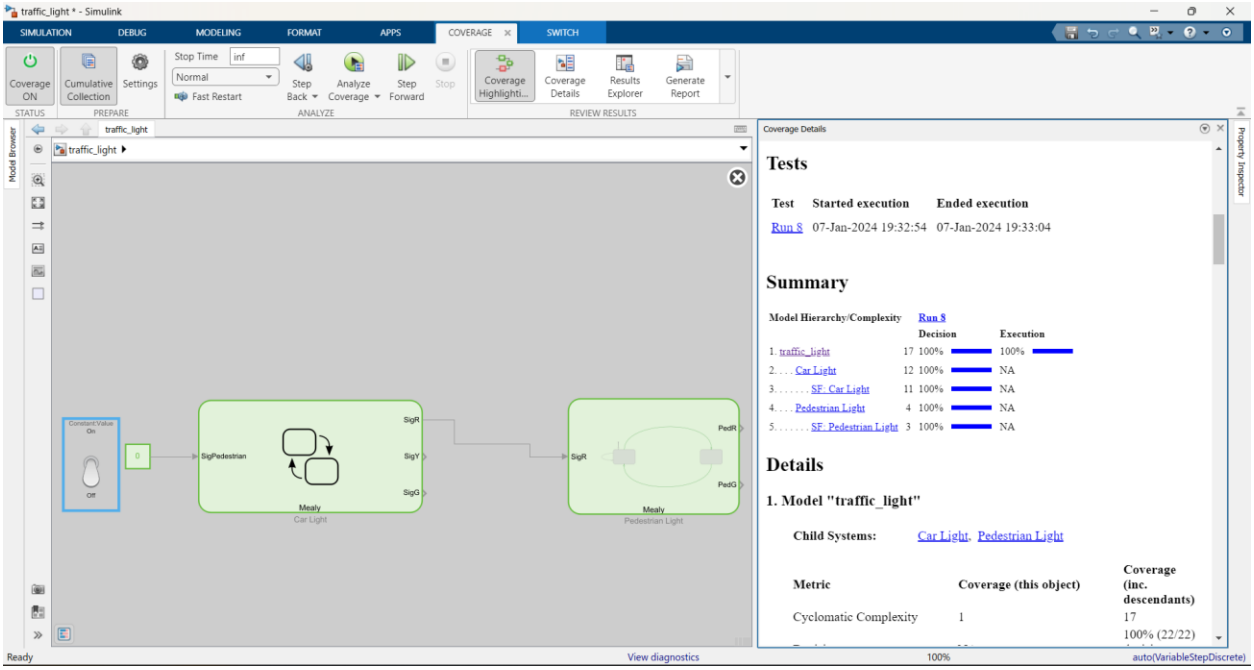
SF = 5 + 16/20 = 5.8

c)



۳. الف) ابتدا از بخش apps، coverage analyzer را انتخاب کرده، سپس analyze coverage را زده و سپس در زمان اجرا، حالت‌های مختلف اجرای سیستم را به سیستم ورودی می‌دهیم. در نهایت به بخش گزارش coverage می‌رویم (این گزارش درون فایل ارسالی پیوست شده است).

در ابتدای این گزارش، یک سری اطلاعات اولیه در مورد نام مدل، ورژن متلب به کار رفته و ... ذکر شده است. سپس در بخش summary، درصد اجرای حالت‌ها و گذارها را نشان داده است که آزمون ما چند درصد از این حالت‌ها و گذارها را پاس کرده است.



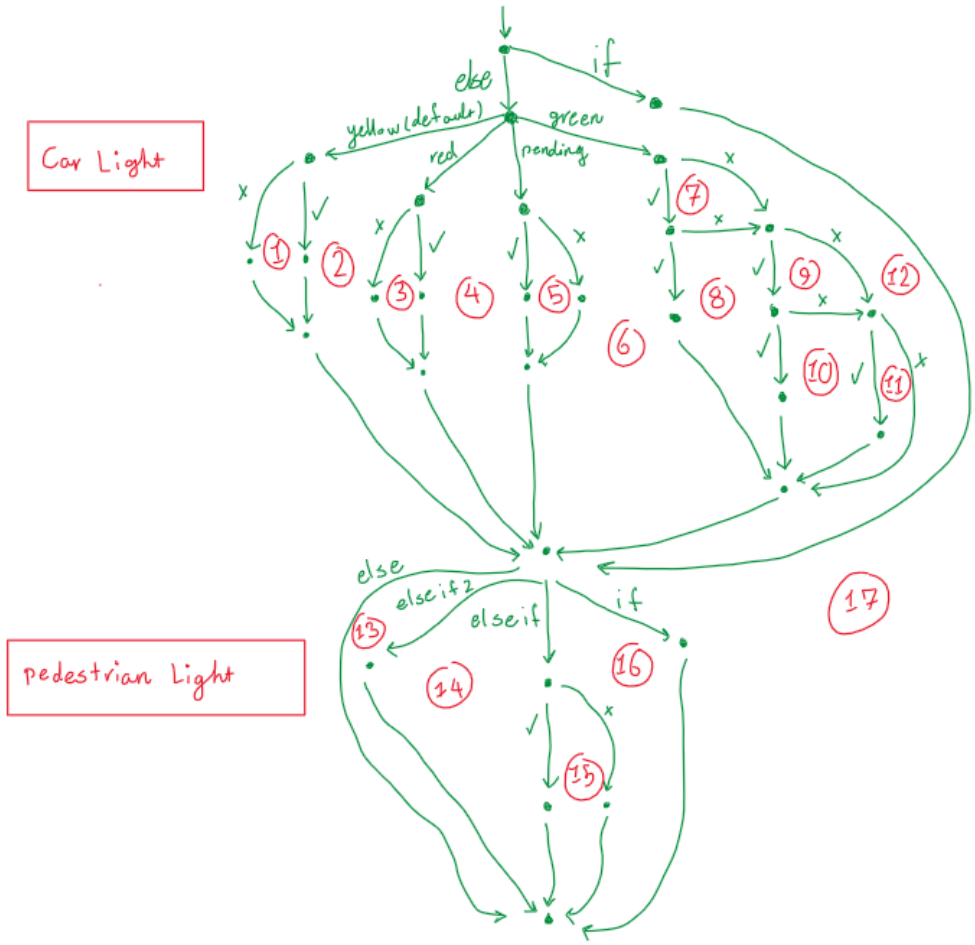
همانطور که قابل مشاهده است، اگر تمامی حالات ممکن را امتحان کنیم، از تمامی این گذارها می‌گذریم و coverage سیستم ما به 100% می‌رسد. علاوه بر این اطلاعات، complexity کد ما نیز قابل مشاهده است (که در این مدل برابر با 17 شده است).

سپس در بخش details، یک سری اطلاعات از جمله اجزای این سیستم و complexity این سیستم، جزئیات یک subsystem و coverage هرکدام از transitionها را به ما نشان می‌دهد.

ب) فرمول معیار SF به صورت روبه‌رو است:

$$SF = SCC + (Globals * 5) + (SLOC/20)$$

طبق کد ضمیمه شده در فایل ارسالی، control flow chart آن به صورت زیر می‌شود:



همانطور که در شکل نشان داده شده، تعداد حفره‌ها شمرده شده که برابر با ۱۶ می‌باشد. در نتیجه مقدار SCC برابر 16 + 1 = 17 می‌شود (که برابر با همین مقدار در گزارش coverage متلب سیمولینک است).

همچنین درون کد، ۱۱ متغیر global داریم: SigR، SigY، SigG، PedG، PedR، count، pcount، is_active_c3_traffic_light، is_c3_traffic_light، is_active_c1_traffic_light، is_c1_traffic_light

و تابع اصلی (traffic_light_step) نیز 75 خط کد است.

حال این مقادیر را درون فرمول جایگذاری می‌کنیم:

$$SF = 17 + (11 * 5) + \left(\frac{75}{20}\right) = 75.75$$

همانطور که مشاهده می‌شود، مقدار معیار *SF* برابر 75.75 شده است که مقدار خیلی بالایی است و کد باید refactor شود.

ج) در این بخش، طبق توضیحات داده شده، ***unity*** را نصب کردیم. سپس در بخش تست، چهار تست زیر را نوشتیم:

```
void tr_r_test(void) {

  TEST_ASSERT_EQUAL(traffic_light_IN_Red,traffic_light_DW.is_c3_traffic_light);
  TEST_ASSERT_EQUAL(traffic_light_IN_Green,traffic_light_DW.is_c1_traffic_light);
}

void tr_g_test(void) {

  TEST_ASSERT_EQUAL(traffic_light_IN_Green,traffic_light_DW.is_c3_traffic_light);
  TEST_ASSERT_EQUAL(traffic_light_IN_Red_c,traffic_light_DW.is_c1_traffic_light);
}

void tr_pend_test(void){

  TEST_ASSERT_EQUAL(traffic_light_IN_Pending,traffic_light_DW.is_c3_traffic_light);
  TEST_ASSERT_EQUAL(traffic_light_IN_Red_c,traffic_light_DW.is_c1_traffic_light);

}


void tr_y_test(void) {

  TEST_ASSERT_EQUAL(traffic_light_IN_Yellow, traffic_light_DW.is_c3_traffic_light);
  TEST_ASSERT_EQUAL(traffic_light_IN_Red_c, traffic_light_DW.is_c1_traffic_light);

}
```

تابع tr_r_test() بررسی می‌کند که آیا در سناریو مشخص شده درون استیت قرمز ماشین و استیت سبز چراغ عابر هستیم یا خیر.

تابع tr_g_test() بررسی می‌کند که آیا در سناریو مشخص شده درون استیت سبز ماشین و استیت قرمز چراغ عابر هستیم یا خیر.

تابع tr_pend_test() بررسی می‌کند که آیا در سناریو مشخص شده درون استیت pending ماشین و استیت قرمز چراغ هستیم یا خیر.

تابع tr_y_test() بررسی می‌کند که آیا در سناریو مشخص شده درون استیت زرد ماشین و استیت قرمز چراغ عابر هستیم یا خیر.

سپس در تابع *main*، این چهار تابع را در سناریو ذکر شده تست کردیم:

- چراغ ماشین در ابتدا قرمز است. در نتیجه چراغ عابر سبز است.
- بعد از ۶۰ تیک، چراغ ماشین سبز می‌شود. در نتیجه چراغ عابر باید قرمز شود.
- زمانی که چراغ عابر سبز است، قبل از ۶۰ تیک دکمه pedestrian زده می‌شود. پس باید وارد استیت pending شویم. چراغ عابر همچنان قرمز است.
- بعد از رسیدن به ۶۰ تیک، چراغ ماشین زرد می‌شود. چراغ عابر همچنان قرمز است.
- بعد از ۵ تیک، چراغ ماشین قرمز می‌شود. در نتیجه چراغ عابر سبز می‌گردد.

```
int main(){
  traffic_light_initialize();
  UNITY_BEGIN();

  // We're in Red state of Car Light.
  traffic_light_U.SigPedestrian = 0;
  traffic_light_DW.count = 0;
  traffic_light_DW.pcount = 0;
  traffic_light_step();
  RUN_TEST(tr_r_test);

  // 60 ticks finished. Then we head to Green state of Car Light.
  traffic_light_DW.count = 60;
  traffic_light_DW.pcount = 60;
  traffic_light_step();
  RUN_TEST(tr_g_test);

  /* After 60 ticks, Pedestrian button enabled.
  |  |  |  |  So we should go to Yellow state of Car Light */
  // traffic_light_DW.count = 60;
  // traffic_light_U.SigPedestrian = 1;
  // traffic_light_step();
  // RUN_TEST(tr_y_test);

  /* Before 60 ticks, Pedestrian button enabled.
  |  |  |  |  So we should go to Pending state of Car Light. */
  traffic_light_DW.count = 10;
  traffic_light_U.SigPedestrian = 1;
  traffic_light_step();
  RUN_TEST(tr_pend_test);

  // We reached 60 ticks. So we move from Pending state to Yellow state of Car Light.
  traffic_light_DW.count = 60;
  traffic_light_U.SigPedestrian = 0;
  traffic_light_step();
  RUN_TEST(tr_y_test);

  // After 5 ticks, we head back to Red state of Car Light.
  traffic_light_DW.count = 5;
  traffic_light_DW.pcount = 0;
  traffic_light_step();
  RUN_TEST(tr_r_test);

}
```

حال این تست‌ها را اجرا می‌کنیم:

```
------------------------------------------------------------------- SUMMARY -------------------------------------------------------------------
Environment     Test        Status    Duration
--------------  ----------  --------  ------------
megaatmega2560  test_light  SKIPPED
native          test_light  PASSED    00:00:00.998
============================================ 5 test cases: 5 succeeded in 00:00:00.998 ============================================
```

همانطور که مشاهده می‌شود، تست‌های داده شده پاس شده‌اند.