# CURS 05-06.
# TESTARE WHITE-BOX

**Verificare, validare şi testare automată**
**[19 Octombrie 2019]**

Lector dr. Camelia Chisăliţă-Creţu
Universitatea Babeş-Bolyai, NTT Data
**Programul Postuniversitar de Pregătire şi Formare Profesională în Informatică**

# Conţinut

- **Criterii de testare**
- **Testare White-Box**
  - Definiţie. Caracteristici
  - Tehnici de testare white-box
- **Testare bazată pe fluxul de control. Componente**
  - Definiţie. Caracteristici. Avantaje şi dezavantaje
  - Graful fluxului de control. Exemple
  - Drumuri în CFG. Exemple
  - Complexitatea ciclomatică. Exemple
- **Testare bazată pe acoperirea drumurilor**
  - Definiţie. Algoritm. Exemplu
- **Testare bazată pe acoperirea codului sursă**
  - Definiţie. Criterii de acoperire
  - Acoperirea instrucţiunilor, deciziilor, condițiilor, deciziilor şi condiţiilor, condiţiilor multiple, buclelor
- **Testare White-box vs Testare Black-box**
  - Avantaje şi dezavantaje
  - Testarea Black-box vs. Testare White-box
- **Testare bazată pe experienţă**
  - Definiţie
  - Error guessing. Exploratory testing
- **Întrebări pentru examen**
- **Bibliografie**

# ABORDĂRI ALE TESTARE

Abordări ale testării. Clasificare

Tehnici de testare asociate

# Abordări ale testării. Clasificare

- abordare de testare
  - modalitate de aplicare a unei tehnici de testare;
- clasificare
  - testare Black-box (**criteriul cutiei negre**, *engl.* **Black-box testing**);
  - testare White-box (**criteriul cutiei transparente**, *engl.* **White-box testing**);
  - testare Grey-box (**criteriul cutiei gri**, *engl.* **Grey-box testing**);
  - testare bazată pe experienţă (**criteriul statistic**, *engl.* **Experienced-based testing**).

# Tehnici de testare asociate

- criteriul cutiei negre (testare Black-box) – testare funcţională:
  - Partiţionarea în clase de echivalenţă;
  - Analiza valorilor limită;
  - Tabele de decizie, Grafe de tranziţie a stărilor, Cazuri de utilizare, Scenarii de utilizare, etc.;
- criteriul cutiei transparente (testare White-box) – testare structurală:
  - Acoperirea fluxului de control (e.g., instrucţiuni, ramificaţii, decizii, condiţii, bucle, drumuri);
  - Acoperirea fluxului de date;
- criteriul cutiei gri (testare Grey-box) – testare mixtă:
  - folosirea simultană a avantajelor abordărilor black-box şi white-box pentru proiectarea cazurilor de testare;
- criteriul statistic:
  - generarea aleatoare de date de test pe baza unor modele;
  - experienţa testerului.

# TESTARE WHITE-BOX

Definiţie. Caracteristici

Tehnici de testare white-box

# Testare White-Box. Definiţie. Caracteristici

- **criteriul cutiei transparente (*engl.* white-box testing, logic driven testing)**:
  - testare structurală;
  - datele de intrare se aleg pe baza instrucţiunilor care trebuie executate, programul este văzut ca o cutie transparentă;
  - avem acces la structura internă a programului (codul sursă);
  - permite identificarea situaţiilor în care execuţia programului nu acoperă diferite structuri ale acestuia.

# Tehnici de testare white-box

- tehnici de proiectare a cazurilor de testare white-box bazate pe:
  1. **fluxul de control:**
     - **acoperirea drumurilor** [NT2005];
     - **acoperirea codului sursă:**
       - **instrucţiunilor, ramificaţiilor, deciziilor, condiţiilor, deciziilor şi condiţiilor, condiţiilor multiple** [Myers2004], condiţiilor/deciziilor modificate;
       - **buclelor** [Beizer1990];
     - acoperirea predicatelor *(engl*. predicate complete coverage);
     - acoperirea prin mutaţii;
  2. fluxul de date.

# TESTARE BAZATĂ PE FLUXUL DE CONTROL. COMPONENTE

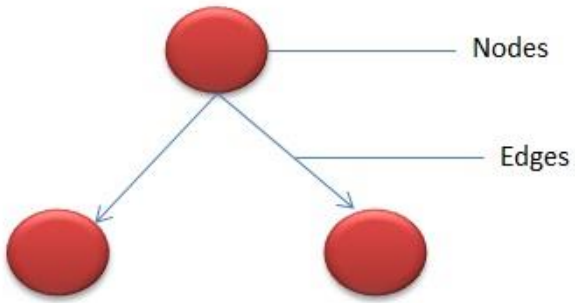
Definiţie. Caracteristici
Graful fluxului de control. Exemple
Drumuri în CFG. Exemple
Complexitatea ciclomatică. Exemple
Testare bazată pe fluxul de control – Avantaje şi dezavantaje

# Testarea bazată pe fluxul de control

- **testarea bazată pe fluxul de control**
  - utilizează **structurile de control** pentru proiectarea cazurilor de testare;
  - **scop:** acoperirea prin cazuri de testare la un nivel statisfăcator a structurilor de control din programul testat;
- **componente:**
  - graful fluxului de control;
  - complexitatea ciclomatică.

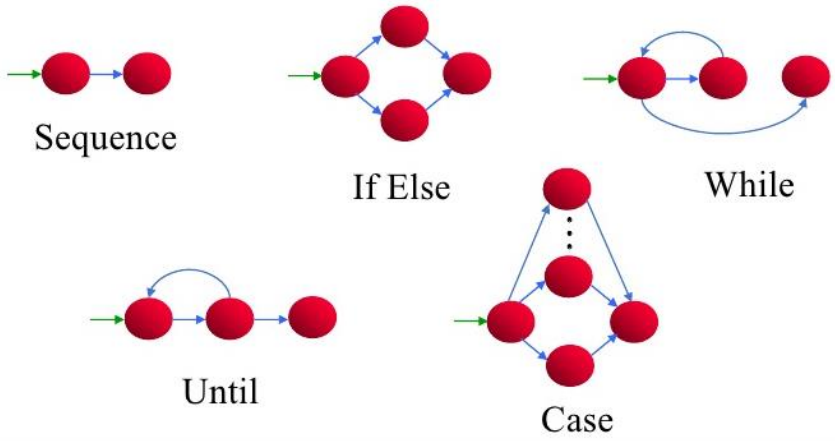
# Graful fluxului de control. Definiţie

- **graful fluxului de control** (*engl.* **Control Flow Graph, CFG**):
  - **reprezentare grafică detaliată a unei unităţi de program;**
  - permite vizualizarea tuturor drumurilor din unitatea de program;
- **graf orientat:**
  - **vârf (*engl.* node):**
    - indică structuri secvenţiale şi condiţiile din structurile alternative sau repetitive;
  - **arc (*engl.* edge):**
    - indică sensul transmiterii controlului logic în cadrul programului;



# CFG. Caracteristici

- permite reprezentarea grafică a structurilor de programare;
- tipuri de vârfuri:
  - **decizie:**
    - are o condiţie prin care se permite ramificarea execuţiei prin cel puţin două căi;
    - e.g., instrucţiunile `if, while, repeat/until, case`;
  - **instrucţiune/calcul:**
    - conţine o secvenţă de instrucţiuni;
  - **conector:**
    - nu conţine o instrucţiune şi reprezintă un punct al programului care uneşte mai multe ramificaţii;
  - **intrare, ieşire:**
    - există un singur vârf intrare şi un singur vârf ieşire;
    - în vârful de intrare nu întră nici un arc;
    - din vârful de ieşire nu iese nici un arc.



# CFG. Construire

- paşi de elaborare a unui CFG:
  1. se numerotează unic fiecare element de structură secvenţială (calcul) şi condiţională (decizie);
  2. se începe pornind de la vârful de intrare, care are (de obicei) numărul 1;
  3. se adaugă celelalte vărfuri corespunzătoare structurilor numerotate şi se unesc prin arce, evidenţiind transmiterea controlului în cadrul programului;
  4. la final, toate ieşirile posibile din program se unesc în vârful de ieşire;

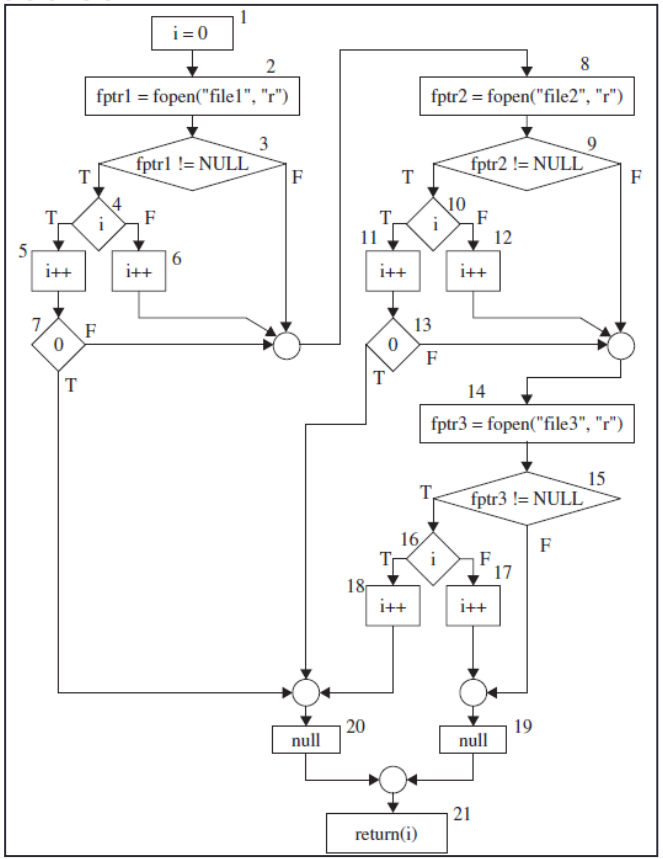
- condiţii complexe care conţin atribuiri ==> CFG are o descriere complexă [NT2005];
  - e.g., `if (((fptr1 = fopen(''file1'', ''r'' )) != NULL) && (i++) && (0)){...}.`

# CFG. Complexitatea construirii

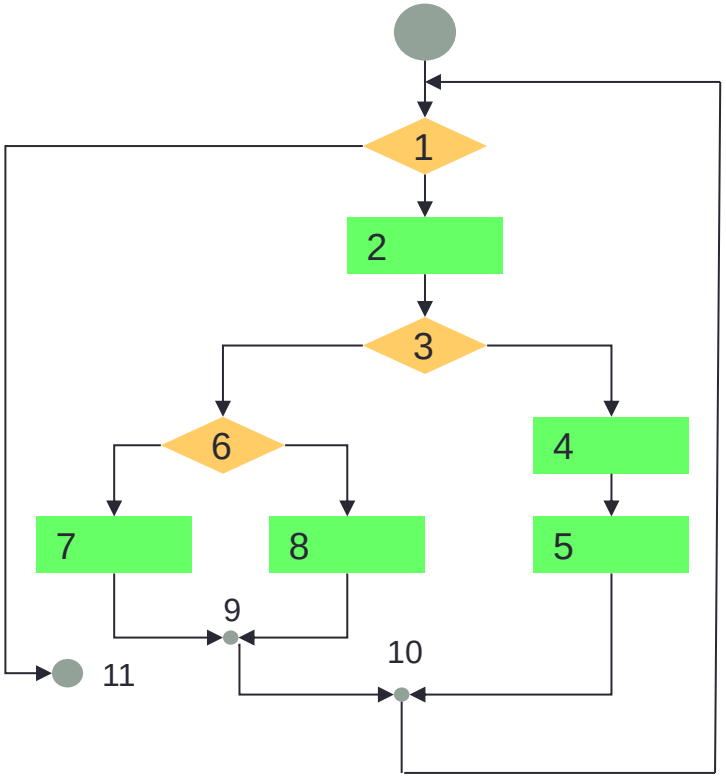
- pentru metoda `openfiles()`, avem CFG alăturat;

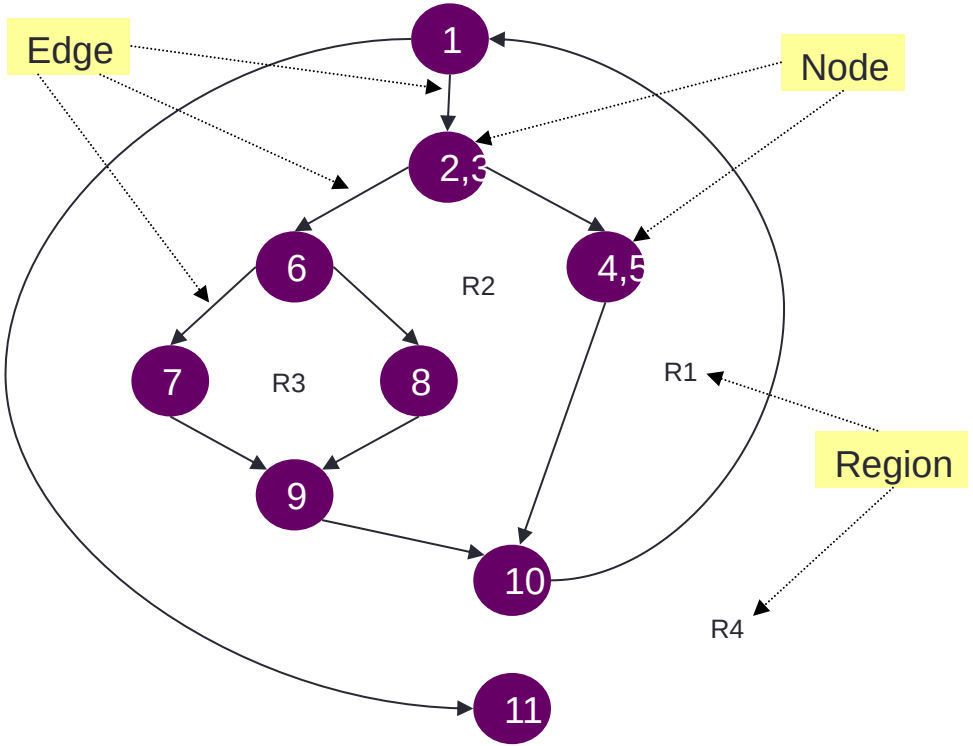
```
FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */

int openfiles(){
   /*
      This function tries to open files "file1", "file2", and
      "file3" for read access, and returns the number of files
      successfully opened. The file pointers of the opened files
      are put in the global variables.
   */
     int i = 0;
     if(
         ((( fptr1 = fopen("file1", "r")) != NULL) && (i++)
                                                   && (0)) ||
         ((( fptr2 = fopen("file2", "r")) != NULL) && (i++)
                                                   && (0)) ||
         ((( fptr3 = fopen("file3", "r")) != NULL) && (i++))
     );
     return(i);
}
```



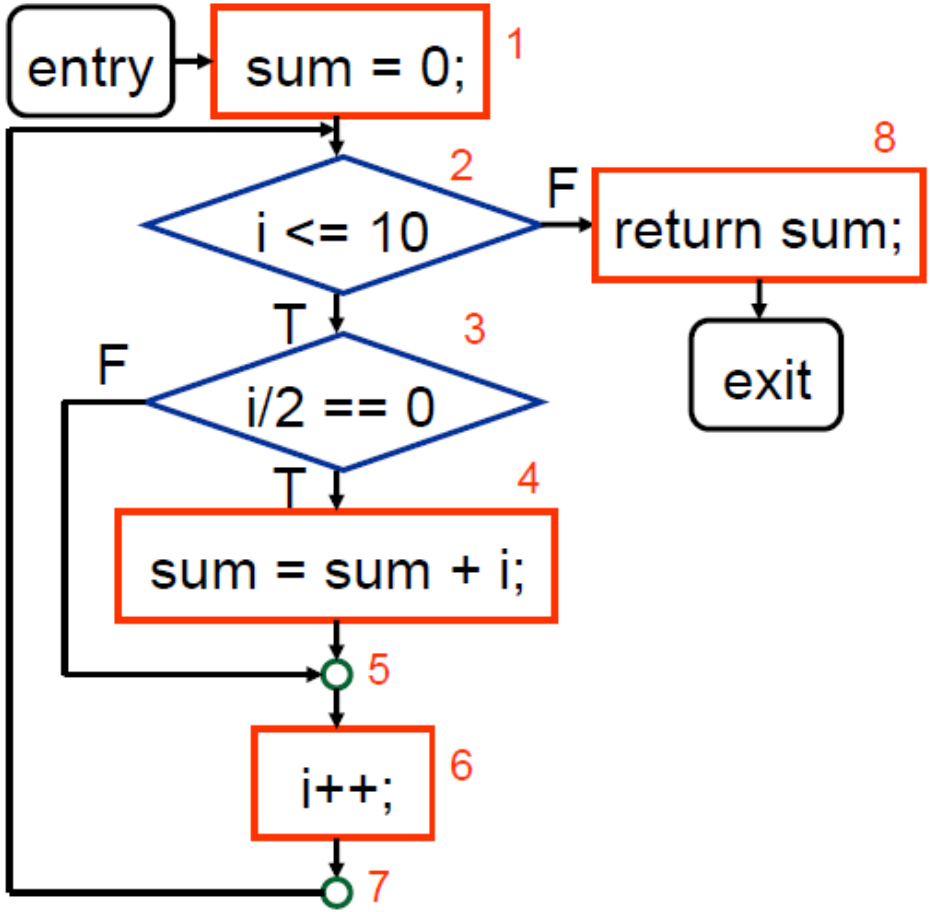
# CFG. Exemple de notaţii



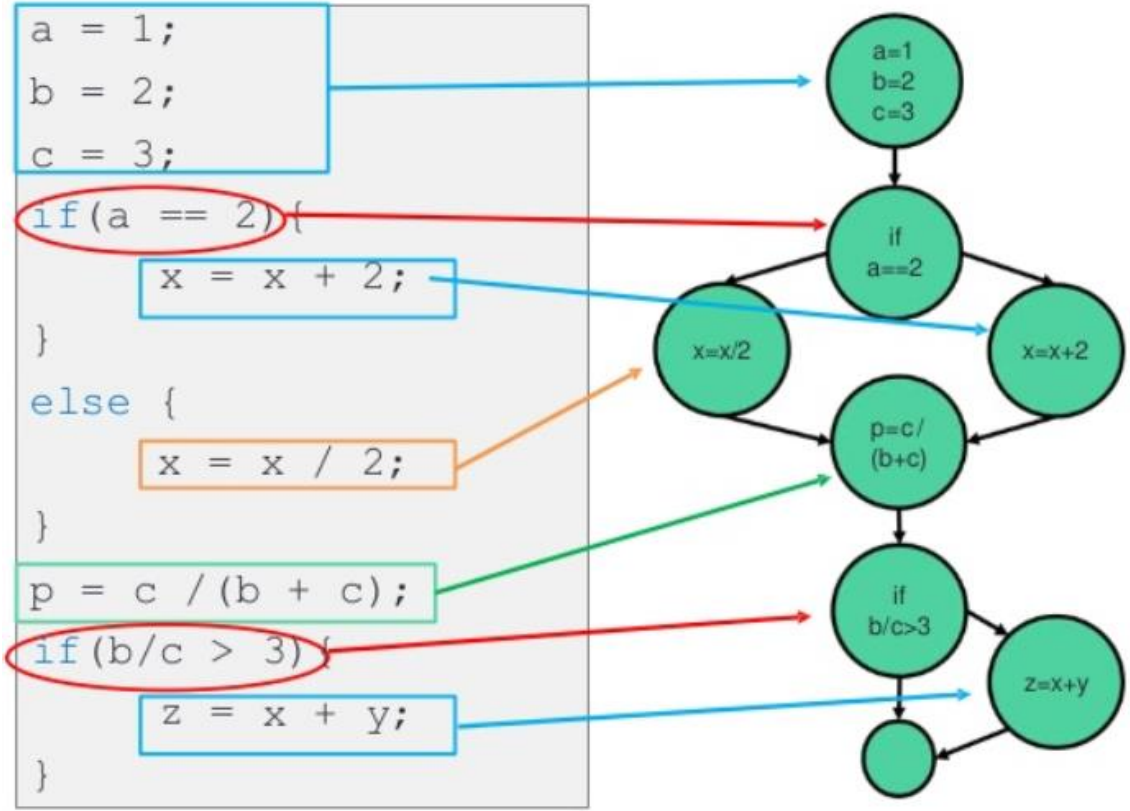



# CFG. Exemple (1)

| | |
|---|---|
| * | `int evenSum(int i) {` |
| 1 | `    int sum = 0;` |
| 2 | `    while (i <= 10) {` |
| 3 | `        if (i/2 == 0) {` |
| 4 | `            sum = sum + i;` |
| 5 | `        }` |
| 6 | `        i++;` |
| 7 | `    }` |
| 8 | `    return sum;` |
| * | `}` |



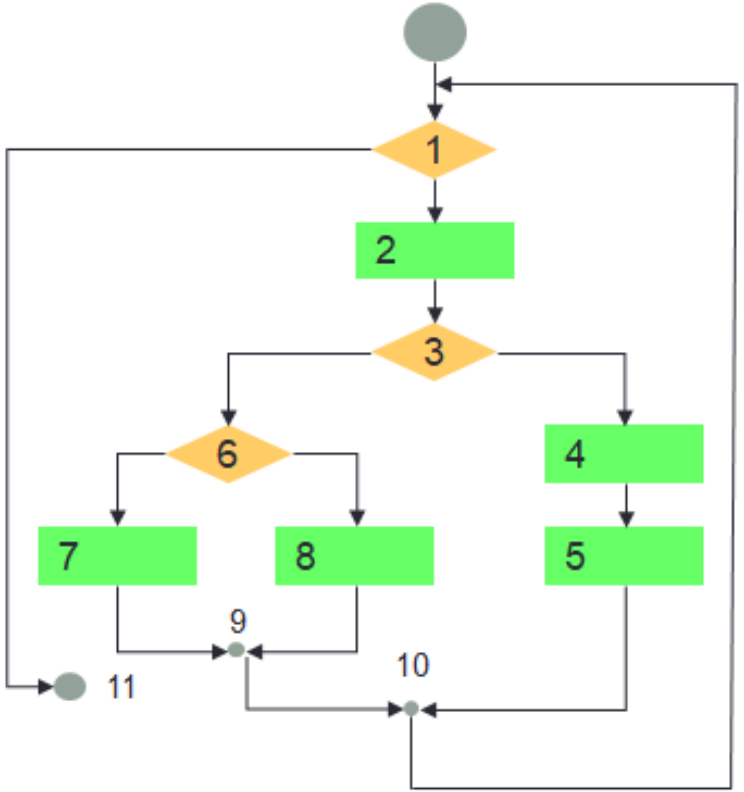
# CFG. Exemple (2)

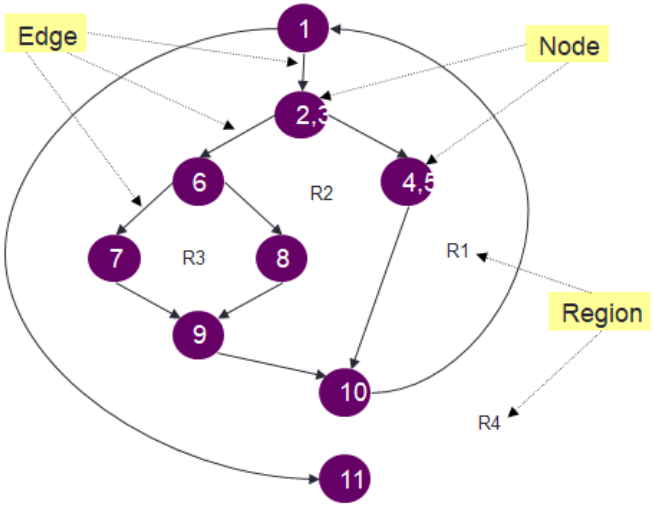


# CFG. Drumuri în CFG

- **drum**:
  - execuţia unei secvenţe de instrucţiuni **de la punctul de intrare până la punctul de ieşire** al CFG asociat unei unităţi de program;
- **drum independent** (*engl.* **independent path**):
  - orice drum în CFG care introduce **cel puţin o instrucţiune nouă sau o condiţie nouă**, care este executată cel puţin o dată;
- mulţimea drumurilor independente formează **mulţimea drumurilor de bază** (*engl.* **basis path set**) a unui CFG;
  - indică **numărul minim de cazuri de testare** care trebuie executate pentru ca toate instrucţiunile să fie executate cel puţin o dată;

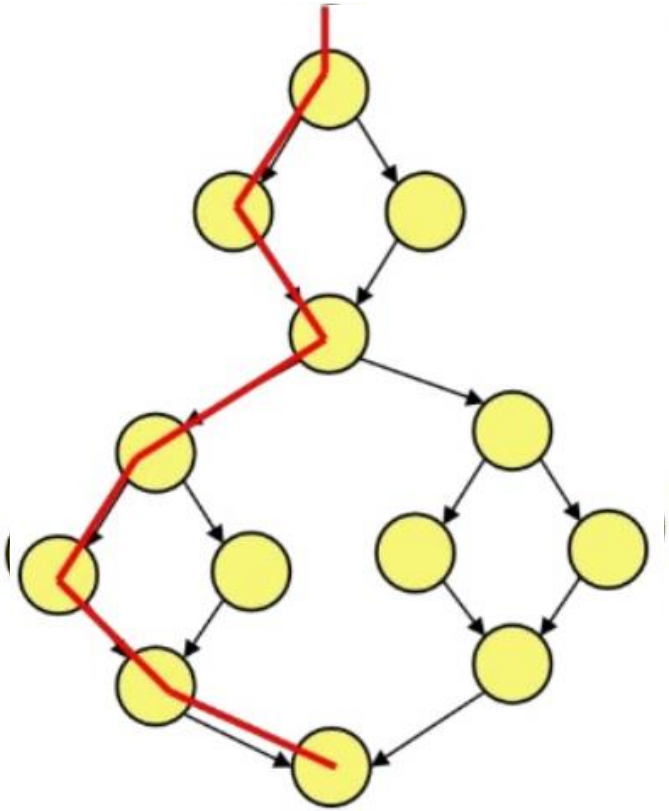
# Drumuri independente în CFG. Exemple (1)

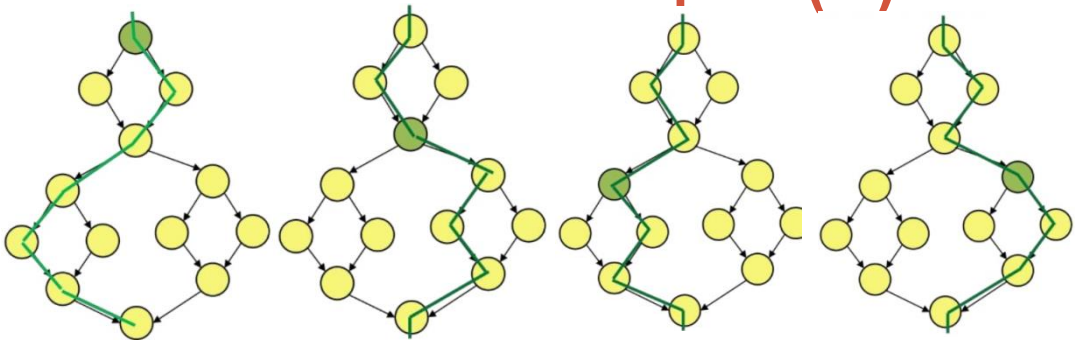


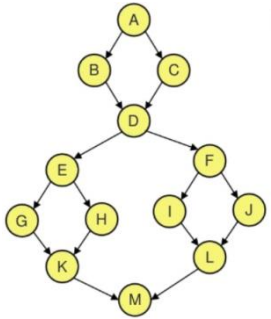




- drumuri independente:
  - **drum 1:** 1(F)-11.
  - **drum 2:** 1(T)-2-3(T)-4-5-10-1(F)-11.
  - **drum 3:** 1(T)-2-3(F)-6(T)-8-9-10-1(F)-11.
  - **drum 4:** 1(T)-2-3(F)-6(F)-7-9-10-1(F)-11.

# Drumuri independente în CFG. Exemple (2)



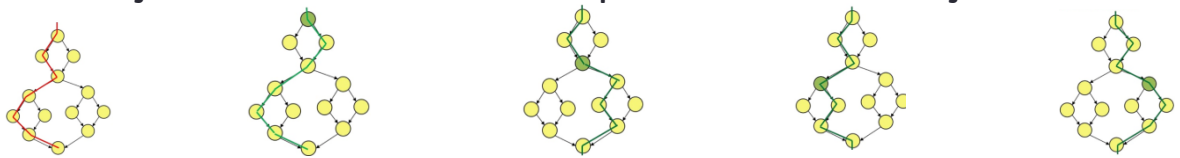





- drumuri independente:
  - **drum 1: A-B-D-E-G-K-M;**
  - **drum 2: A-C-D-E-G-K-M;**
  - **drum 3: A-B-D-F-I-L-M;.**
  - **drum 4: A-B-D-E-H-K-M;**
  - **drum 5: A-C-D-F-J-L-M.**

# CFG. Algoritm de construire a drumurilor independente

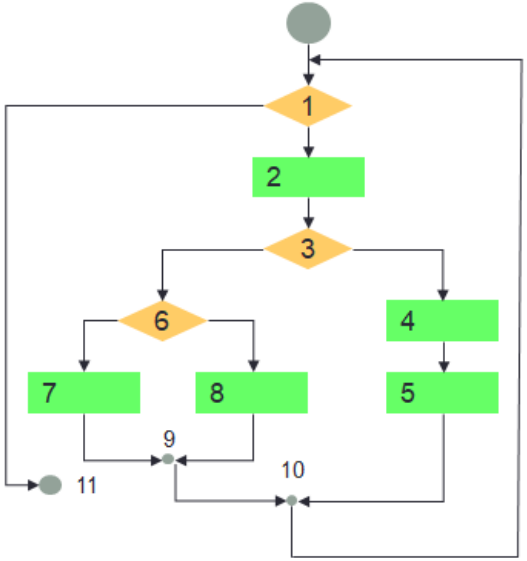
- **Algoritmul lui McCabe [McCabe1984, McCabe Baseline Method]**
  1. se alege un drum normal (numit **drum iniţial, D1**); se recomandă ca acesta să aibă cât mai multe decizii este posibil;
  2. pentru generarea următorului drum (**D2**), se modifică rezultatul evaluării primei decizii de pe D1 și păstrând numărul de decizii ale drumului D1;
  3. pentru generarea următorului drum (**D3**), se modifică rezultatul evaluării celei de a doua decizii de pe D1;
  4. se repetă pasul 3 până când toate deciziile de pe D1 au fost modificate/inversate;
  5. se reiau paşii de la 1..4 considerând ca drum iniţial pe D2, modificând/inversând deciziile, până când toate se obţin toate drumurile independente din mulţimea de bază a CFG.



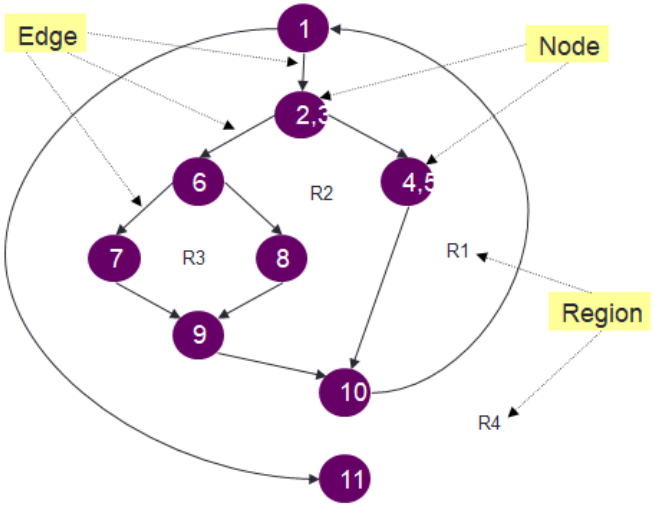
# Complexitatea ciclomatică. Definiţie

- **complexitatea ciclomatică** (*engl.* **McCabe's cyclomatic complexity, CC**):
  - *metrică software* aplicată pentru măsurarea cantitativă a complexităţii logice a unui program;
  - *permite determinarea numărului de drumuri independente din mulţimea de bază a unui CFG;*
- modalităţi de calcul a CC la nivelul CFG:
  - ***CC* = numărul de regiuni din CFG;**
  - ***CC* = *E* – *N* + 2, unde *E* - *#arce*, N - *#vârfuri* ;**
  - ***CC* = *P* + 1, unde *P* - *#vârfuri condiţie*.**
- **regiune:**
  - **zonă a CFG marginită parţial sau în totalitate de arce şi vârfuri;**

# CC. Exemple (1)

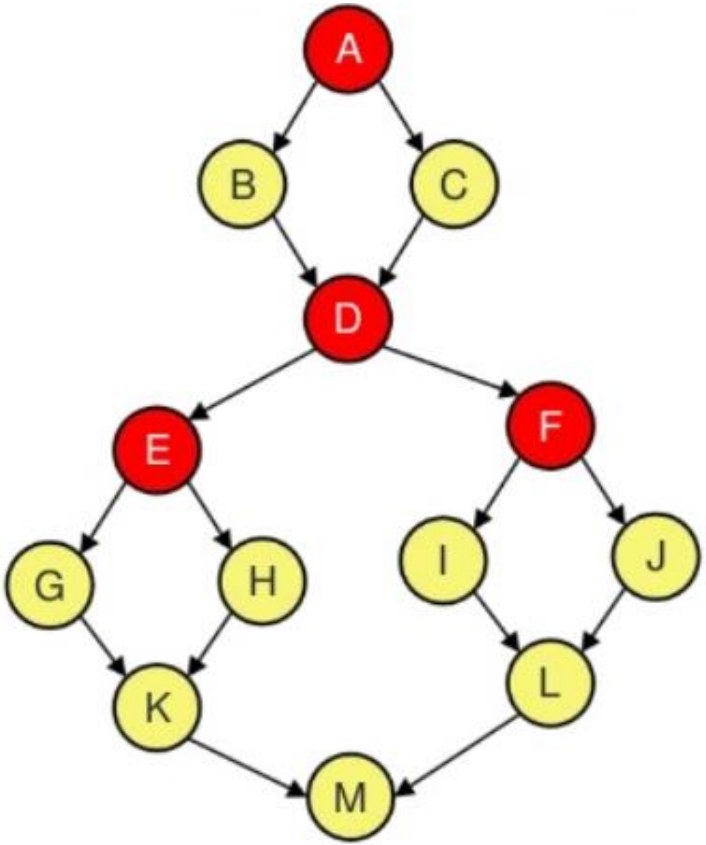



- CC pentru CFG:
  - CC = numărul de regiuni = 4 regiuni = 4.
  - CC = E − N + 2 = 14 arce - 12 vârfuri + 2 = 4.
  - CC = P + 1 = 3 vârfuri condiţie + 1 = 4.




- drumuri independente:
  - **drum 1:** 1(F)-11.
  - **drum 2:** 1(T)-2-3(T)-4-5-10-1(F)-11.
  - **drum 3:** 1(T)-2-3(F)-6(T)-8-9-10-1(F)-11.
  - **drum 4:** 1(T)-2-3(F)-6(F)-7-9-10-1(F)-11.

# CC. Exemple (2)




- drumuri independente:
  - drum 1: A-B-D-E-G-K-M;
  - drum 2: A-C-D-E-G-K-M;
  - drum 3: A-B-D-F-I-L-M;.
  - drum 4: A-B-D-E-H-K-M;
  - drum 5: A-C-D-F-J-L-M.
- CC pentru CFG:
  - CC = numărul de regiuni = 5 regiuni = 5;
  - CC = E − N + 2 = 16 arce - 13 vârfuri + 2 = 5;
  - CC = P + 1 = 4 vârfuri condiţie + 1 = 5.

# Testarea bazată pe CFG. Avantaje şi dezavantaje

## Avantaje

- Testarea de bază aplicată în **testarea unitară**, care sunt dezvoltate la momentul curent;
- Se aplică pentru modulele pentru care **prin inspectare nu pot fi suficient verificate**;
- Limbajele orientate-obiect reduc numărul de bug-uri la nivelul fluxului de control;

## Dezavantaje

- Dacă este aplicată de tester şi nu de programator, este necesar ca testerul să aibă abilităţi de programare pentru a înţelege codul sursă şi modul de execuţie al acestuia;
- **Tehnică de testare consumatoare de timp**, deoarece mai întâi se elaborează CFG, CC, drumuri independente şi ulterior se proiectează cazurile de testare;
- Consideră că:
  - Specificaţiile sunt corecte;
  - Datele sunt definite şi accesate corespunzător;
  - Nu există alte bug-uri pe lângă cele determinate de fluxul de control.

# TESTARE BAZATĂ PE ACOPERIREA DRUMURILOR

Definiţie

Algoritm

Exemplu

# Testare bazată pe acoperirea drumurilor. Definiţie

- **acoperirea tuturor drumurilor** (*engl.* **all path coverage, apc**):
  - testarea tuturor drumurilor programului;
- avantaje şi dezavantaje:
  - permite identificarea tuturor defectelor, dar **nu şi de pe drumurile care lipsesc**;
  - **dificil** de realizat în practică pentru **programe cu structuri repetitive** ===> se alege un număr redus de drumuri;

# Testare bazată pe acoperirea drumurilor. Algoritm

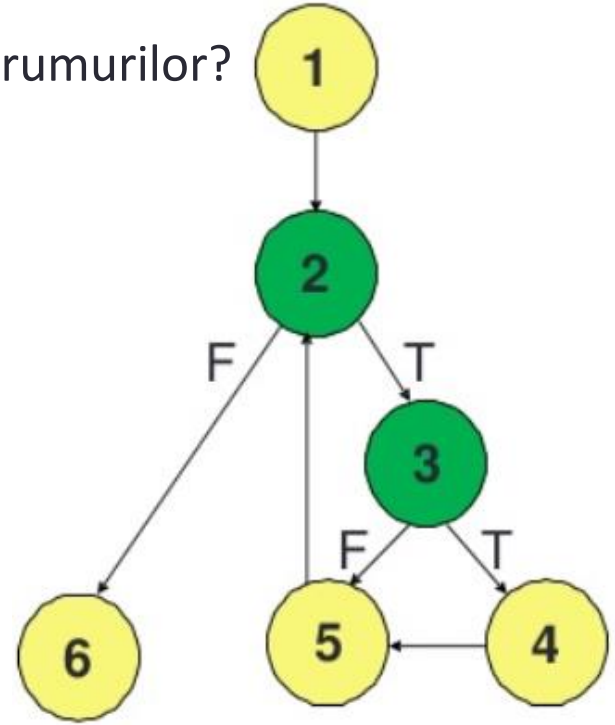
- Algoritmul de proiectare a cazurilor de testare bazat pe drumuri este:
  1. Se elaborează CFG;
  2. Se calculează CC pe baza CFG;
  3. Se determină mulţimea de bază a CFG (cu drumuri independente, liniare);
  4. Se proiectează câte un caz de testare pentru fiecare drum independent identificat.

- ordinea  de selectare a drumurilor:
  - drumuri scurte;
  - drumuri de lungime crescândă;
  - drumuri lungi, complexe, alese arbitrar.

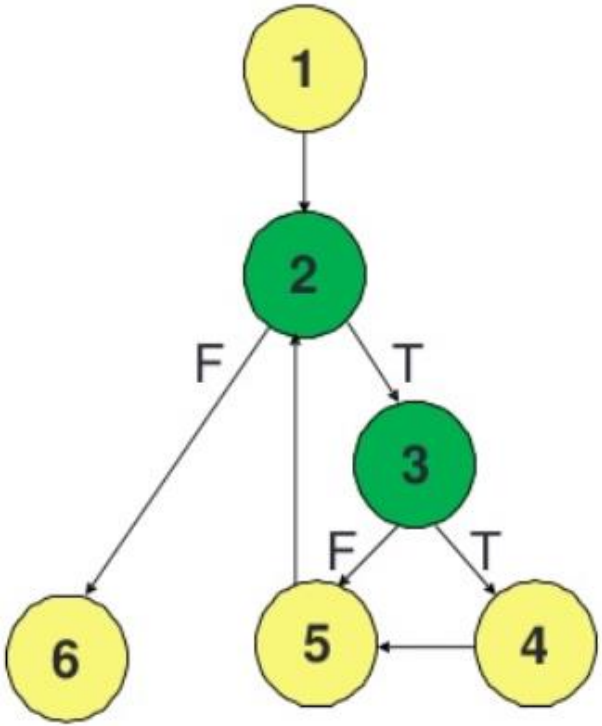
# Testare bazată pe acoperirea drumurilor. Exemplu

Câte cazuri de testare sunt necesare pentru acoperira drumurilor?

```
const int SIZE = 10;
int i;
int array[SIZE] = {52,88,90,21,62,10,16,39,45,80};
int min = array[0];
while(i< 10) {
      if(array[i] < min) {
            min = array[i];
      }
      i = i + 1;
}
cout << min;
```



# Testare bazată pe acoperirea drumurilor. Exemplu (cont)




- CC pentru CFG:
  - CC = numărul de regiuni = 3 regiuni = 3;
  - CC = E − N + 2 = 7 arce - 6 vârfuri + 2 = 3;
  - CC = P + 1 = 2 vârfuri condiţie + 1 = 3.
- drumuri independente:
  - **drum 1:** 1-2(F)-6.
  - **drum 2:** 1-2(T)-3(F)-5-2(F)-6.
  - **drum 3:** 1-2(T)-3(T)-4-5-2(F)-6.

# TESTARE BAZATĂ PE ACOPERIREA CODULUI SURSĂ

Definiţie. Criterii de acoperire

Acoperirea instrucţiunilor. Definiţie. Exemplu

Acoperirea deciziilor. Definiţie. Exemplu

Acoperirea condiţiilor. Definiţie. Exemplu

Acoperirea deciziilor şi condiţiilor. Definiţie. Exemplu

Acoperirea condiţiilor multiple. Definiţie. Exemplu

Acoperirea buclelor. Definiţie. Exemplu

# Testare bazată pe acoperirea codului sursă. Definiţie

- **acoperirea codului sursă**:
  - testarea tuturor structurilor de control folosind un număr minim de teste, astfel încât să fie satisfăcute criteriile:
    - **acoperirea instrucţiunilor** (*engl.* **statement/line/node coverage**);
    - acoperirea ramificaţiilor:
      - **acoperirea deciziilor** (arcelor) (*engl.* **decision/branch/edge coverage**);
      - **acoperirea condiţiilor** (*engl.* **condition coverage**);
      - **acoperirea deciziilor şi condiţiilor** (*engl.* **decision-condition coverage**);
      - **acoperirea condiţiilor multiple** (*engl.* **multiple condition coverage**);
    - acoperirea structurilor repetitive:
      - **acoperirea buclelor** (*engl.* **loop coverage**).

# Acoperirea instrucţiunilor. Definiţie

- **acoperirea instrucţiunilor** (*engl.* **statement/line/node coverage, sc**):
  - proiectarea cazurilor de testare astfel încât toate instrucţiunile sunt executate cel puţin o dată, adică fiecare vârf al CFG este vizitat;
  - **cel mai slab criteriu de acoperire în testare;**
    - o mulţime de teste care nu realizează acoperire 100% a vârfurilor nu este considerată acceptabilă.

# SC. Exemplu

**Fiecare intrucţiune trebuie să fie executată cel puţin o dată.**

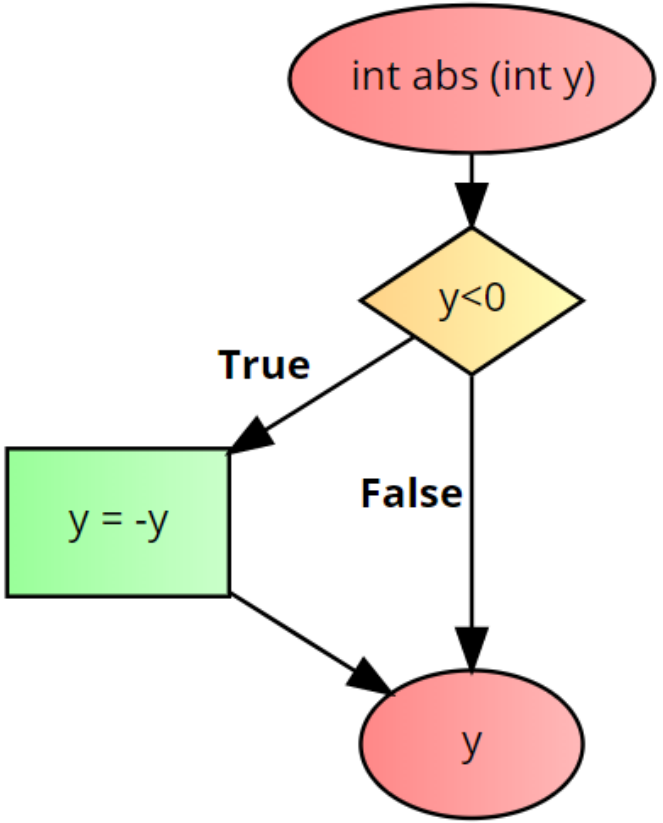
// returnează valoarea absolută a lui y

```
int abs (int y) {
   if (y<0)
      y = -y;
   return y;
}
```

**Care este numărul minim de cazuri de testare necesar?**

| TC | Input | Expected result | Actual result |
|---|---|---|---|
| 1 | -2 | 2 | 2 |

**Criteriul de acoperire a instrucţiunilor este îndeplinit 100%.**

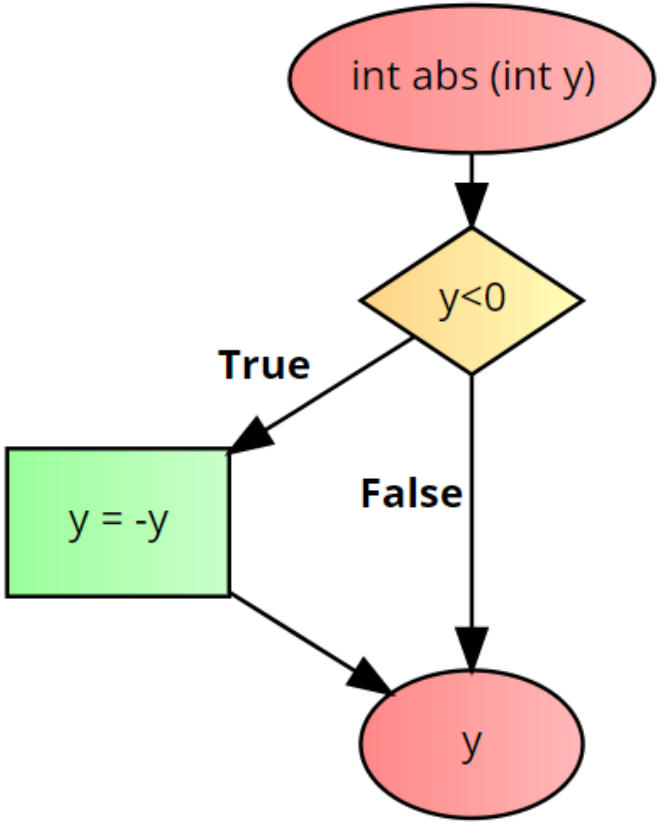


# SC. Observaţii

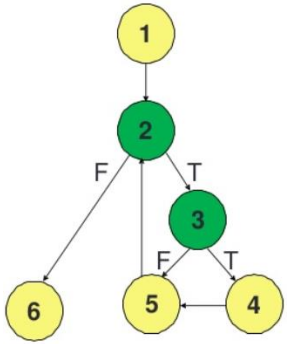
- cel mai slab criteriu de acoperire deoarece:
  - nu acoperă ramificaţia `else` pentru instrucţiunile `if` care nu descriu explicit această ramificaţie; nu evdenţiază **implicit** prezenţa posibilelor bug-uri de pe aceste ramificaţii;
- SC se recomandă doar atunci când nu există alte criterii de acoperire care se pot aplica.

```
// returnează valoarea absolută a lui y
int abs (int y) {
   if (y<0)
      y = -y;
   return y;
}
```

**Este necesară acoperirea deciziilor.**



# Acoperirea deciziilor. Definiţie




- **ramificaţie** (*engl.* **branch/edge**):
  - arc care porneşte dintr-un vârf;
  - din fiecare vârf porneşte cel mult un arc, mai puţin din vârful de ieşire al CFG;
  - din vârfurile de decizie pornesc două arce, etichetate cu true şi false;
- **acoperirea deciziilor** (engl. **branch/edge/decision coverage**, **dc**):
  - acoperirea unui arc **a** = drum care parcurge arcul **a**;
  - proiectarea cazurilor de testare se face astfel încât *fiecare arc de decizie* să fie parcurs cel puţin o dată;

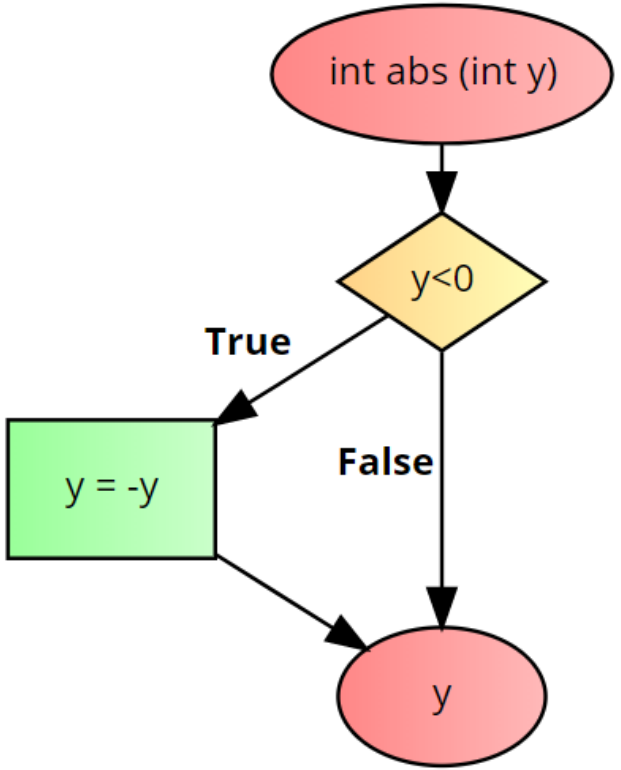
- regulă de selectare:
  - fiecare decizie selectată, evaluată la `true` sau `false`, trebuie să se găsească pe cel puţin un drum.

# DC. Exemplu

**Pentru fiecare decizie, fiecare ramificaţie (`true`, `false`) trebuie să fie executată cel puţin o dată.**

// returnează valoarea absolută a lui y

```
int abs (int y) {
   if (y<0)
      y = -y;
   return y;
}
```



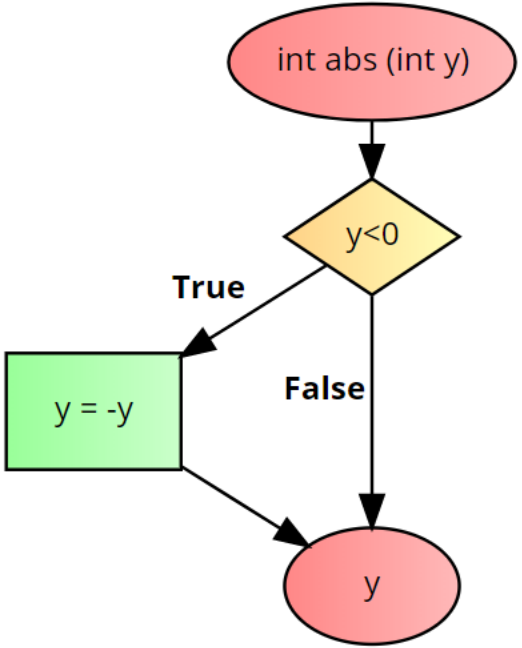

**Care este numărul minim de cazuri de testare necesar ?**

| TC | Input | Decision (y<0) | Expected result | Actual result |
|---|---|---|---|---|
| 1 | -2 | true | 2 | 2 |
| 2 | 3 | false | 3 | 3 |

**Criteriul de acoperire a deciziilor este îndeplinit 100%. Ambele ramificaţii ale deciziei au fost explorate.**

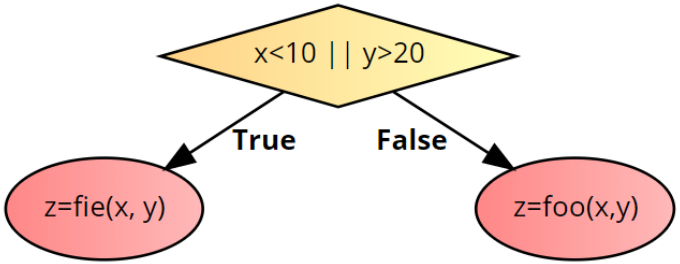
# DC vs. SC

- **dc ===> sc**;
  - instrucţiunile se află pe arce; dacă se parcurge fiecare arc atunci se execută şi instrucţiunile asociate;



# DC. Observaţii

- e.g., `y = fou(x);`
  `if (x<10 || y>20)`
  `{z=fie(x, y);}`
  `else {z=foo(x,y);}`
- pentru x=1 şi y=2 **nu** mai este relevantă evaluarea condiţiei `y>20`;

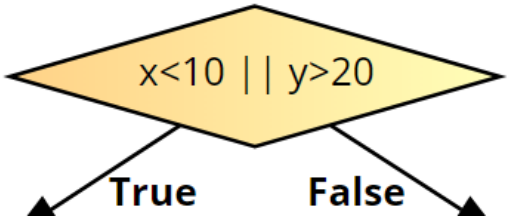



- în deciziile formate din mai multe condiţii, unele condiţii pot să rămână neacoperite fiind **irelevante** pentru rezultatul final al deciziei;

- dacă condiţia este scrisă greşit, e.g., y<20 în loc de y>20, cazuri de testare ca x=1, y=2 nu evidenţiază defectul.

**Este necesară acoperirea condiţiilor.**

# Acoperirea condiţiilor. Definiţie




- **condiţie:**
  - expresie logică dintr-un vârf de decizie;
  - o decizie este formată din una mai mai multe condiţii;
- **acoperirea condiţiilor** (*engl.* **condition coverage, cc**):
  - proiectarea cazurilor de testare se realizează astfel încât fiecare condiţie din fiecare decizie ia fiecare dintre valorile posibile, cel puţin o dată;

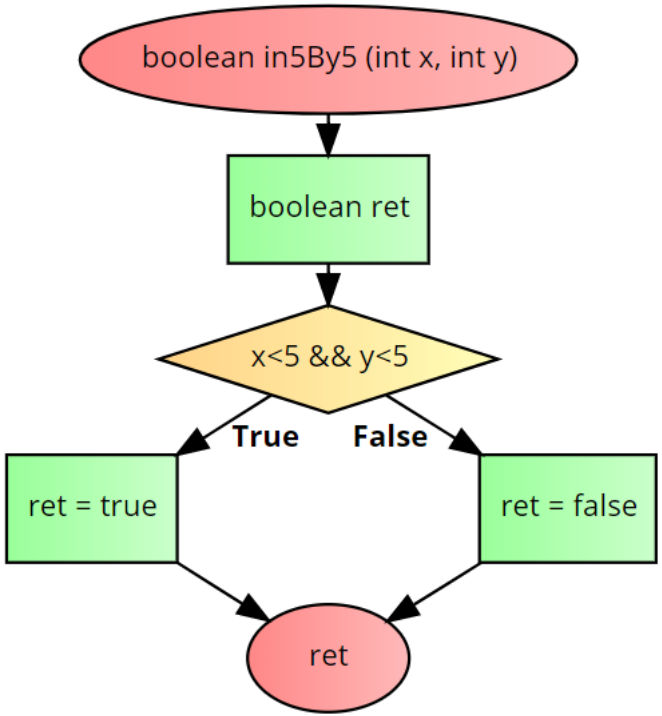
- regulă de selectare:
  - pentru fiecare decizie care conţine mai multe condiţii, fiecare condiţie selectată va fi evaluată la `true` sau `false` şi se va găsi pe cel puţin un drum.

# CC. Exemplu

**Fiecare condiţie din fiecare decizie trebuie să fie executată cel puţin o dată cu fiecare din valorile posibile (e.g., `true`, `false`)**

```
// returnează true dacă (x,y) este în cadranul (5,5).
boolean in5By5 (int x, int y) {
   boolean ret;
   if (x<5 && y<5)
      ret = true;
   else ret = false;
   return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**



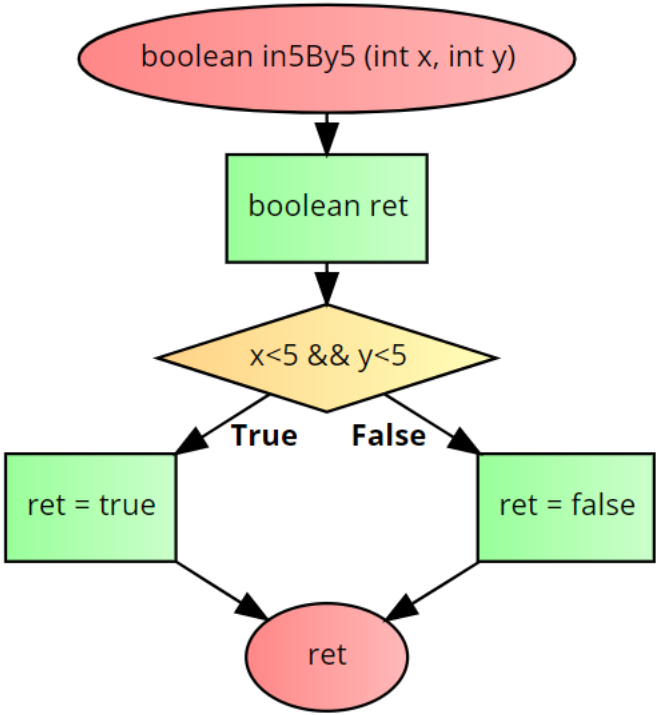
# CC. Exemplu (cont.)

// returnează `true` dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
    boolean ret;
    if (x<5 && y<5)
        ret = true;
    else ret = false;
    return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**

| TC | x | y | Decision (x<5) && (y<5) | Expected result | Actual result |
|---|---|---|---|---|---|
| 1 | 2 | 9 | T && F = **F** | false | false |
| 2 | 9 | 2 | F &&T = **F** | false | false |

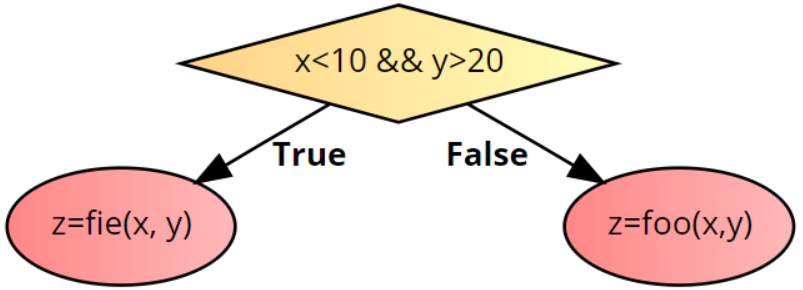



**Criteriul de acoperire a condiţiilor este îndeplinit 100%.**

**Toate rezultatele posibile ale evaluării condiţiilor din decizie au fost explorate.**

# CC. Observaţii

- e.g., `if (x<10 && y>20)`
  `{z=fie(x, y);}`
  `else {z=foo(x,y);}`
- pentru x=11 şi y=21, avem `false && true =` **`false;`**
- pentru x=1 şi y=1, avem `true && false =` **`false;`**

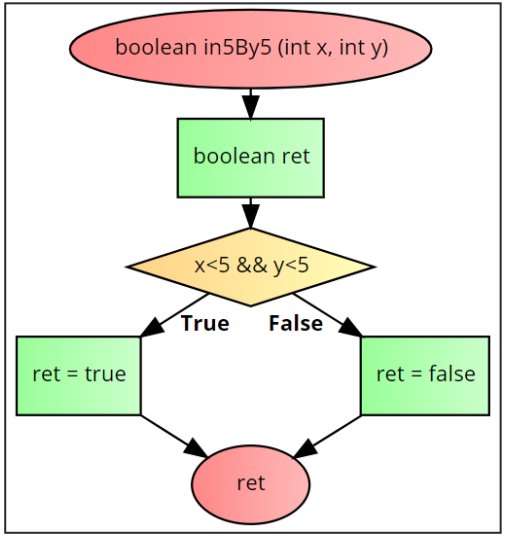


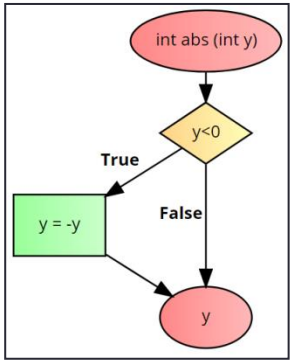

- fiecare *condiţie* selectată este acoperită prin evaluarea la `true` şi `false`, dar *decizia* nu este acoperită, doar ramificaţia `false` este explorată;

**Este necesară acoperirea deciziilor şi condiţiilor.**

# CC vs. DC

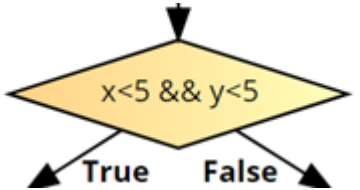
- **în general, cc ===> dc**;
  - prin acoperirea condiţiilor se poate acoperi şi decizia;
- caz particular:
  - **cc = dc atunci când decizia conţine doar o condiţie;**
    - e.g., *decizia* `y<0` este evaluată la `true` sau `false`, similar cu evaluarea *condiţiei* `y<0`, care este evaluată la `true` sau `false` ==> **acoperirea condiţiei este similară cu acoperirea deciziei**;



# Acoperirea deciziilor şi condiţiilor. Definiţie

- **acoperirea deciziilor şi condiţiilor** (*engl.* **decision and condition coverage, dcc**):
  - proiectarea cazurilor de testare astfel încât:
    - fiecare condiţie din fiecare decizie ia toate valorile posibile, cel puţin o dată;
    - fiecare decizie ia toate valorile posibile cel puţin o dată;




- regulă de selectare:
  - pentru fiecare decizie care conţine mai multe condiţii, fiecare condiţie selectată va fi evaluată la `true` sau `false` şi împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puţin un drum.

# DCC. Exemplu

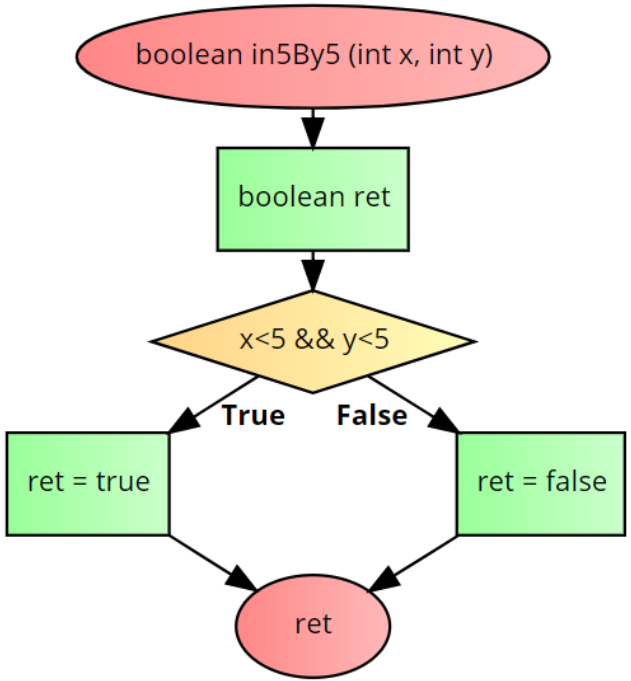
**Fiecare *condiţie* din fiecare decizie trebuie să fie executată cel puţin o dată cu fiecare din valorile posibile, e.g., `true`, `false`.**

**Fiecare *decizie* trebuie să fie executată cel puţin o dată cu fiecare din valorile posibile, e.g., `true`, `false`.**

// returnează `true` dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
   boolean ret;
   if (x<5 && y<5)
      ret = true;
   else ret = false;
   return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**



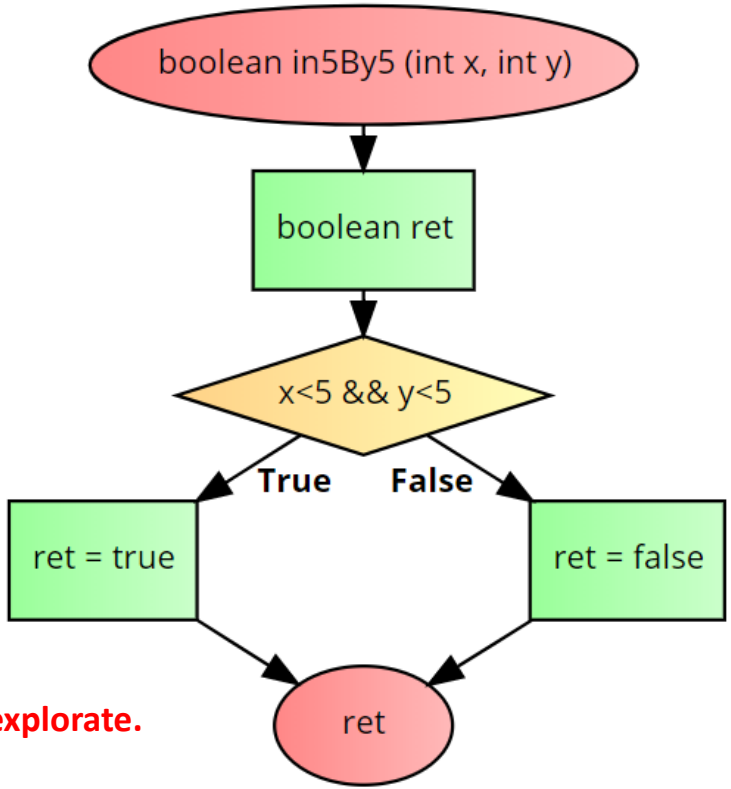
# DCC. Exemplu (cont.)

// returnează `true` dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
    boolean ret;
    if (x<5 && y<5)
        ret = true;
    else ret = false;
    return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**

| TC | x | y | Decision (x<5) && (y<5) | Expected result | Actual result |
|---|---|---|---|---|---|
| 1 | 2 | 3 | T && T = **T** | true | true |
| 2 | 9 | 7 | F && F = **F** | false | false |




**Criteriul de acoperire a deciziilor şi condiţiilor este îndeplinit 100%.**

**Toate rezultatele posibile ale evaluării condiţiilor din decizie au fost explorate.**

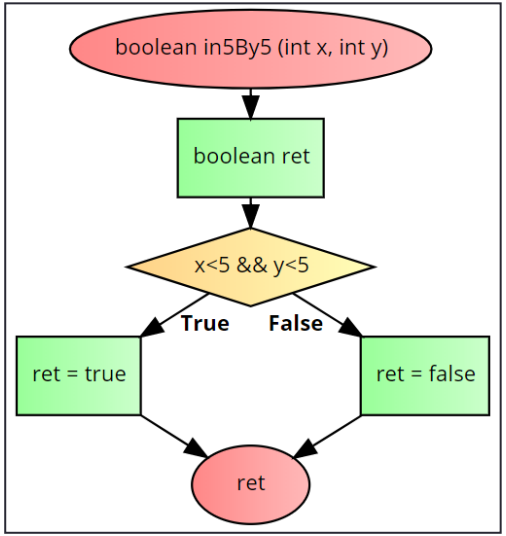
**Ambele ramificaţii ale deciziei au fost explorate.**

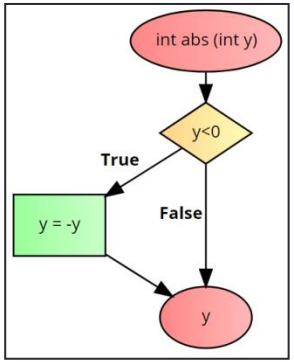
# DCC vs. CC, DCC vs. DC

- **dcc ===> cc**;
  - prin acoperirea deciziilor şi condiţiilor se acoperă condiţiile;
- **dcc ===> dc**;
  - prin acoperirea deciziilor şi condiţiilor se acoperă deciziile;

- caz particular:
  - **dcc = dc şi dcc = cc atunci când decizia conţine doar o condiţie;**
    - e.g., decizia `y<0` este evaluată la `true` sau `false`, similar cu evaluarea condiţiei `y<0`, care este evaluată la `true` sau `false` ==> **acoperirea deciziei şi condiţiei (dcc) este similară cu acoperirea deciziei (dc), care este similară cu acoperirea condiţiei (cc)**;

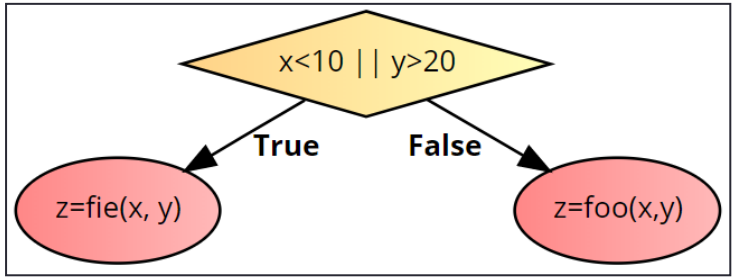
**dcc se aplică doar atunci când decizia este formată din mai multe condiţii.**



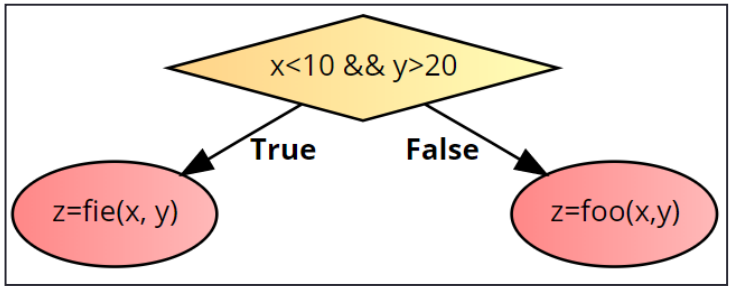



# DCC. Observaţii

- condiţiile logice care folosesc operatorii `&&` şi `||` nu pot fi acoperite prin **dcc**, deoarece compilatorul realizează diverse optmizări (scurt-circuitare la evaluare);

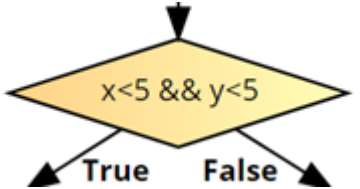



**Este necesară acoperirea condiţiilor multiple.**



# Acoperirea condiţiilor multiple. Definiţie

- **acoperirea condiţiilor multiple** (*engl.* **multiple condition coverage, mcc**):
  - proiectarea cazurilor de testare se realizează asfel încât:
    - toate combinaţiile posibile ale valorilor de ieşire ale unei condiţii, în fiecare decizie, să fie parcurse cel puţin o dată;




- regulă de selectare:
  - fiecare decizie care conţine mai multe condiţii, va combina fiecare condiţie selectată care este evaluată la `true` sau `false` cu celelalte condiţii în toate variantele posibile şi împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puţin un drum.

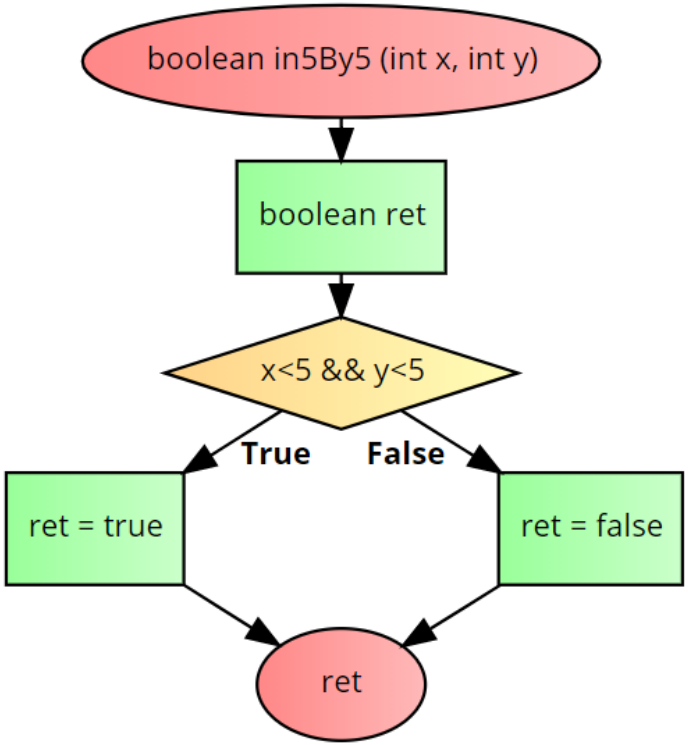
# MCC. Exemplu

**Fiecare *condiţie* din fiecare decizie trebuie să fie executată în toate combinaţiile posibile cu toate celelalte condiţii din cadrul aceleaşi decizii.**

// returnează `true` dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
   boolean ret;
   if (x<5 && y<5)
      ret = true;
   else ret = false;
   return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**



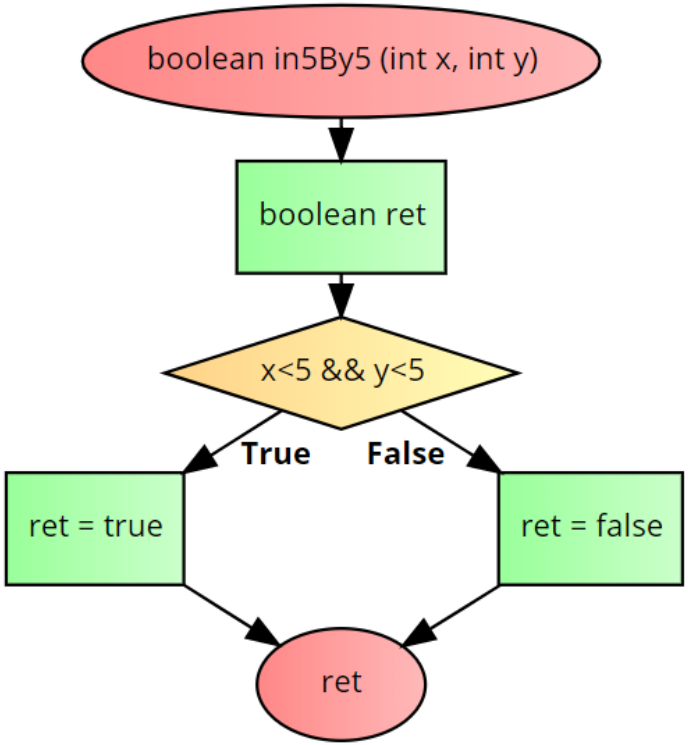
# MCC. Exemplu (cont.)

// returnează `true` dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
    boolean ret;
    if (x<5 && y<5)
        ret = true;
    else ret = false;
    return ret;
}
```

**Care este numărul minim de cazuri de testare necesar ?**

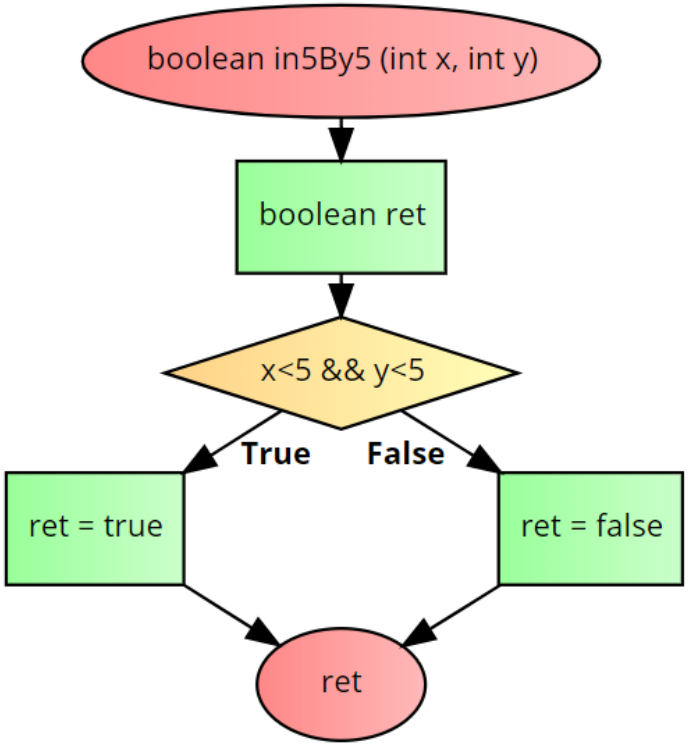
| TC | x | y | Decision (x<5) && (y<5) | Expected result | Actual result |
|---|---|---|---|---|---|
| 1 | 2 | 3 | T && T = **T** | true | true |
| 2 | 9 | 7 | F && F = **F** | false | false |
| 3 | 2 | 7 | T && F = **F** | false | false |
| 4 | 9 | 3 | F && T = **F** | false | false |




**Criteriu de acoperire a condiţiilor multiple îndeplinit 100%. Toate cominaţiile posibile ale condiţiilor au fost explorate.**

# MCC vs. DCC

- **mcc ===> dcc**;
  - prin acoperirea multiplă a condiţiilor se acoperă deciziile şi condiţiile;

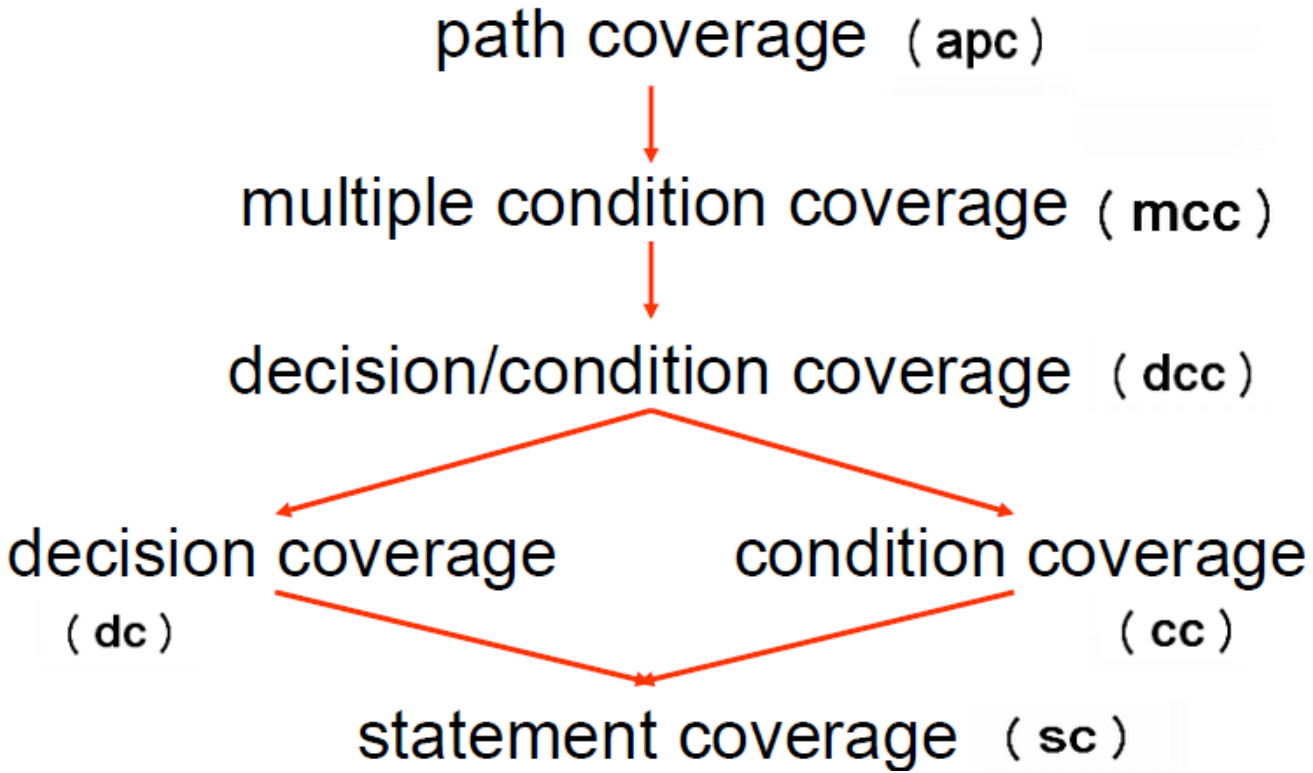


# Testare bazată pe acoperirea codului sursă. Reguli de acoperire minimală

- dacă programul are o singură condiţie în fiecare vârf de decizie, atunci se aplică
  - **acoperirea deciziilor (dc), în acest caz dc = cc;**

- dacă programul are condiţii multiple în vârfuri de decizie, atunci se aplică
  - **acoperirea condiţiilor multiple (mcc).**

# APC vs. SC vs DC. vs. CC vs. DCC vs. MCC

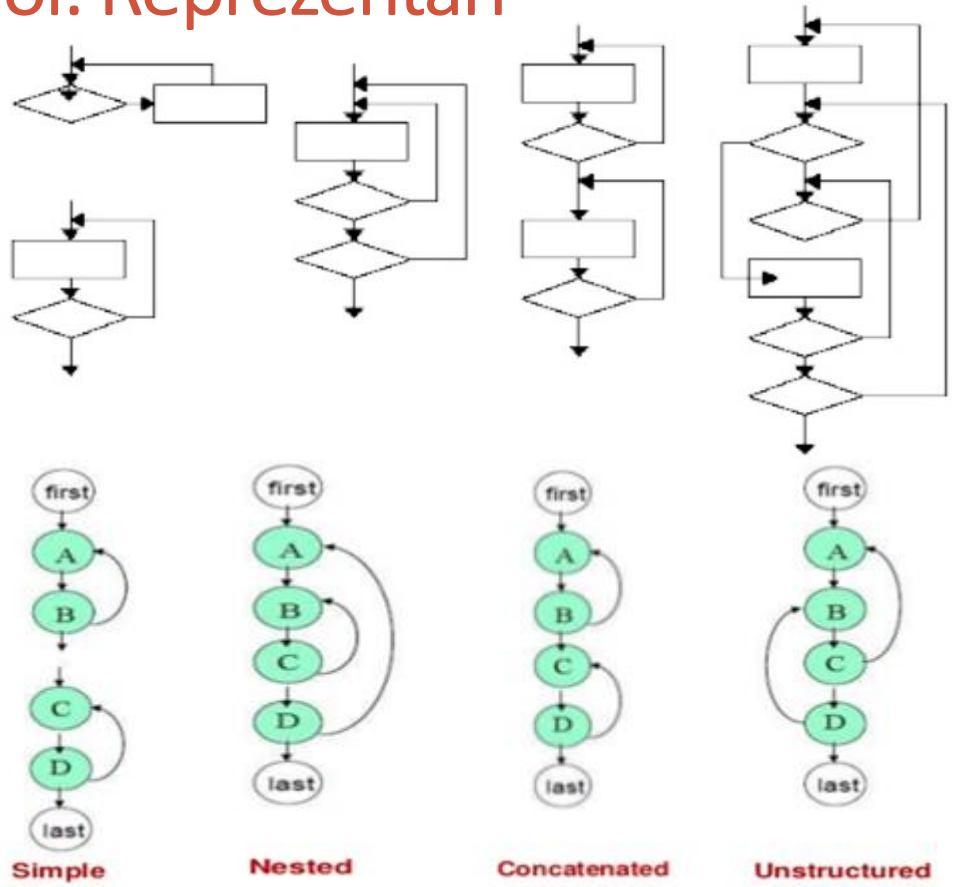


# Acoperirea buclelor. Definiţie

- **acoperirea buclelor** (*engl.* **loop coverage**, **lc**);
  - proiectarea cazurilor de testare astfel încât structurile repetitive să fie iterate de un număr *variabil* de ori;

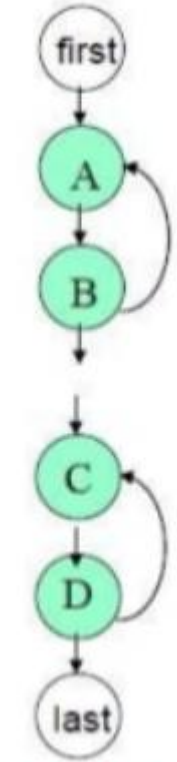
# LC. Clasificarea buclelor. Reprezentări

- tipuri de bucle:
  - simple;
  - imbricate;
  - concatenate;
  - nestructurate.



# LC. Bucle simple

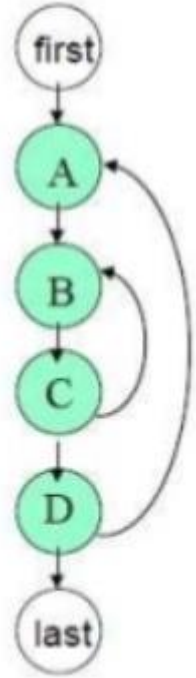
- **bucle simple** (n – numărul maxim de parcurgeri al buclei):
  - omiterea bublei (0 parcurgeri);
  - 1 parcurgere a buclei;
  - 2 parcurgeri ale buclei (evidenţiază defecte de iniţializare);
  - m parcurgeri ale buclei, unde m < n;
  - n-1 parcurgeri ale buclei;
  - n parcurgeri ale buclei;
  - n + 1 parcurgeri ale buclei.



Simple

# LC. Bucle imbricate

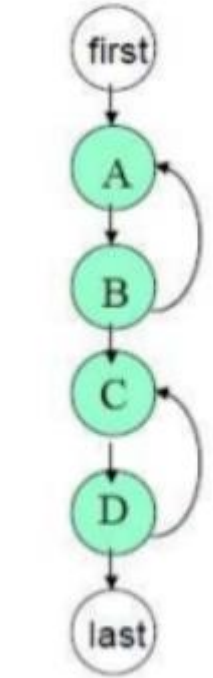
- **bucle imbricate**:
  1. se porneşte de la bucla cea mai interioară; toate celelalte bucle sunt setate pe valori minime;
  2. se testează bucla cea mai interioară ca şi buclă simplă, păstrând buclele exterioare la valoarea minimă a parametrului de iteraţie;
  3. se progresează spre exterior, testându-se următoarea buclă şi păstrând buclele exterioare la valorile minime;
  4. se continuă până când toate buclele sunt testate.




Nested

# LC. Bucle concatenate

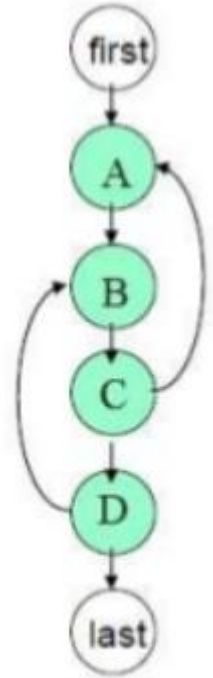
- **bucle concatenate**:
  - dacă buclele sunt indepedente unele de altele:
    - se aplică testarea **buclelor simple**;
  - dacă buclele sunt dependente (e.g., variabila de indexare a primei bucle este valoarea iniţială a celei de a doua):
    - se aplică testarea **buclelor imbricate**.




Concatenated

# LC. Bucle nestructurate

- **bucle nestructurate**:
  - în general, indică folosirea instrucţiunii `goto`;
  - se recomanda restructurarea acestui tip de buclă pentru a reflecta elementele programării structurate.




Unstructured

# TESTARE WHITE-BOX VS. TESTARE BLACK-BOX

Testare White-Box. Avantaje si dezavantaje

Testare White-Box vs. Testare Black-Box

# Testare White-box

## Avantaje

- cazurile de testare sunt proiectate pe baza **structurii interne a codului sursă**, i.e., în funcţie de structurile de programare folosite;
- identifică disfuncţionalităţi în execuţia anumitor secvenţe de cod, e.g., unele structuri de programare nu sunt acoperite;
  - permite acoperirea cu teste a codului scris.

## Dezavantaje

- **nu poate testa cerinţe care nu sunt implementate**, nu poate identifica bug-urile din codul sursă care lipsește;
- **proiectarea cazurilor de testare poate începe doar după implementare**;
- testerul trebuie să cunoască limbajul de programare în care a fost elaborat codul sursă;
- **ineficientă pentru module de mari dimensiuni**.

# Testarea Black-box vs. Testare White-box

## Testare Black-box

- Testare funcţională, testare comportamentală (*engl.* **behavioral testing**);
- cazurile de testare sunt proiectate pe baza specificaţiilor, nu este necesar să avem acces la codul sursă;
- **suprinde ambiguităţile sau inconsistenţele din specificaţii;**
- nu se există informaţii despre implementare;
- activitatea testerului este independentă de cea a programatorului; testerul poate proiecta cazurile de testare înainte de finalizarea codului sursă;
- eficientă şi pentru module de mari dimensiuni.

## Testare White-box

- Testare structurală (*engl.* **structural testing**);
- cazurile de testare sunt proiectate pe baza structurii interne a codului sursă, i.e., în funcţie de structurile de programare folosite;
- **nu poate testa cerinţe care nu sunt implementate;**
- proiectarea cazurilor de testare poate începe doar după implementare;
- ineficientă pentru module de mari dimensiuni – construirea CFG şi calculul CC sunt activităţi costisitoare.

# Testarea Black-box vs. Testarea White-box

**Întrebări:**

- Ce cazuri de testare trebuie actualizate după modificarea specificațiilor și a codului sursă asociat?

- Ce cazuri de testare trebuie actualizate după modificarea codului sursă, fără modificarea specificaţiilor?

# TESTARE BAZATĂ PE EXPERIENŢĂ

Definiţie
Error guessing
Exploratory testing

# Testare bazată pe experienţă. Definiţie. Clasificare

- **testare bazată pe experienţă** (*engl.* **experience-based testing**):
  - cazurile de testare se identifică pe baza abilităţilor, intuiţiei şi experienţei testerului;
- tehnici de testare asociate:
  - *engl.* **error guessing** (**EG**);
    - **scop:** identificarea presupunerilor incorecte ale programatorului, cauzate de specificaţiile incomplete;
  - *engl.* **exploratory testing** (**ET**);
    - **scop:** identificarea defectelor când se utilizează specificaţii insuficiente, există o presiune severă a termenului de predare sau pentru a completa tehnicile de testare formale, e.g., ECP, BVA.

# EG. Proiectarea cazurilor de testare. Reguli

1. se stabileşte o listă cu:
   - greşeli posibile;
   - situaţii care generează bug-uri;
   - presupuneri pe care programatorul le face pe baza specificaţiei;
2. se scriu cazuri de testare pe baza cazurilor speciale enumerate în listă;
   - E.g.:
     - un fişier care trebuie citit este gol sau nu există;
     - la o impărţire numitorul este 0;
     - fişierul începe cu linii fără informaţii;
     - la o operaţie de sortare datele de intrare sunt deja ordonate.

# ÎNTREBĂRI PENTRU EXAMEN

Întrebări cu răspuns scurt

Întrebări cu răspuns lung

# Întrebări cu răspuns scurt

- **Întrebări cu răspuns scurt:**
- **Curs 5:**
  1. Definiţi noţiunea: testare white-box. Exemplificaţi.
  2. Definiţi noţiunea: graful fluxului de control. Exemplificaţi.
  3. Definiţi noţiunea: drum independent în CFG. Exemplificaţi.
  4. Definiţi noţiunea: complexitate ciclomatică. Exemplificaţi aplicarea celor trei formule de calcul.
  5. Definiţi noţiunea: acoperirea drumurilor. Exemplificaţi.
  6. Definiţi noţiunea: acoperirea instrucţiunilor. Exemplificaţi.
  7. Definiţi noţiunea: acoperirea deciziilor. Exemplificaţi.
  8. Definiţi noţiunea: acoperirea condiţiilor. Exemplificaţi.
  9. Definiţi noţiunea: acoperirea deciziilor şi condiţiilor. Exemplificaţi.
  10. Definiţi noţiunea: acoperirea condiţiilor multiple. Exemplificaţi.
- **Curs 6:**
  1. Definiţi noţiunea: acoperirea buclelor. Exemplificaţi tipurile de bucle.
  2. Definiţi noţiunea: error guessing. Exemplificaţi.
  3. Definiţi noţiunea: exploratory testing. Exemplificaţi.
  4. Descrieţi (pe scurt) şi exemplificaţi trei avantaje ale testării white-box.
  5. Descrieţi (pe scurt) şi exemplificaţi trei dezavantaje ale testării white-box.

# Întrebări cu răspuns lung

- **Întrebări cu răspuns lung:**
- **Curs 5**
  1. Comparaţi noţiunile: sc vs. apc. Exemplificaţi.
  2. Comparaţi noţiunile: sc vs. dc. Exemplificaţi.
  3. Comparaţi noţiunile: sc vs. cc. Exemplificaţi.
  4. Comparaţi noţiunile: sc vs. dcc. Exemplificaţi.
  5. Comparaţi noţiunile: sc vs. mcc. Exemplificaţi.
  6. Comparaţi noţiunile: dc vs. cc. Exemplificaţi.
  7. Comparaţi noţiunile: dc vs. dcc. Exemplificaţi.
  8. Comparaţi noţiunile: dc vs. mcc. Exemplificaţi.
  9. Comparaţi noţiunile: cc vs. dcc. Exemplificaţi.
  10. Comparaţi noţiunile: cc vs. mcc. Exemplificaţi.
  11. Comparaţi noţiunile: dcc vs. mcc. Exemplificaţi.
  12. Comparaţi noţiunile: dc vs. cc vs. dcc. Exemplificaţi.
  13. Prezentaţi relaţia dintre: apc, sc, dc, cc, dcc, mcc. Exemplificaţi două tipuri de acoperire.
- **Curs 6:**
  1. Exemplificaţi prin două situaţii concrete avantajele utilizării testării white-box în comparaţie cu testarea black-box.
  2. Este necesară re-proiectarea cazurilor de testare atunci când se modifică specificaţia şi implemenarea unei metode. Dacă da, ce cazuri de testare trebuie modificate? Justificaţi răspunsul. Dacă nu, justificaţi răspunsul. Exemplificaţi.
  3. Este necesară re-proiectarea cazurilor de testare atunci când se modifică implementarea unei metode, fără a modifica şi specificaţia acesteia? Dacă da, ce cazuri de testare trebuie modificate? Justificaţi răspunsul. Dacă nu, justificaţi răspunsul. Exemplificaţi.
  4. Comparaţi noţiunile: lc pentru bucle simple şi lc pentru bucle imbricate. Exemplificaţi.
  5. Comparaţi noţiunile: error guessing şi exploratory testing. Exemplificaţi.

# Referințe bibliografice


- **[Myers2004]** Glenford J. Myers, *The Art of Software Testing,* John Wiley & Sons, Inc., 2004.
- **[NT2005]** K. Naik and P. Tripathy. *Software Testing and Quality Assurance*, Wiley Publishing, 2005.
- **[Patton2005]** R. Patton, *Software Testing,* Sams Publishing, 2005.
- **[Collard2003]** J. F. Collard, I. Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- **[Beizer1990]** Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold., New York, 1990.