

# CURS 03.

# TESTARE WHITE-BOX

---

**Verificarea și validarea sistemelor soft**  
**[13 Martie 2018]**

Lector dr. Camelia Chisăliță-Crețu  
Universitatea Babeș-Bolyai

# Conținut

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

# CRITERII DE TESTARE

---

Criteriu de testare

Clasificarea criteriilor de testare

Tehnici de testare asociate

# Criteriu de testare. Clasificare

- criteriu de testare
  - ansamblu de condiții prin care se stabilesc cazurile de testare;
- clasificare
  - criteriul cutiei negre (testare Black-Box);
  - criteriul cutiei transparente (testare White-Box);
  - criteriul statistic.

# Tehnici de testare asociate

- criteriul cutiei negre (testare Black-Box) – testare funcțională:
  - Partiționarea în clase de echivalență;
  - Analiza valorilor limită;
  - Tabele de decizie, Grafe de tranziție a stărilor, Scenarii de execuție ale cazurilor de utilizare, etc.;
- criteriul cutiei transparente (testare White-Box) – testare structurală:
  - Acoperirea fluxului de control (e.g., instrucțiuni, decizii, condiții, bucle, drumuri);
  - Acoperirea fluxului de date;
- criteriul statistic:
  - generarea aleatoare de date de test pe baza unor modele;
  - experiența testerului.

# TESTARE WHITE-BOX

---

Definiție. Caracteristici.

Tehnici de testare white-box

# Testare White-Box. Definiție. Caracteristici

- **criteriul cutiei transparente** (*engl. white-box testing, logic driven testing*):
  - testare structurală;
  - datele de intrare se aleg pe baza instrucțiunilor care trebuie executate, programul este văzut ca o cutie transparentă;
  - avem acces la structura internă a programului (codul sursă);
  - permite identificarea situațiilor în care execuția 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:
      - instrucțiunilor, ramificațiilor, deciziilor, condițiilor, deciziilor și condițiilor, condițiilor multiple [[Myers2004](#)];
      - buclelor [[Collard2003](#)];
  2. fluxul de date.



# TESTARE BAZATĂ PE FLUXUL DE CONTROL. COMPONENTE

---

Definiție. Caracteristici.

Graful fluxului de control. Exemple

Drumuri în CFG. Exemple

Complexitatea ciclomatică. Exemple

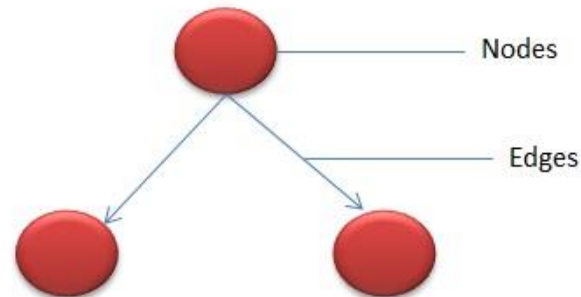
Testare bazată pe fluxul de control – Avantaje și dezavantaje

# 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 satisfăcător a structurilor de control din programul testat;
- **componente:**
  - graful fluxului de control;
  - complexitatea ciclomatică;
  - criterii de acoperire a drumurilor.

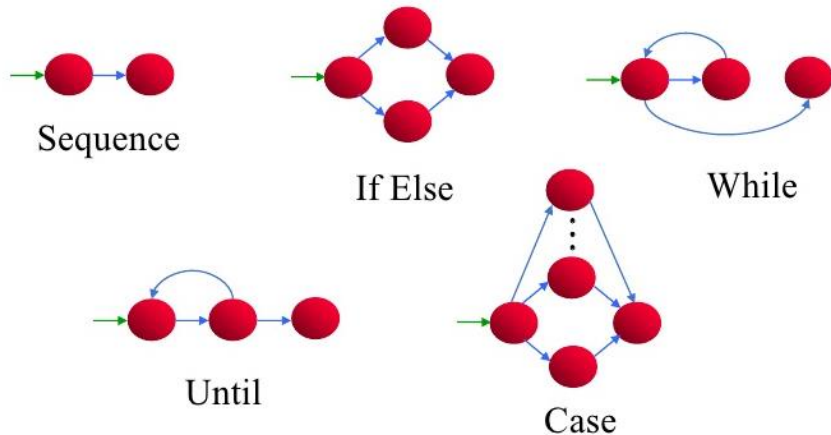
# Graful fluxului de control. Definiție

- **graful fluxului de control** (*engl. Control Flow Graph, CFG*):
  - reprezentare grafică detaliată a unei unități de program;
  - permite vizualizarea tuturor drumurilor din unitatea de program;
- **graf orientat**:
  - **vârf** (*engl. node*):
    - indică structuri secvențiale și condițiile 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 condiție prin care se permite ramificarea execuției prin cel puțin două căi;
    - e.g., instrucțiunile `if`, `while`, `repeat/until`, `case`;
  - **conector:**
    - nu conține o instrucțiune și reprezintă un punct al programului care unește mai multe ramificații;
  - **instrucțiune/calcul:**
    - conține o secvență de instrucțiuni;
  - **intrare, ieșire:**
    - există un singur vârf intrare și un singur vârf ieșire;
    - în vârful de intrare nu intră nici un arc;
    - din vârful de ieșire nu iese nici un arc.



# CFG. Construire

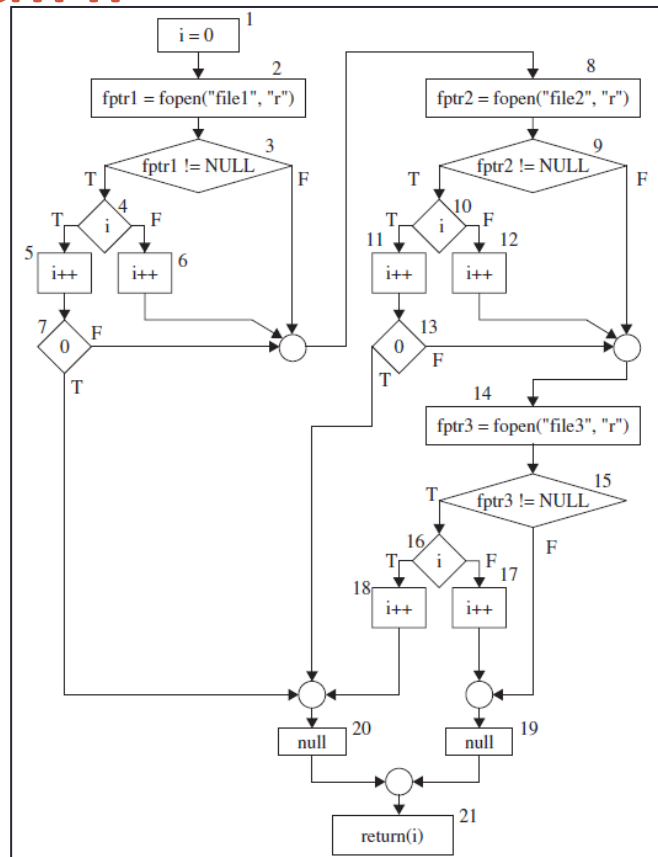
- pași de elaborare a unui CFG:
  1. se numerotează unic fiecare element de structură secvențială (calcul) și condițională (decizie);
  2. se începe pornind de la vârful de intrare, care are (de obicei) numărul 1;
  3. se adaugă celelalte vârfuri corespunzătoare structurilor numerotate și se unesc prin arce, evidențiind transmiterea controlului în cadrul programului;
  4. la final, toate ieșirile posibile din program se unesc în vârful de ieșire;
- condiții complexe care conțin atribuiri ==> CFG are o descriere complexă [[NT2005](#)];
  - e.g., `if (((fptrl = fopen(''file1'', ''r'' )) != NULL) && (i++) && (0)){...}`.

# CFG. Complexitatea construirii

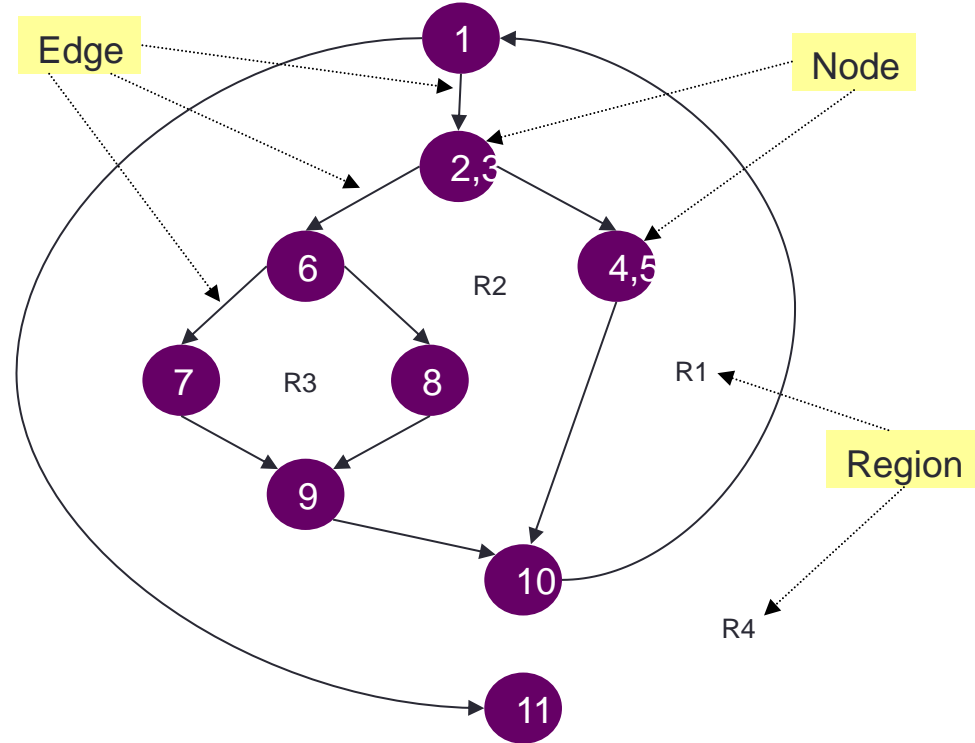
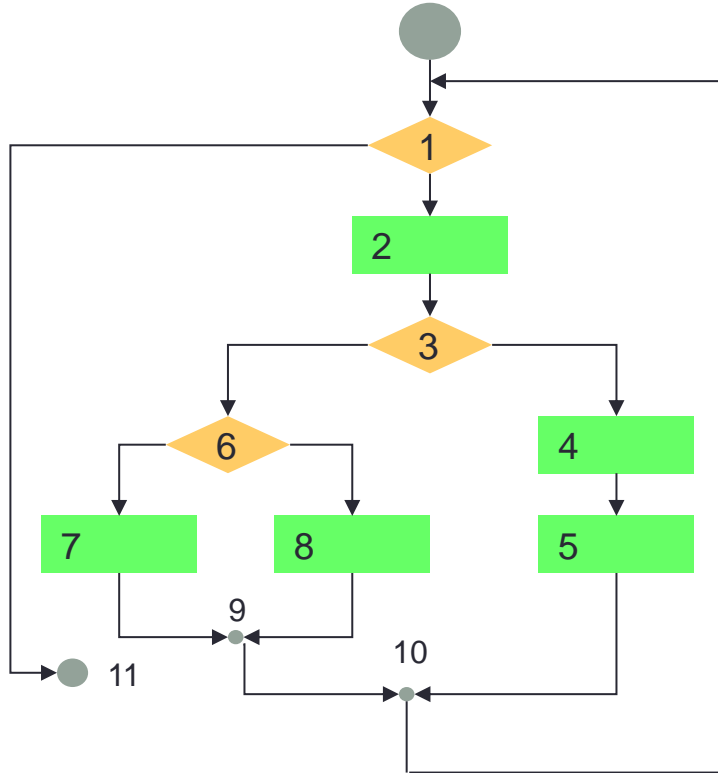
- pentru metoda `openfiles()`, avem CFG alăturat;

```
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);
}
```

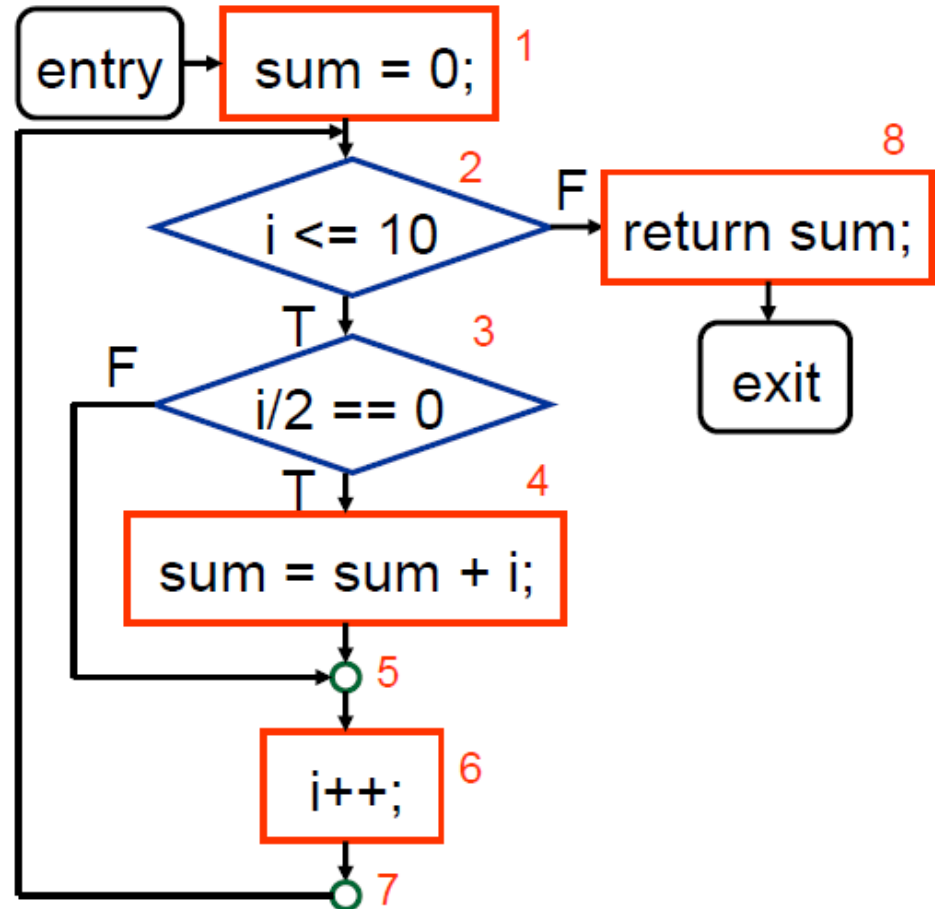


# CFG. Exemple de notații



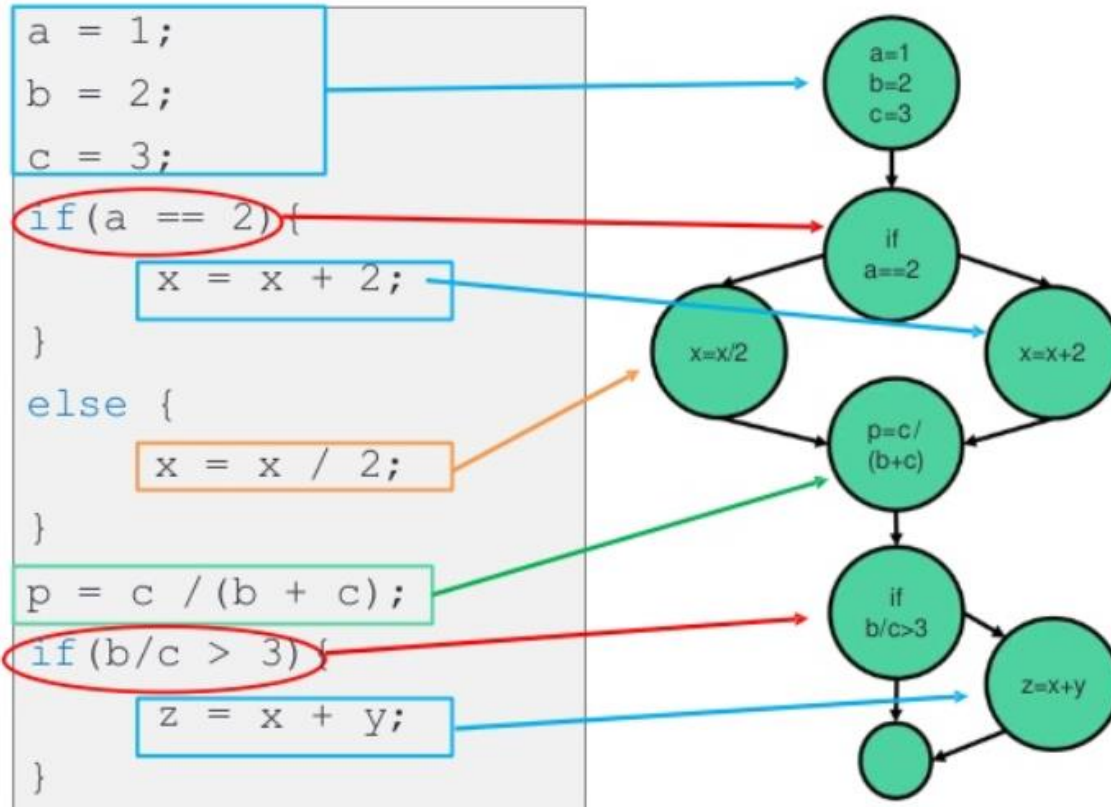
# CFG. Example (1)

*	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;
*	}





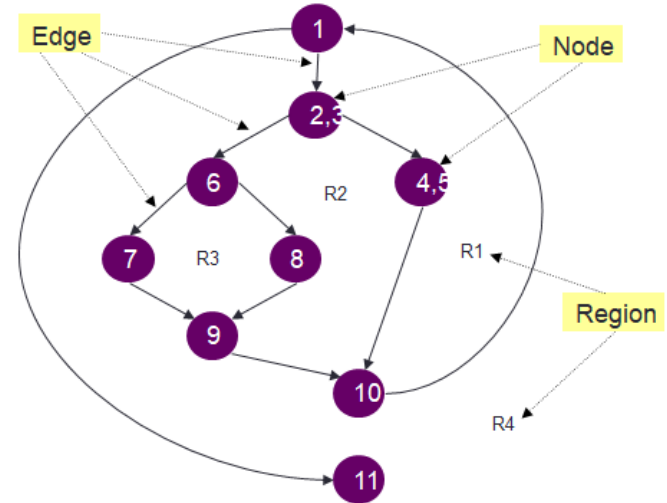
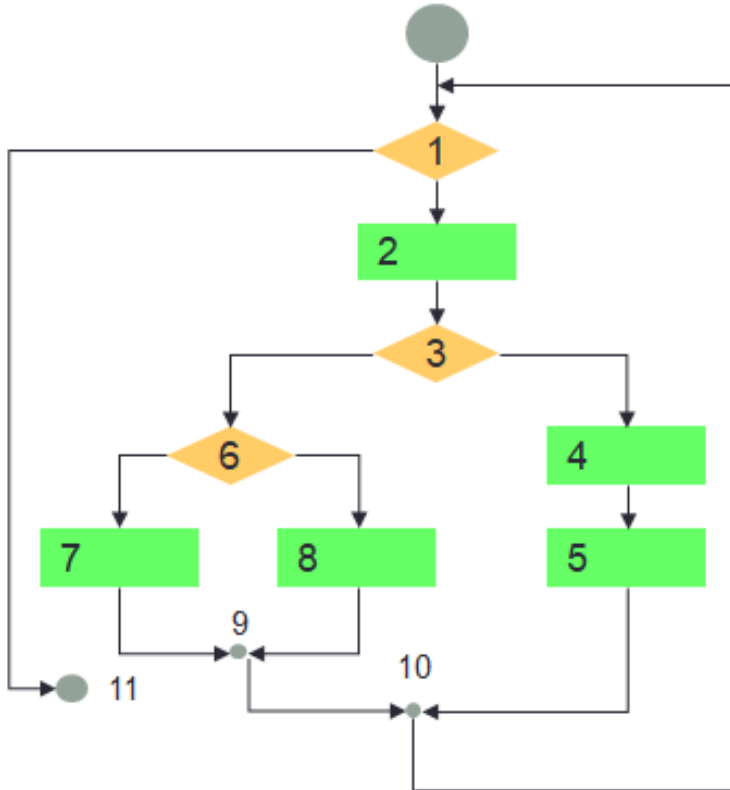
# CFG. Example (2)



# CFG. Drumuri în CFG

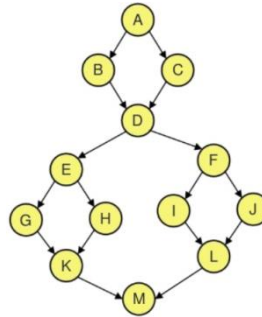
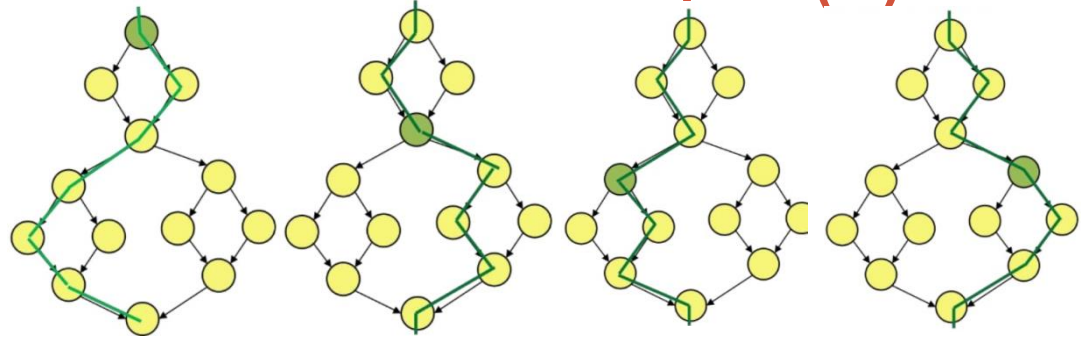
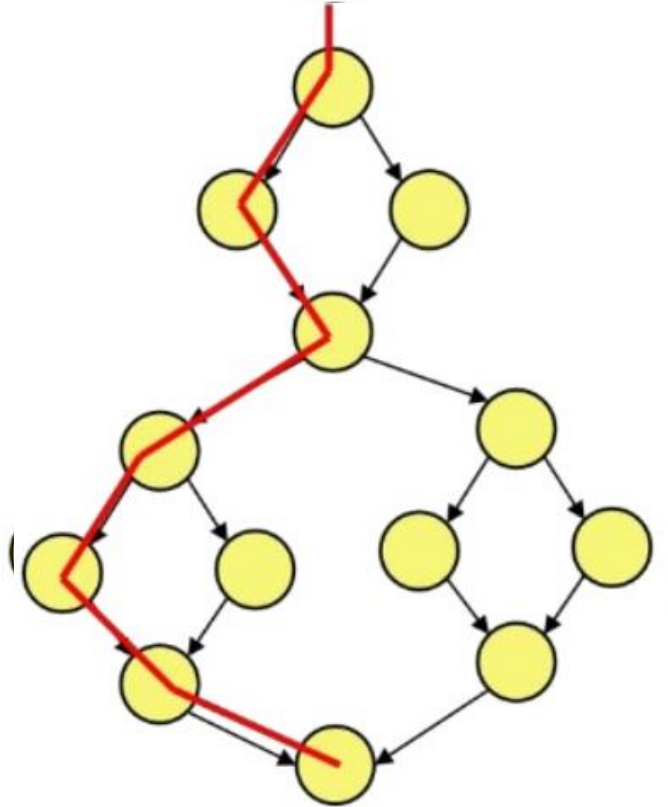
- **drum:**
  - execuția unei secvențe de instrucțiuni **de la punctul de intrare până la punctul de ieșire** al CFG asociat unei unități de program;
- **drum independent** (*engl. independent path*):
  - orice drum în CFG care introduce **cel puțin o instrucțiune nouă sau o condiție nouă**, care este executată cel puțin o dată;
- mulțimea drumurilor independente formează mulțimea drumurilor de bază (*engl. basis path set*) a unui CFG;
  - indică numărul maxim de cazuri de testare care trebuie executate pentru ca toate instrucțiunile să fie executate cel puțin 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.

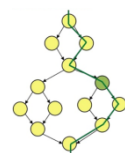
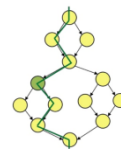
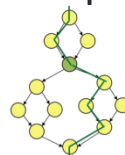
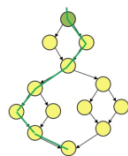
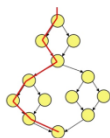
# Drumuri independente în CFG. Exemple (2)



- 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

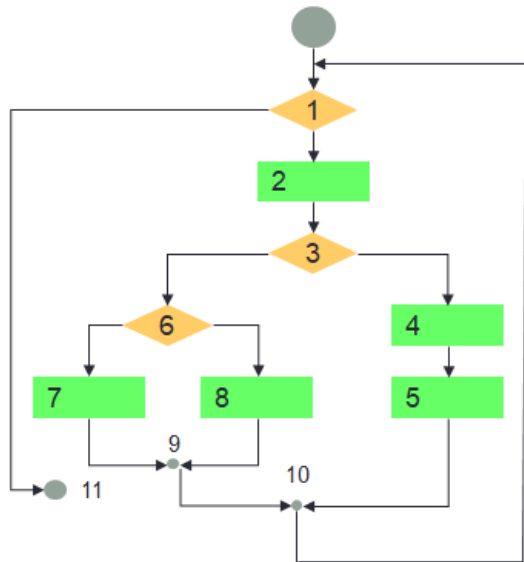
- **Algoritmul lui McCabe [McCabe1984, McCabe Baseline Method]**
  1. se alege un drum normal (numit **drum inițial, D1**); se recomandă ca acesta să aibă cât mai multe decizii este posibil;
  2. pentru generarea următorului drum (**D2**), se modifică rezultatul evaluării primei decizii de pe D1 și păstrând 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 pașii de la 1..4 considerând ca drum inițial pe D2, modificând/inversând deciziile, până când toate se obțin toate drumurile independente din mulțimea de bază a CFG.



# Complexitatea ciclomatică. Definiție

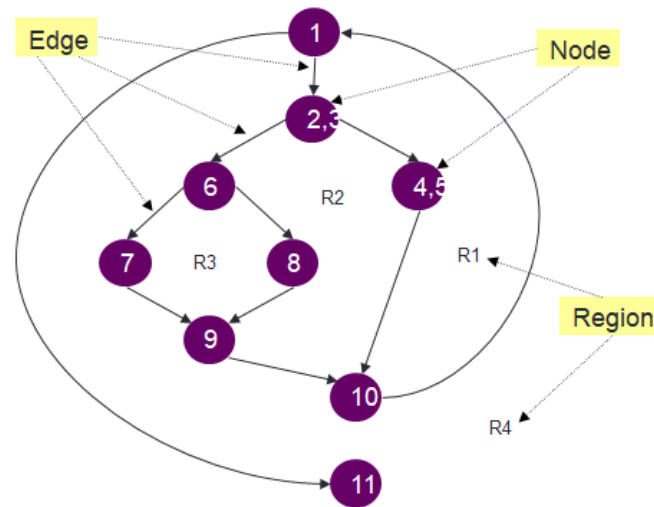
- **complexitatea ciclomatică** (*engl. cyclomatic complexity, CC*):
  - *metrică software* aplicată pentru măsurarea cantitativă a complexității logice a unui program;
  - *permite determinarea numărului de drumuri independente din mulțimea de bază a unui CFG;*
- modalități de calcul a CC la nivelul CFG:
  - **CC = numărul de regiuni din CFG;**
  - **$CC = E - N + 2$ , unde  $E$  - #arce,  $N$  - #vârfuri ;**
  - **$CC = P + 1$ , unde  $P$  - #vârfuri condiție.**
- **regiune:**
  - **zonă a CFG marginită parțial sau în totalitate de arce și vârfuri;**

# CC. Exemple (1)



- CC pentru CFG:

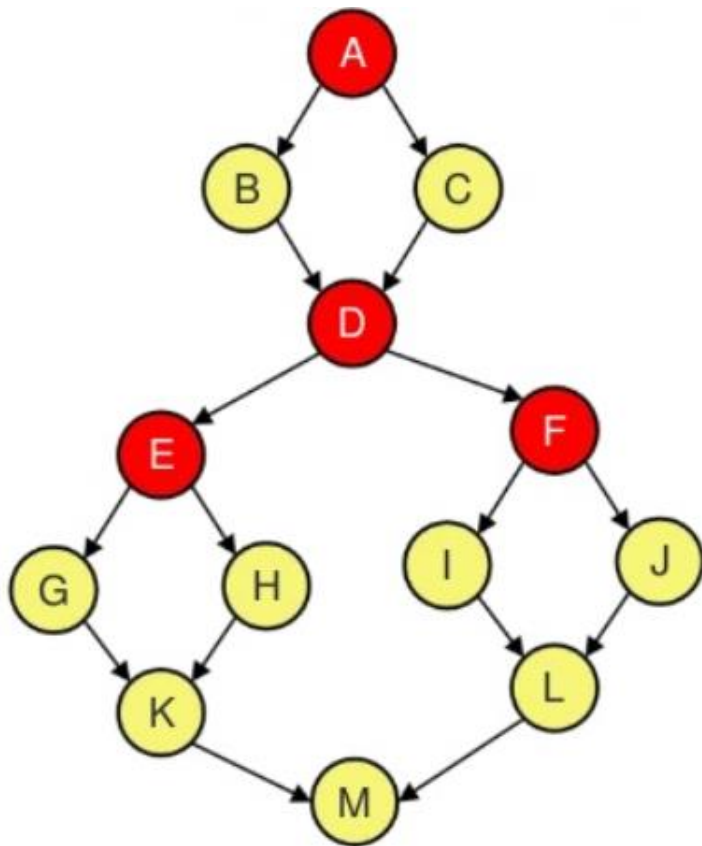
- $CC = \text{numărul de regiuni} = 4 \text{ regiuni} = 4.$
- $CC = E - N + 2 = 14 \text{ arce} - 12 \text{ vârfuri} + 2 = 4.$
- $CC = P + 1 = 3 \text{ vârfuri condiție} + 1 = 4.$



- 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)



- CC pentru CFG:
  - $CC = \text{numărul de regiuni} = 5 \text{ regiuni} = 5;$
  - $CC = E - N + 2 = 16 \text{ arce} - 13 \text{ vârfuri} + 2 = 5;$
  - $CC = P + 1 = 4 \text{ vârfuri condiție} + 1 = 5.$
- 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.



# Testarea bazată pe CFG. Avantaje și 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 și nu de programator, este necesar ca testerul să aibă abilități de programare pentru a înțelege codul sursă și modul de execuție al acestuia;
- Tehnică de testare consumatoare de timp, deoarece mai întâi se elaborează CFG, CC, drumuri independente și ulterior se proiectează cazurile de testare;
- Consideră că:
  - Specificațiile sunt corecte;
  - Datele sunt definite și accesate corespunzător;
  - Nu există alte bug-uri pe lângă cele determinate de fluxul de control;

# TESTARE BAZATĂ PE ACOPERIREA DRUMURILOR

---

Definiție

Algoritm

Exemplu

# Testare bazată pe acoperirea drumurilor. Definiție

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

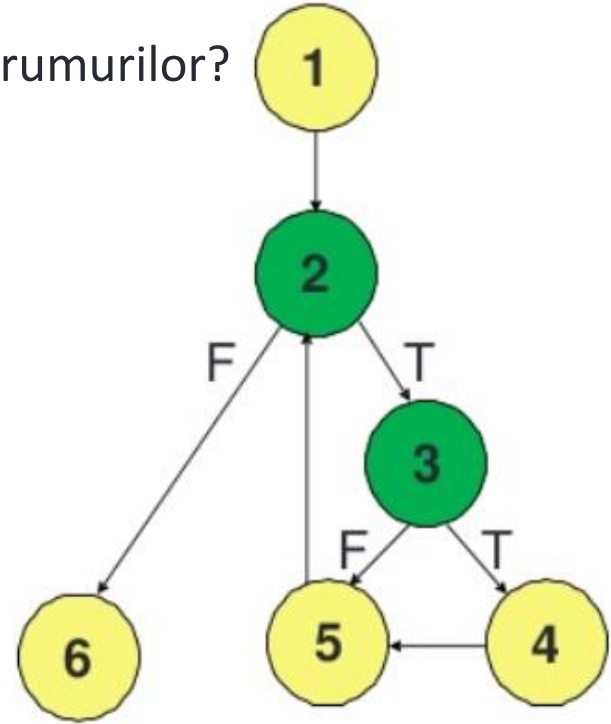
# Testare bazată pe acoperirea drumurilor. Algoritm

- 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 mulțimea de bază a CFG (cu drumuri independente, liniare);
  4. Se proiectează câte un caz de testare pentru fiecare drum independent identificat.

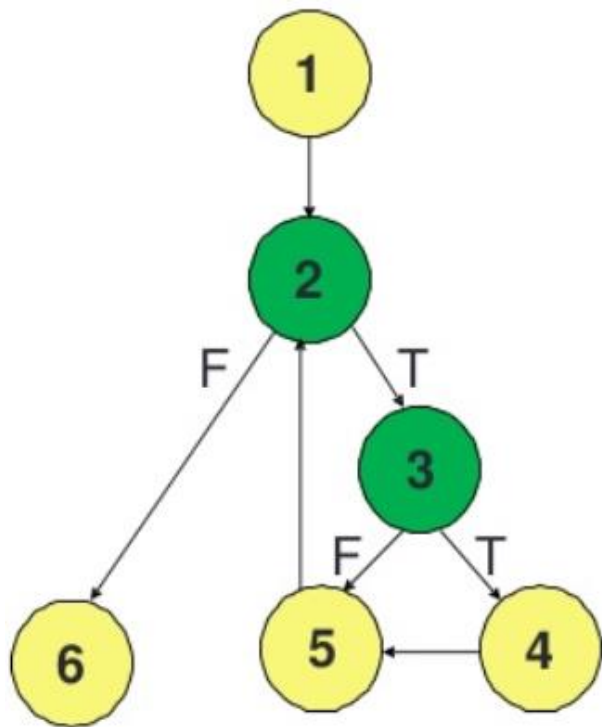
# Testare bazată pe acoperirea drumurilor. Exemplu

Câte cazuri de testare sunt necesare pentru acoperirea 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:
  - $CC = \text{numărul de regiuni} = 3 \text{ regiuni} = 3;$
  - $CC = E - N + 2 = 7 \text{ arce} - 6 \text{ vârfuri} + 2 = 3;$
  - $CC = P + 1 = 2 \text{ vârfuri condiție} + 1 = 3.$
- 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

---

Definiție. Criterii de acoperire

Acoperirea instrucțiunilor. Definiție. Exemplu

Acoperirea deciziilor. Definiție. Exemplu

Acoperirea condițiilor. Definiție. Exemplu

Acoperirea deciziilor și condițiilor. Definiție. Exemplu

Acoperirea condițiilor multiple. Definiție. Exemplu

Acoperirea buclelor. Definiție. Exemplu

# Testare bazată pe acoperirea codului sursă. Definiție

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



# Acoperirea instrucțiunilor. Definiție

- **acoperirea instrucțiunilor** (*engl. statement/line/node coverage, sc*):
  - proiectarea cazurilor de testare astfel încât toate instrucțiunile sunt executate cel puțin o dată, adică fiecare vârf al CFG este vizitat;
  - **cel mai slab criteriu de acoperire în testare;**
    - o mulțime de teste care nu realizează acoperire 100% a vârfurilor nu este considerată acceptabilă;
- **ordinea de selectare a drumurilor:**
  - drumuri scurte;
  - drumuri de lungime crescândă;
  - drumuri lungi, complexe, alese arbitrar.

## SC. Exemplu

Fiecare instrucțiune trebuie să fie executată cel puțin o dată.

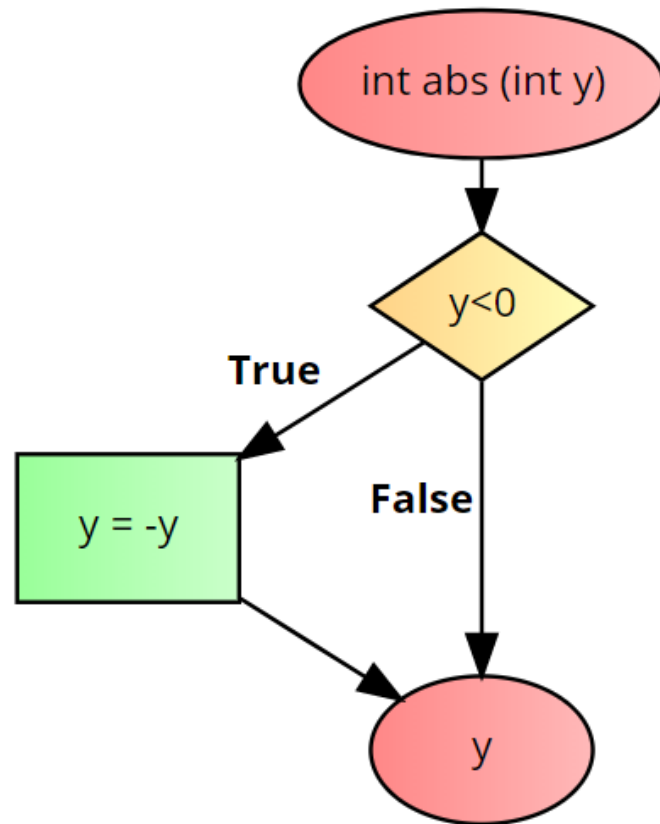
// returnează valoarea absolută a lui y

```
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 instrucțiunilor este îndeplinit 100%.**



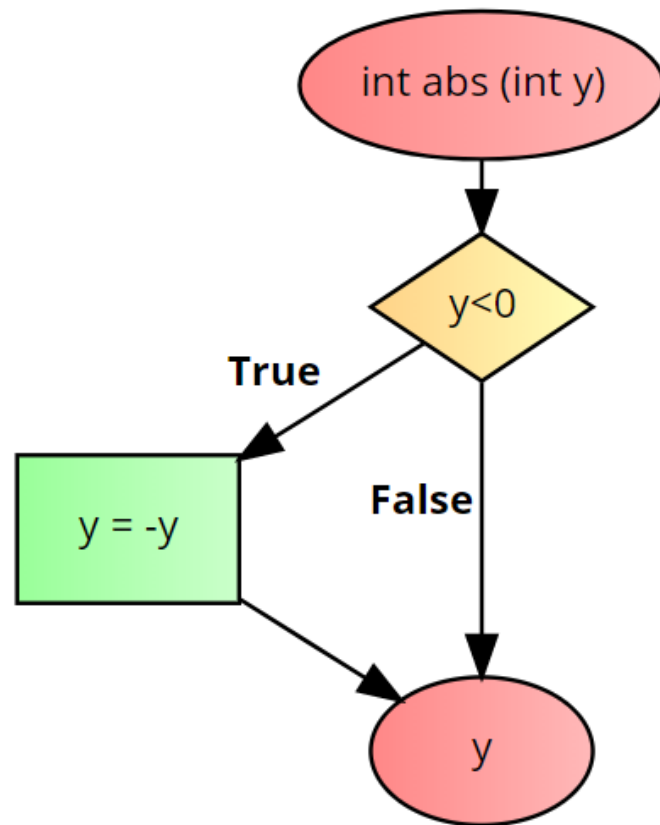
## SC. Observații

- SC se recomandă doar atunci când nu există alte criterii de acoperire care să fie aplicate.

// returnează valoarea absolută a lui y

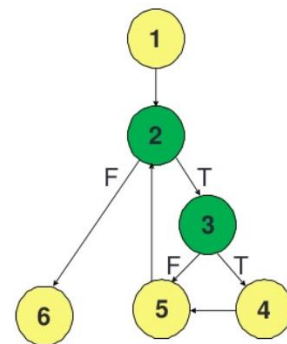
```
int abs (int y) {  
    if (y<0)  
        y = -y;  
    return y;  
}
```

**Este necesară acoperirea deciziilor.**



# Acoperirea deciziilor. Definiție

- **ramificație** (*engl.* **branch/edge**):
  - arc care pornește dintr-un vârf;
  - din fiecare vârf pornește cel mult un arc, mai puțin din vârful de ieșire al CFG;
  - din vârfurile de decizie pornesc două arce, etichetate cu true și 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 puțin o dată;
- regulă de selectare:
  - fiecare decizie selectată, evaluată la `true` sau `false`, trebuie să se găsească pe cel puțin un drum.



# DC. Exemplu

Pentru fiecare decizie, fiecare ramificație (true, false) trebuie să fie executată cel puțin o dată.

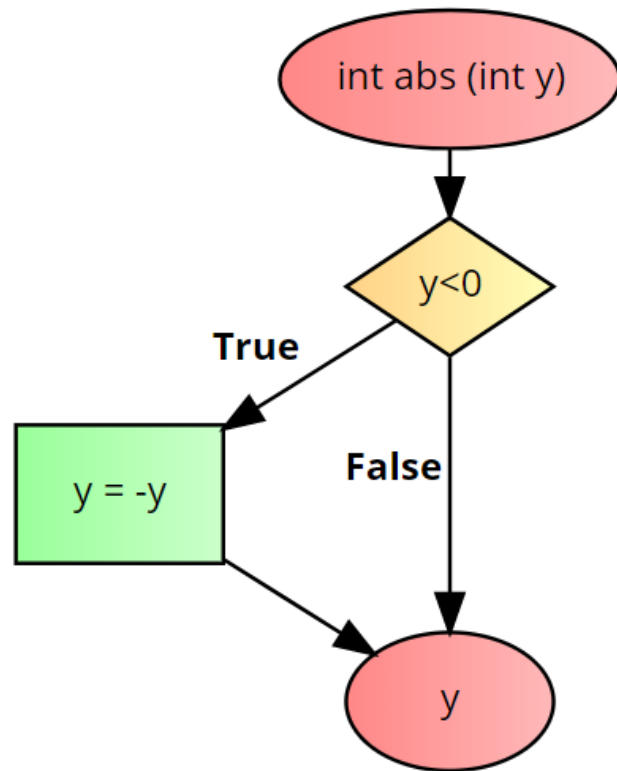
// returnează valoarea absolută a lui y

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

Care este numărul minim de cazuri de testare necesar ?

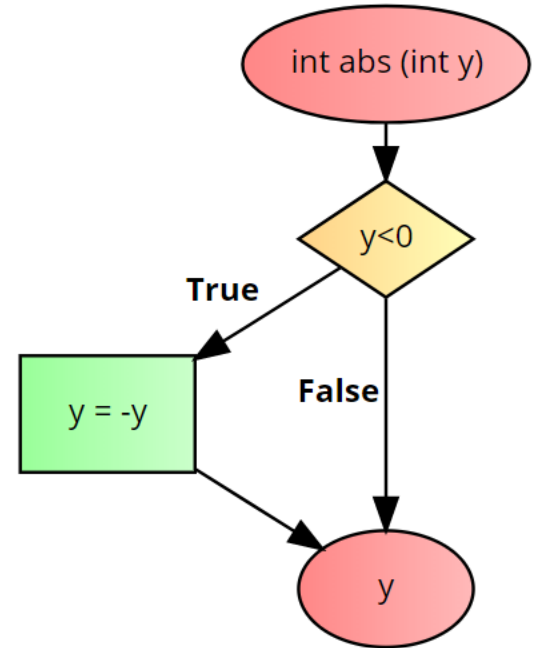
TC	Input	Decision (y<0)	Expected result	Actual result
1	-2	true	2	2
2	3	false	3	3

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



# DC vs. SC

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

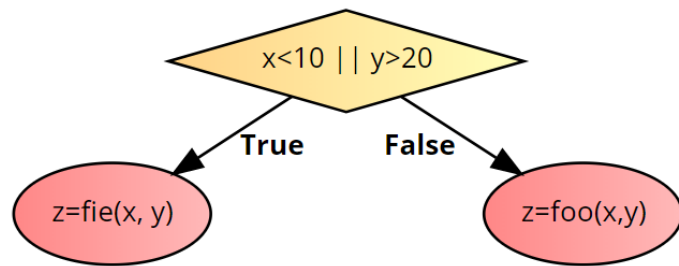


## DC. Observații

- e.g., 

```
if (x<10 || y>20)
    {z=fie(x, y);}
else {z=foo(x,y);}
```

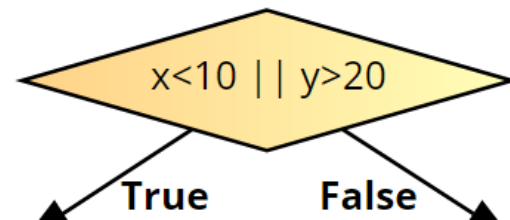
  - pentru  $x=1$  și  $y=2$  **nu** mai este relevantă evaluarea condiției  $y>20$ ;
- în deciziile formate din mai multe condiții, unele condiții pot să rămână neacoperite fiind irelevante pentru rezultatul final al deciziei;
- dacă condiția este scrisă greșit, e.g.,  $y>20$  în loc de  $y<20$ , cazuri de testare ca  $x=1, y=2$  nu evidențiază defectul.



**Este necesară acoperirea condițiilor.**

# Acoperirea condițiilor. Definiție

- **condiție:**
  - expresie logică dintr-un vârf de decizie;
  - o decizie este formată din una mai multe condiții;
- **acoperirea condițiilor** (*engl. condition coverage, cc*):
  - proiectarea cazurilor de testare se realizează astfel încât fiecare condiție din fiecare decizie ia fiecare dintre valorile posibile, cel puțin o dată;
- **regulă de selectare:**
  - pentru fiecare decizie care conține mai multe condiții, fiecare condiție selectată va fi evaluată la `true` sau `false` și se va găsi pe cel puțin un drum.





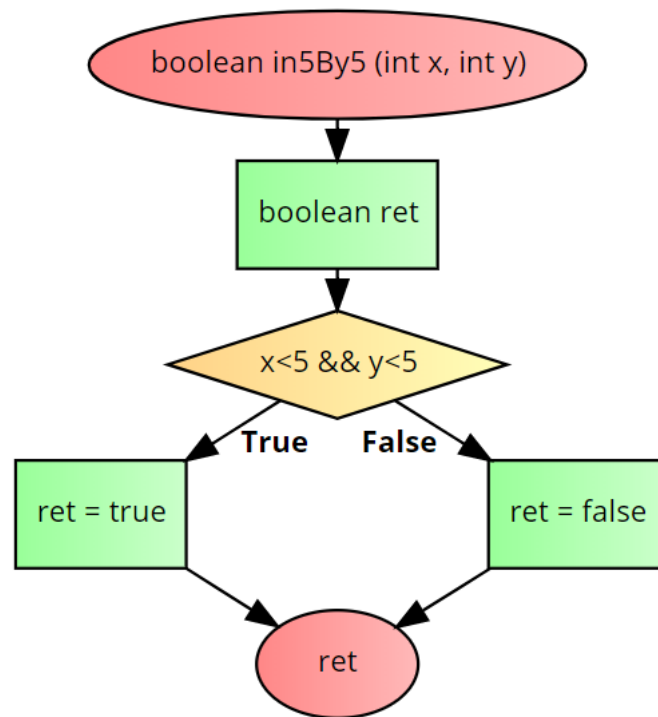
# CC. Exemplu

Fiecare condiție din fiecare decizie trebuie să fie executată cel puțin 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).

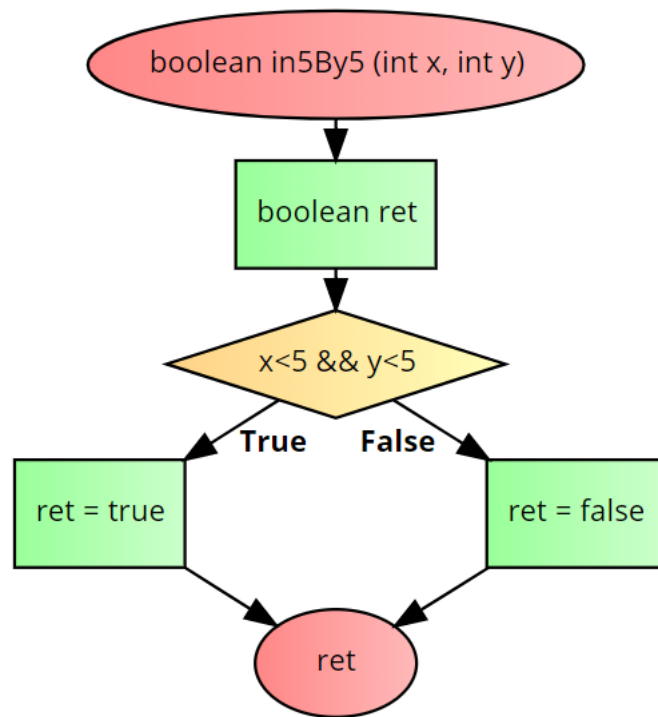
```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

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 condițiilor este îndeplinit 100%.**

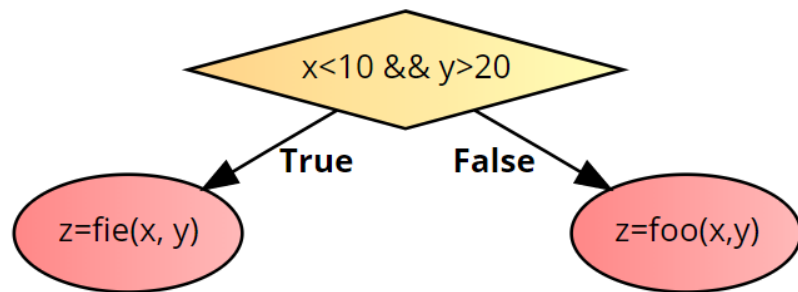
**Toate rezultatele posibile ale evaluării condițiilor din decizie au fost explorate.**



## CC. Observații

- e.g., 

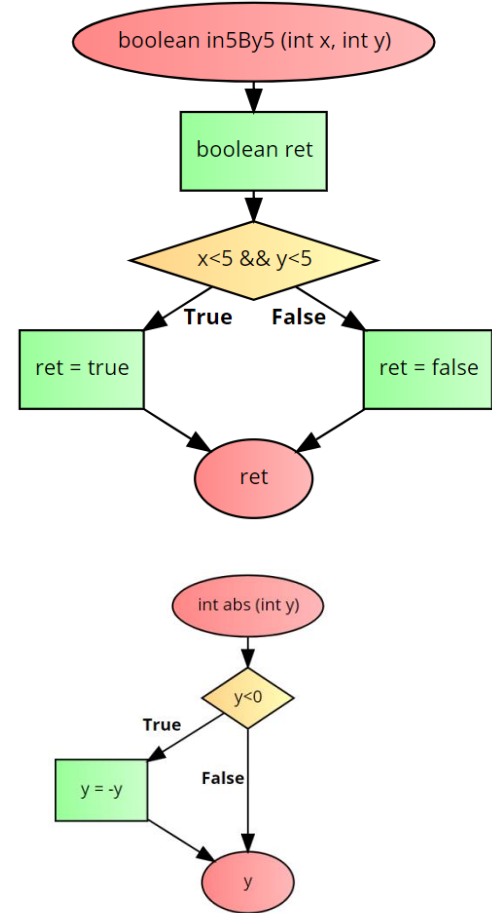
```
if (x<10 && y>20)
    {z=fie(x, y);}
else {z=foo(x,y);}
```
- pentru  $x=11$  și  $y=21$ , avem  $\text{false} \ \&\& \ \text{true} = \mathbf{false}$ ;
- pentru  $x=1$  și  $y=1$ , avem  $\text{true} \ \&\& \ \text{false} = \mathbf{false}$ ;
- fiecare *condiție* selectată este acoperită prin evaluarea la `true` și `false`, dar *decizia* nu este acoperită, doar ramificația `false` este explorată;



**Este necesară acoperirea deciziilor și condițiilor.**

# CC vs. DC

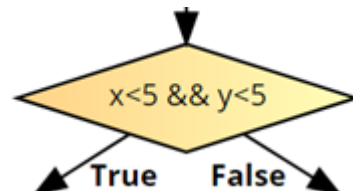
- în general, **cc**  $\implies$  **dc**;
  - prin acoperirea condițiilor se poate acoperi și decizia;
- caz particular:
  - **cc = dc** atunci când decizia conține doar o condiție;
    - e.g., *decizia*  $y < 0$  este evaluată la `true` sau `false`, similar cu evaluarea *condiției*  $y < 0$ , care este evaluată la `true` sau `false`  $\implies$  **acoperirea condiției este similară cu acoperirea deciziei**;



# Acoperirea deciziilor și condițiilor. Definiție

- **acoperirea deciziilor și condițiilor** (*engl. decision and condition coverage, dcc*):

- proiectarea cazurilor de testare astfel încât:
  - fiecare condiție din fiecare decizie ia toate valorile posibile, cel puțin o dată;
  - fiecare decizie ia toate valorile posibile cel puțin o dată;



- **regulă de selectare:**

- pentru fiecare decizie care conține mai multe condiții, fiecare condiție selectată va fi evaluată la `true` sau `false` și împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puțin un drum.

# DCC. Exemplu

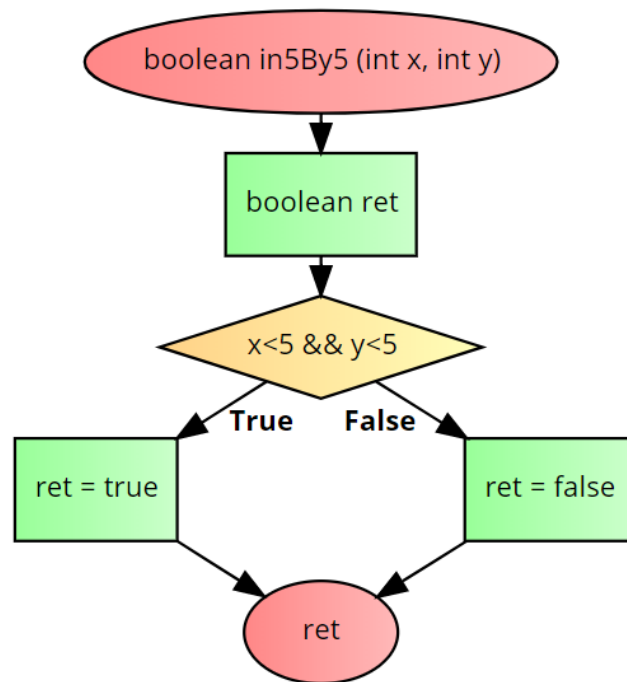
Fiecare *condiție* din fiecare decizie trebuie să fie executată cel puțin o dată cu fiecare din valorile posibile, e.g., `true`, `false`.

Fiecare *decizie* trebuie să fie executată cel puțin 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).

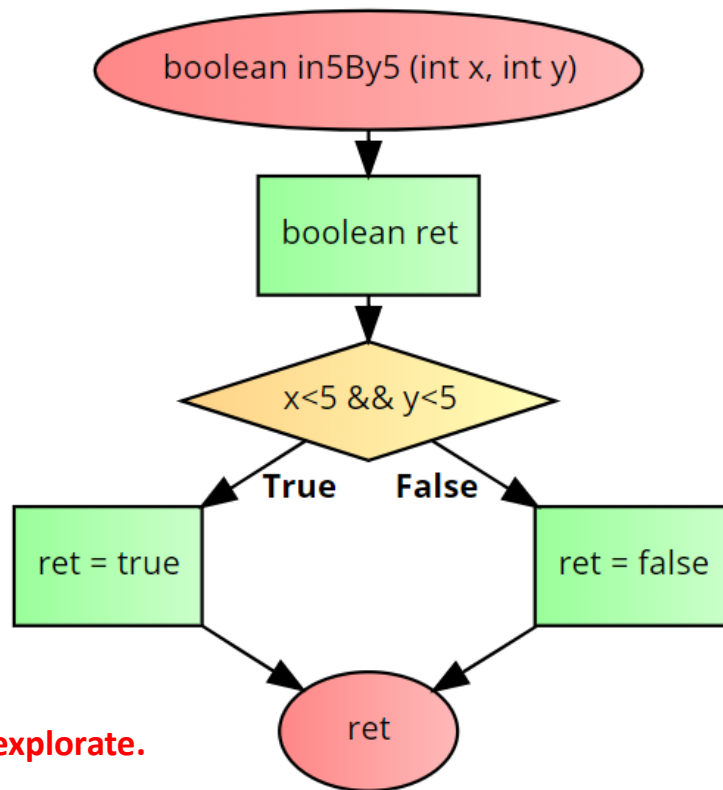
```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

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 și condițiilor este îndeplinit 100%.**

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

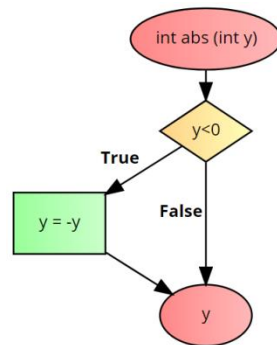
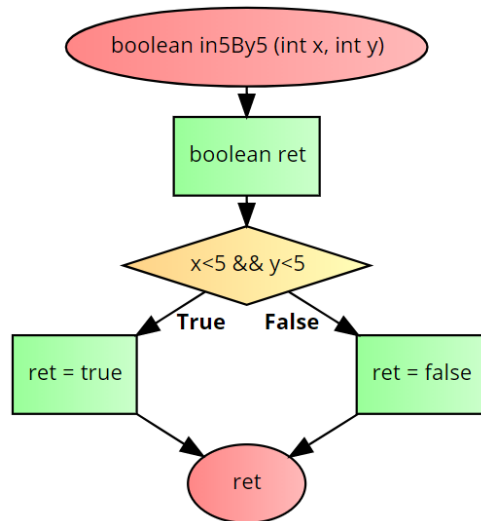
**Ambele ramificații ale deciziei au fost explorate.**



# DCC vs. CC, DCC vs. DC

- **dcc ==> cc;**
  - prin acoperirea deciziilor și condițiilor se acoperă condițiile;
- **dcc ==> dc;**
  - prin acoperirea deciziilor și condițiilor se acoperă deciziile;
- caz particular:
  - **dcc = dc și dcc = cc atunci când decizia conține doar o condiție;**
    - e.g., decizia `y<0` este evaluată la `true` sau `false`, similar cu evaluarea condiției `y<0`, care este evaluată la `true` sau `false` ==> **acoperirea deciziei și condiției (dcc) este similară cu acoperirea deciziei (dc), care este similară cu acoperirea condiției (cc);**

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

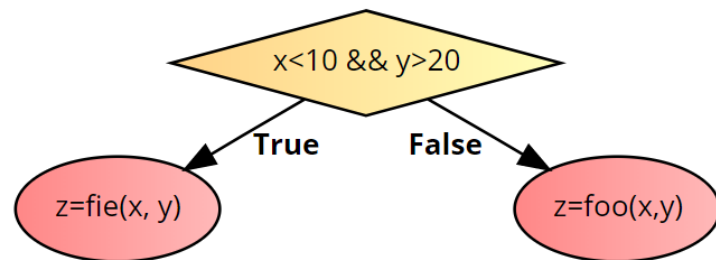
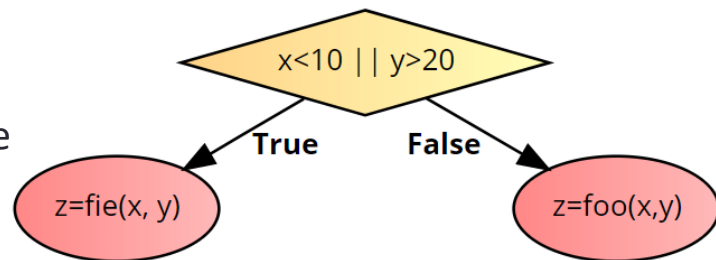




# DCC. Observații

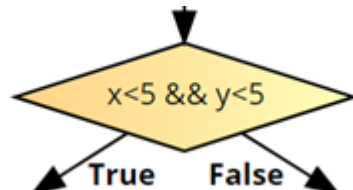
- condițiile logice care folosesc operatorii `&&` și `||` nu pot fi acoperite prin **dcc**, deoarece compilatorul realizează diverse optimizări (scurt-circuitare la evaluare);

**Este necesară acoperirea condițiilor multiple.**



# Acoperirea condițiilor multiple. Definiție

- **acoperirea condițiilor multiple** (*engl. multiple condition coverage*, **mcc**):
  - proiectarea cazurilor de testare se realizează astfel încât:
    - toate combinațiile posibile ale valorilor de ieșire ale unei condiții, în fiecare decizie, să fie parcurse cel puțin o dată;
- regulă de selectare:
  - fiecare decizie care conține mai multe condiții, va combina fiecare condiție selectată care este evaluată la `true` sau `false` cu celelalte condiții în toate variantele posibile și împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puțin un drum.



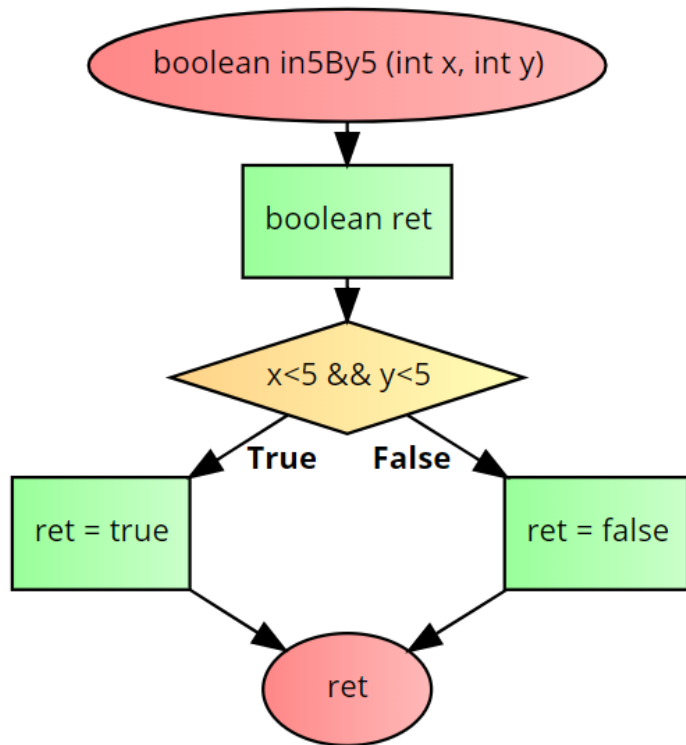
# MCC. Exemplu

Fiecare *condiție* din fiecare decizie trebuie să fie executată în toate combinațiile posibile cu toate celelalte condiții din cadrul aceleași 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 ?



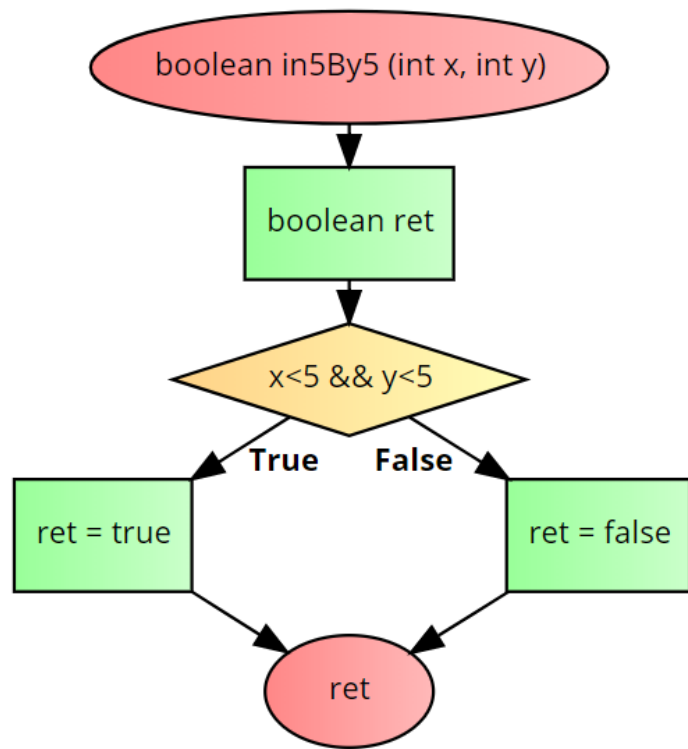
# MCC. Exemplu (cont.)

// 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 ?

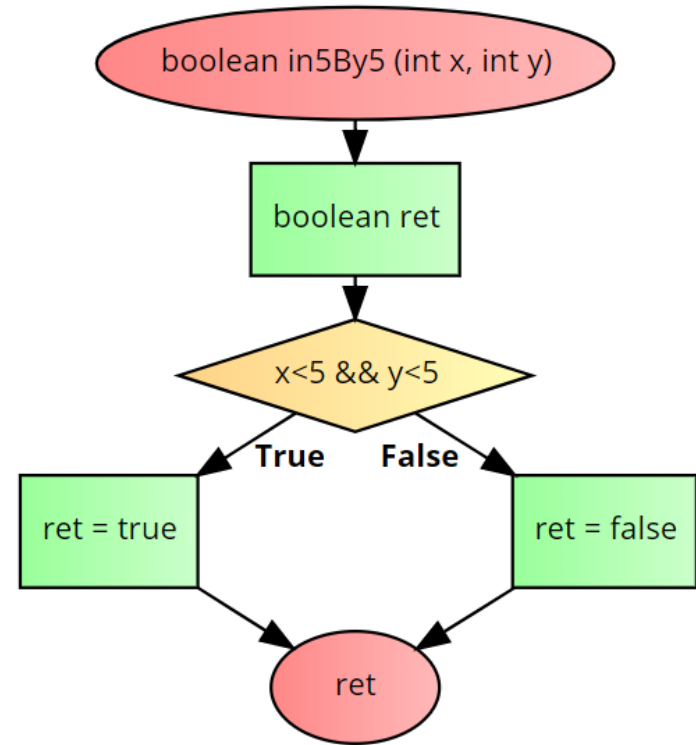
TC	x	y	Decision (x<5) && (y<5)	Expected result	Actual result
1	2	3	T && T = T	true	true
2	9	7	F && F = F	false	false
3	2	7	T && F = F	false	false
4	9	3	F && T = F	false	false



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

# MCC vs. DCC

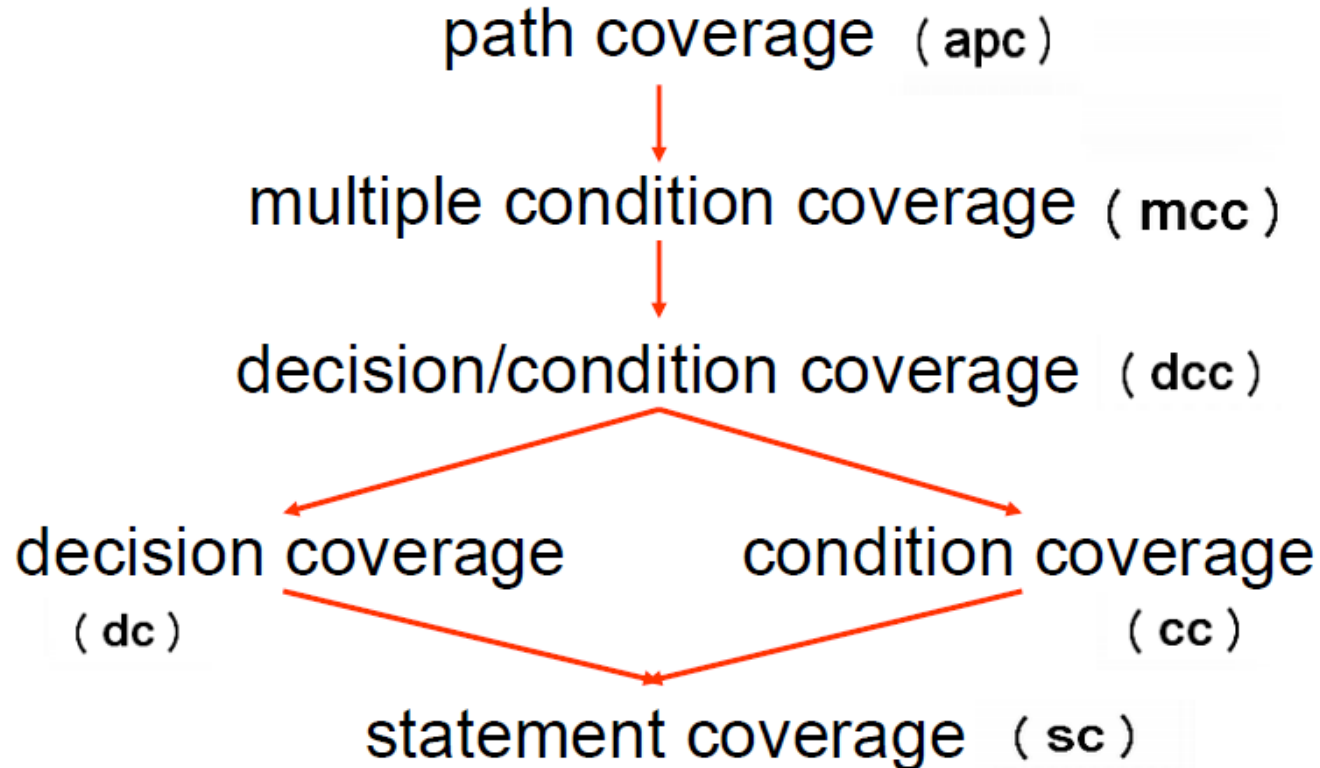
- **mcc ==> dcc;**
  - prin acoperirea multiplă a condițiilor se acoperă deciziile și condițiile;



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

- dacă programul are o singură condiție în fiecare vârf de decizie, atunci se aplică
  - **acoperirea deciziilor (dc)**, în acest caz **dc = cc**;
- dacă programul are condiții multiple în vârfuri de decizie, atunci se aplică
  - **acoperirea condițiilor multiple (mcc)**.

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



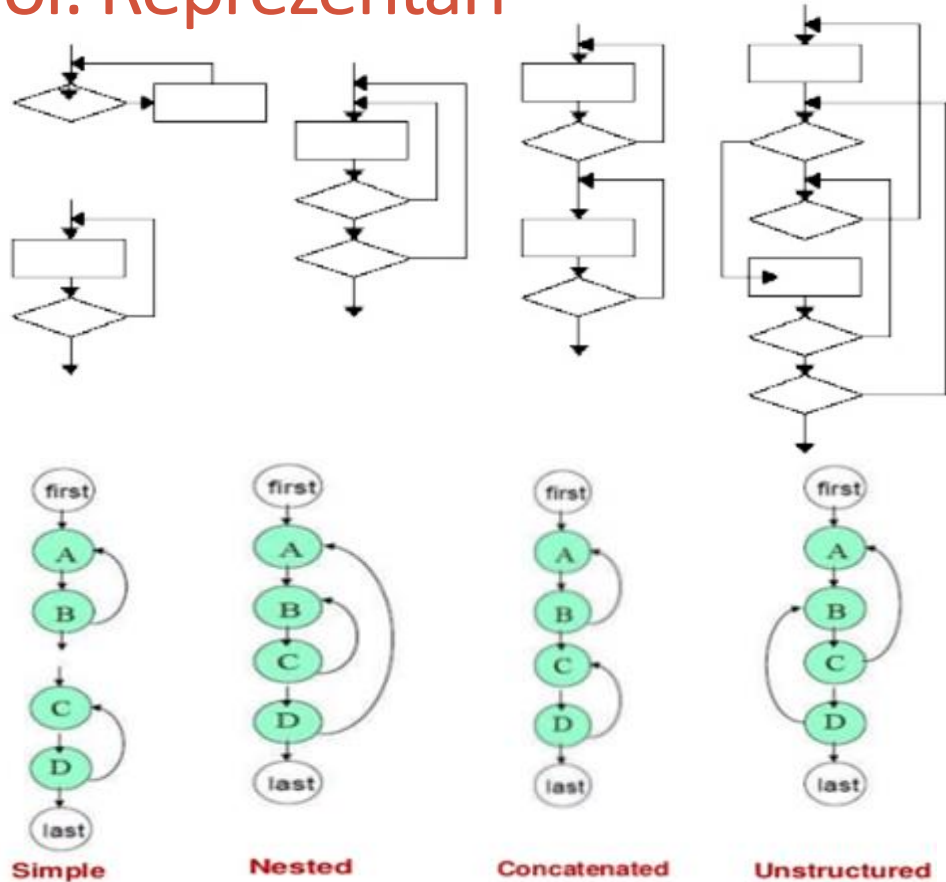
# Acoperirea buclelor. Definiție

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



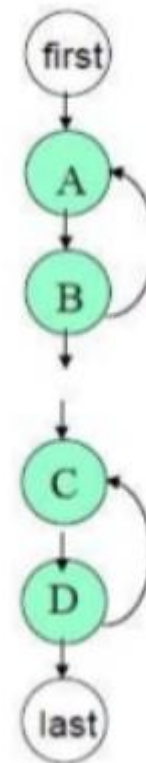
# LC. Clasificarea buclelor. Reprezentări

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



## LC. Bucle simple

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

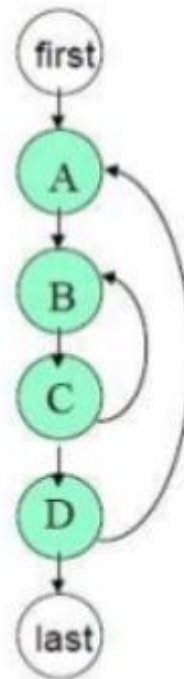


Simple

# LC. Buclă imbricate

- **bucle imbricate:**

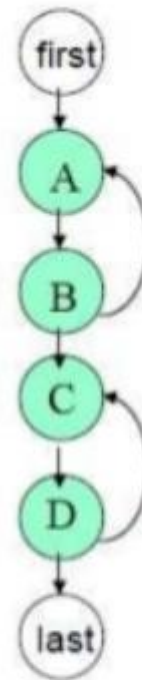
1. se pornește de la bucla cea mai interioară; toate celelalte bucle sunt setate pe valori minime;
2. se testează bucla cea mai interioară ca și buclă simplă, păstrând buclele exterioare la valoarea minimă a parametrului de iterație;
3. se progresează spre exterior, testându-se următoarea buclă, dar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



**Nested**

# LC. Buclă concatenate

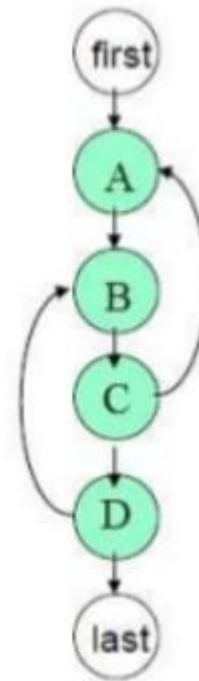
- **bucle concatenate:**
  - dacă buclele sunt independente unele de altele:
    - se aplică testarea **buclelor simple**;
  - dacă buclele sunt dependente (e.g., variabila de indexare a primei bucle este valoarea inițială a celei de a doua):
    - se aplică testarea **buclelor imbricate**.



**Concatenated**

# LC. Buclă nestructurată

- **bucle nestructurate:**
  - în general, indică folosirea instrucțiunii **goto**;
  - se recomandă 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

Întrebări

# Testare White-box

## Avantaje

- cazurile de testare sunt proiectate pe baza structurii interne a codului sursă, i.e., în funcție de structurile de programare folosite;
- identifică disfuncționalități în execuția anumitor secvențe de cod, e.g., unele structuri de programare nu sunt acoperite;

## Dezavantaje

- nu poate testa cerințe 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 funcțională, testare comportamentală (*engl. **behavioral testing***);
- cazurile de testare sunt proiectate pe baza specificațiilor, nu este necesar să avem acces la codul sursă;
- surprinde ambiguitățile sau inconsistențele din specificații;
- nu se există informații despre implementare;
- activitatea testerului este independentă de cea a programatorului; testerul poate proiecta cazurile de testare înainte de finalizarea codului sursă;
- eficientă și pentru module de mari dimensiuni.

## Testare White-box

- Testare structurală;
- cazurile de testare sunt proiectate pe baza structurii interne a codului sursă, i.e., în funcție de structurile de programare folosite;
- nu poate testa cerințe care nu sunt implementate;
- proiectarea cazurilor de testare poate începe doar după implementare;
- ineficientă pentru module de mari dimensiuni – construirea CFG și calculul CC sunt activități costisitoare.



# Testarea Black-box vs. Testarea White-box

## Întrebări:

- Ce cazuri de testare trebuie actualizate la modificarea specificațiilor și a codului sursă asociat?
- Ce cazuri de testare trebuie actualizate la modificarea codului sursă, fără modificarea specificațiilor?

# TESTARE BAZATĂ PE EXPERIENȚĂ

---

Definiție

Error guessing

Exploratory testing

# Testare bazată pe experiență. Definiție. Clasificare

- **testare bazată pe experiență** (*engl. experience-based testing*):
  - cazurile de testare se identifică pe baza abilităților, intuiției și experienței testerului;
- tehnici de testare asociate:
  - *engl. error guessing (EG)*;
    - **scop**: identificarea presupunerilor incorecte ale programatorului, cauzate de specificațiile incomplete;
  - *engl. exploratory testing (ET)*;
    - **scop**: identificarea defectelor când se utilizează specificații insuficiente, există o presiune severă a termenului de predare sau pentru a completa tehnicile de testare formale (EP, BVA).

# EG. Proiectarea cazurilor de testare. Reguli

1. se stabilește o listă cu:

- erori posibile;
- situații care generează erori;
- presupuneri pe care programatorul le face pe baza specificației;

2. se scriu cazuri de testare pe baza cazurilor speciale enumerate în listă;

- exemple:
  - un fișier care trebuie citit este gol sau nu există;
  - la o împărțire numitorul este 0;
  - fișierul începe cu linii fără informații;
  - vectorul este deja sortat.

PENTRU EXAMEN...

---

# Pentru examen...

- **testare white-box:**
  - definiție, caracteristici, avantaje și dezavantaje;
  - CFG (definiție și construire), drumuri independente (definiție), CC (definiție, 3 moduri de calcul);
  - construirea CFG, determinarea drumurilor independente și calculul CC (3 moduri) pentru metode concrete;
  - criteriile de acoperire **apc, sc, dc, cc, dcc, mcc** și **lc** (definiție, compararea a două criterii, relațiile existente între criterii);
  - testare black-box vs. testare white-box;
- **testare bazată pe experiență**
  - Error guessing;
  - Exploratory testing.

# Cursul următor...



- **Curs 04:**
  - Firmă invitată: **Evozon**
  - Tematică: **Testare automată**
  - Data: **Mărți, 20 Martie 2018**
  - Orele: **14:00-16:00**
  - Sala: **2/I (N.Iorga), Clădirea Centrală**

# 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.