# Esercizi su verifica

- Criteri strutturali di selezione dei test (CS)
- Analisi dataflow (AD)
- Criteri dataflow di selezione dei test (CD)

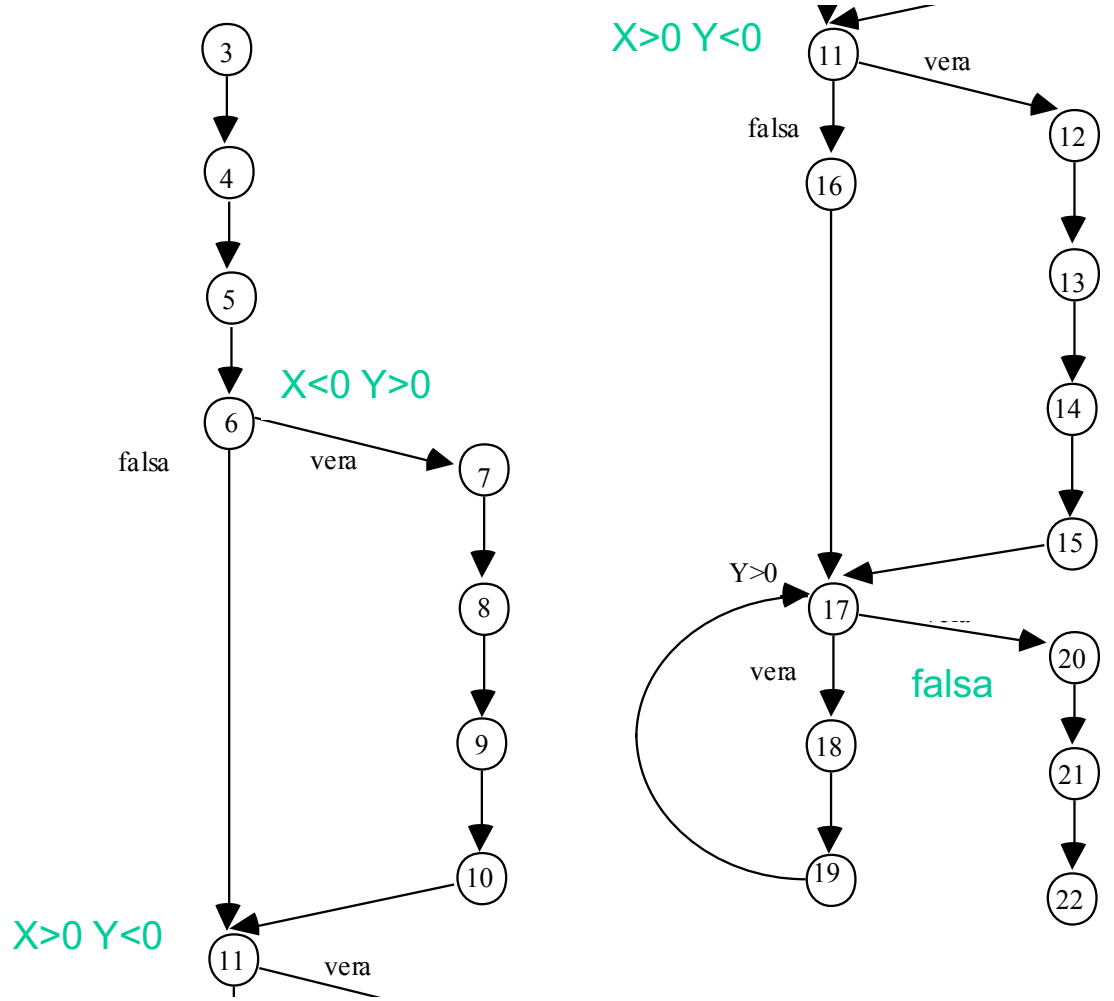
- Esecuzione simbolica (ES)

# Esercizio CS-1

- Si individui un insieme di casi di test di ***dimensione minima*** secondo il criterio di ***copertura dei comandi*** per il programma a fianco.
- Dire se e come andrebbe variato tale insieme di dati di test per soddisfare il criterio di copertura delle decisioni.

```
1.  program main;
2.  var X, Y, ABS_P: integer;
3.  begin
4.        read(X);
5.        read(Y);
6.        if X < 0 and Y >0
7.        then
8.              begin     X := -X;
9.                        ABS_P:=1
10.             end;
11.       if Y < 0 and X >0
12.       then
13.             begin     ABS_P:=1;
14.                       Y := -Y
15.             end
16.       else ABS_P:=1;
17.       while Y>0 do
18.             begin     ABS_P:=ABS_P+X;
19.                       Y := Y-1
20.             end;
21.       writeln(ABS_P);
22. end.
```

# Soluzione




- Per coprire tutti i nodi è sufficiente eseguire il programma con dati di test che soddisfano le condizioni:

  **Caso 1:** X<0 Y>0

  **Caso 2:** X>0 Y<0

- Il test sintetizzato può essere:

  T::={(X=-1;Y=2), (X=2; Y=-1)}

- I due casi coprono anche tutti gli archi.

# Esercizio AD-1

- Si determinino le espressioni regolari D-U per ciascuna variabile del programma a fianco. Cosa suggerisce tale risultato?

```
program ventisette;
var     A,B,C: Integer;
        X,Y: real;
begin
        Read(A);
        Read(B);
        if (B-A*C)>0
                then    begin
                        X:=-B+sqrt(B-A*C);
                        Y:=-B-sqrt(B-A*C);
                        end
        else X:=-Y;
        while A-B > 0
        do      begin
                A:= A-C;
                X:=X-1;
                Y:=Y-2;
                end
end.
```

# Soluzione

```
program ventisette;                                  A   B   C   X   Y
var     A,B,C: Integer;                              a   a   a
        X,Y: real;                                                a   a
begin
        Read(A);                                     d
        Read(B);                                         d
        if (B-A*C)>0                                 u   u   u
                then    begin
                        X:=-B+sqrt(B-A*C);           u   u   u   d
                        Y:=-B-sqrt(B-A*C);           u   u   u       d
                        end
        else X:=-Y;                                                   d   u
        while A-B > 0                                u   u
        do      begin
                A:= A-C;                             ud              u
                X:=X-1;                                                   ud
                Y:=Y-2;                                                       ud
                end
end.
```

# Espressioni regolari

Per **A**:

adu(uu+ε) u (udu)*

Per **B**:

adu(uu+ε) u (u)*

Per **C**:

**au**(uu+ε) (u)*

Per **X**:

a(d+d)(ud)*

Per **Y**:

**a**(d+**u**)(ud)*

Osservazioni:

Ci sono usi di C senza che questa variabile sia stata definita.

Ci possono essere usi di Y senza che questa variabile sia stata definita.

# Esercizio AD-2

- Si trovino le espressioni regolari associate alle variabili del seguente programma.
- Cosa suggerisce tale risultato?

```
program A
var X, Y, Z: Integer
begin
    Read(X);
    Read(Y);
    if X >Y
        then X:=X-1;
        else Y:=Y-1;
    fi
    while X+Y > 0
    do
        X:= Y-Z
        Z:= X+Y
        Y:= Y-X
    od
    if Z > 0
        then Z:= X+Y + Z
        else  Z:= X-Y -Z
    fi
end
```

# Soluzione

| program A | X | Y | Z |
|---|---|---|---|
| var X, Y, Z: Integer | a | a | a |
| **begin** | | | |
| Read(X); | d | | |
| Read(Y); | | d | |
| **if** X >Y | u | u | |
| **then** X:=X-1; | ud | ε | |
| **else** Y:=Y-1; | ε | ud | |
| **fi** | | | |
| **while** X+Y > 0 | u | u | |
| **do** | | | |
| X:= Y-Z | d | u | u |
| Z:= X+Y | u | u | d |
| Y:= Y-X | u | ud | |
| **od** | | | |
| **if** Z > 0 | ε | ε | u |
| **then** Z:=X+Y+Z | u | u | ud |
| **else** Z:= X-Y -Z | u | u | ud |
| **fi** | | | |
| **end** | | | |

# Soluzione

- X: adu (ud + ε) u ( duuu)* (u + u),
  semplificabile in
  adu (ud + ε) u ( duuu)* u
- Y: adu (ε + ud) u ( uuudu)* (u + u),
  semplificabile in
  adu (ε + ud) u ( uuudu)* u
- Z: a(ud)*u (ud + ud),
  semplificabile in
  a(ud)*u ud (genera una stringa con **au**)

# Esercizio AD-3

- Si calcolino le espressioni regolari associate alle variabili del programma a fianco.
- Si commentino i risultati ottenuti

```
program ese
var x,y,z : Integer;
read(x, y);
while x > 0 and y < x do
      x := x - 1;
      if x >= 1 then
            begin
                  y := y + x;
                  z  := y - x;
            end
      else
            begin
                  y := y - x;
                  z:= x + z;
            end
      endif
enddo
print(z);
end ese
```

# Soluzione

| | X | Y | Z |
|---|---|---|---|
| **program** ese | a | a | a |
| **var** x,y,z : Integer; | d | d | |
| read(x, y); | u | u | |
| **while** x > 0 and y < x **do** | ud | | |
| x := x - 1; | u | | |
| **if** x >= 1 **then** | | | |
| **begin** | | | |
| y := y + x; | u | ud | |
| z := y - x; | u | u | d |
| **end** | | | |
| **else** | | | |
| **begin** | | | |
| y := y - x; | u | ud | |
| z:= x + z; | u | | ud |
| **end** | | | |
| **endif** | | | |
| **enddo** | | | |
| print(z); | | | u |
| **end** ese | | | |

# Soluzione

```
program ese                          X     Y     Z
var x,y,z : Integer;                 a     a     a
read(x, y);                          d     d
while x > 0 and y < x do             u     u
    x := x - 1;                     ud
    if x >= 1 then                   u
        begin
        y := y + x;                  u    ud
        z  := y - x;                 u     u     d
        end
    else
        begin
        y := y - x;                  u    ud
        z:= x + z;                   u          ud
        end
    endif
enddo
print(z);                                       u
end ese
```

L'espressione regolare per X risulta:

adu(udu(uu+uu))u)*

# Soluzione

```
program ese                          X     Y     Z
var x,y,z : Integer;                 a     a     a
read(x, y);                          d     d
while x > 0 and y < x do             u     u
    x := x - 1;                     ud
    if x >= 1 then                   u
        begin
        y := y + x;                  u    ud
        z  := y - x;                 u     u     d
        end
    else
        begin
        y := y - x;                  u    ud
        z:= x + z;                   u          ud
        end
    endif
enddo
print(z);                                       u
end ese
```

L'espressione regolare per X risulta:

adu(udu(uu+uu))u)*

L'espressione regolare per Y risulta:

adu((udu+ud)u)*

# Soluzione

| program ese | X | Y | Z |
|---|---|---|---|
| var x,y,z : Integer; | a | a | a |
| read(x, y); | d | d | |
| while x > 0 and y < x do | u | u | |
| x := x - 1; | ud | | |
| if x >= 1 then | u | | |
| begin | | | |
| y := y + x; | u | ud | |
| z := y - x; | u | u | d |
| end | | | |
| else | | | |
| begin | | | |
| y := y - x; | u | ud | |
| z:= x + z; | u | | ud |
| end | | | |
| endif | | | |
| enddo | | | |
| print(z); | | | u |
| end ese | | | |

L'espressione regolare per X risulta:

adu(udu(uu+uu))u)*

L'espressione regolare per Y risulta:

adu((udu+ud)u)*

L'espressione regolare per Z risulta:

a(d + ud)*u

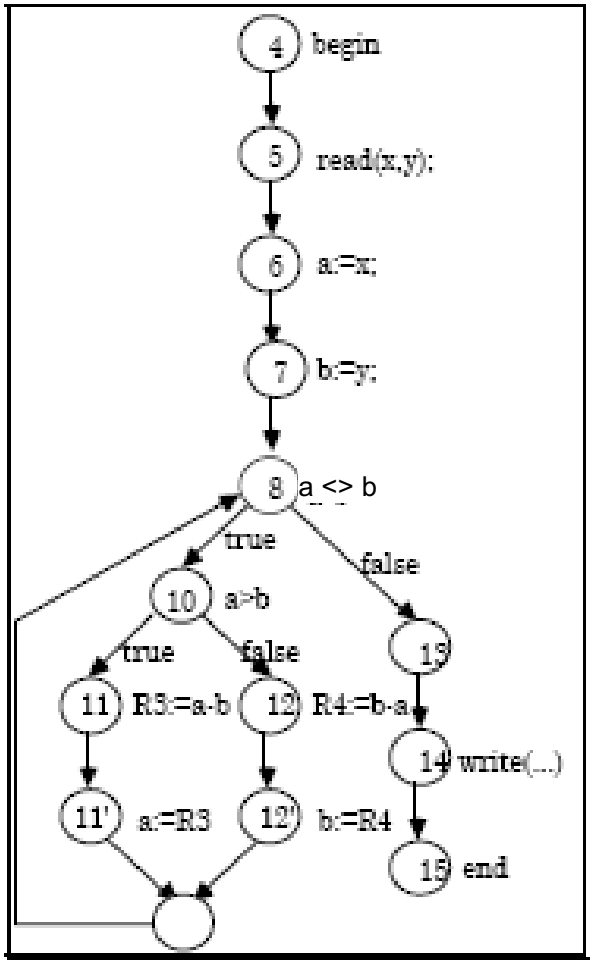
Ci sono possibili usi di Z senza che questa variabile sia stata definita.

# Criteri di selezione DataFlow

Definizione insieme *du(var,nodo)*:

- Per ogni nodo *i* e ogni variabile *x* appartenente all'insieme *def* associato al nodo *i*, si definisce *insieme du(x, i)* come l'insieme di tutti i nodi *j* tali che esiste un cammino libero da definizioni rispetto alla variabile *x* dal nodo *i* al nodo *j* e *x* è usata nel nodo *j.*

# Esempio



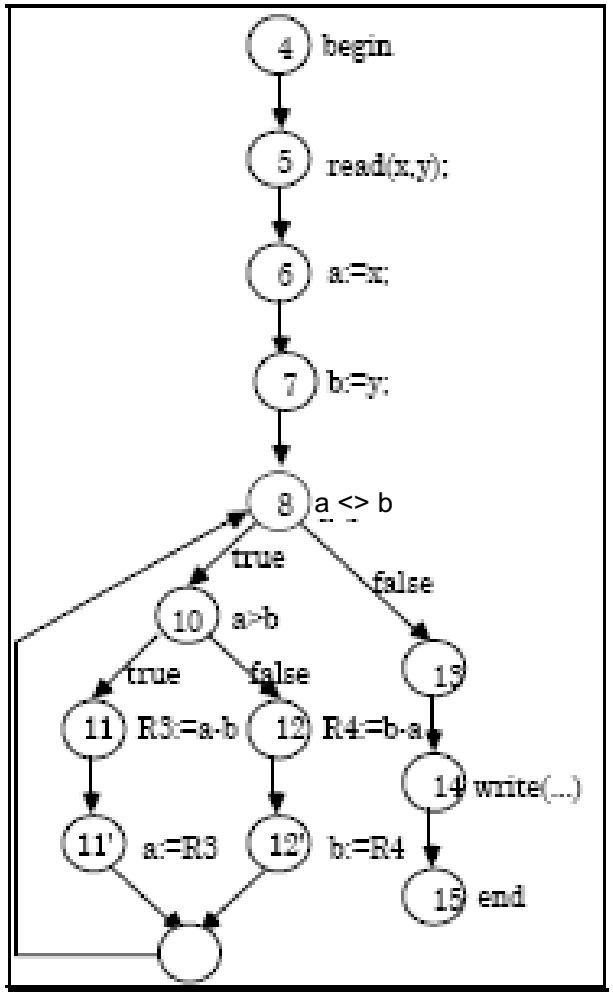

- Determinare gli insiemi: du(a,6), du(a,11'), du(x,5)

- du(a,6) = {8, 10, 11, 12, 14}
- du(a,11') = {8, 11, 12, 14}
- du(x,5) = {6}

# Criterio di copertura delle definizioni

- Soddisfatto da un test set T per un programma P se e solo se per ogni nodo *i* e per ogni variabile *x* appartenente all'insieme *def(i)* **esistono** un nodo u di *du(i, x)* ed un elemento d di T tali che l'esecuzione di P(d) percorre un cammino libero da definizioni rispetto a x dal nodo i al nodo u.

# Esempio

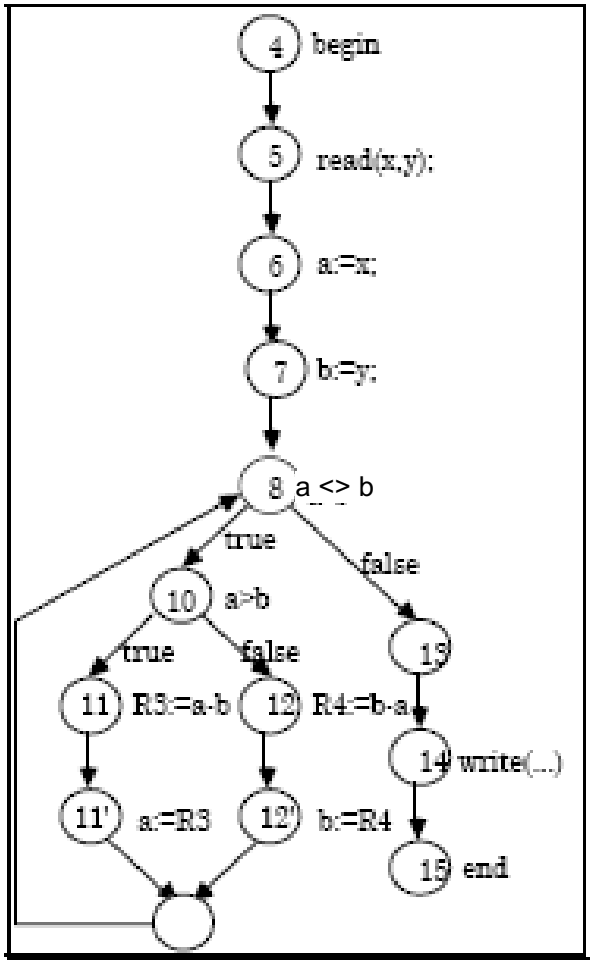



- Il criterio è soddisfatto da test set che comportano al massimo una iterazione del ciclo while.
- Es. non richiede che la def. di a in 11' sia seguita dall'uso in 11.
- Pertanto non rileva eventuali errori in questa sequenza

# Criterio di copertura di tutti gli usi

- Soddisfatto da un test set T per un programma P se e solo se per ogni nodo *i* del grafo di controllo di P, per ogni variabile *x* appartenente all'insieme *def(i)* e ***per ogni nodo u*** appartenente a *du(i,x)*, esiste un elemento d di T tale che l'esecuzione di P(d) percorre un cammino libero da definizioni rispetto a x dal nodo i al nodo u.

# Esempio




- Il criterio seleziona test set adeguati.
- Es: la def. di a in 11’ deve essere seguita dagli usi in 8,10, 11, 12, 14.

# Esercizio CD-1

- Dato il programma a fianco (in Pascal), si determini un ***insieme (minimo) di cammini*** da coprire per soddisfare il ***criterio di copertura di tutti gli usi***.

```
1  program undici;
2  var     A,B,C: Integer;
3          X,Y: real;
4  begin
5          Read(A);
6          Read(B);
7          if (B-A*C)>0
8                  then    begin
9                          X:=-B+sqrt(B-A*C);
10                         Y:=-B-sqrt(B-A*C);
11                         end
12         else X:=-Y;
13         while (A-B > 0) do
14                 begin
15                 A:= A-C;
16                 X:=X-1;
17                 Y:=Y-2;
18                 end
19 end.
```

# Soluzione

| | | def | use | du(A) | du(B) | du(C) | du(X) | du(Y) |
|---|---|---|---|---|---|---|---|---|
| 1 | **program** undici; | | | | | | | |
| 2 | **var** A,B,C: Integer; | | | | | | | |
| 3 | X,Y: real; | | | | | | | |
| 4 | **begin** | | | | | | | |
| 5 | Read(A); | A | | 7,9,10, 13,15 | | | | |
| 6 | Read(B); | B | | | 7,9,10,1 3 | | | |
| 7 | **if** (B-A*C)>0 | | A,B,C | | | | | |
| 8 | **then begin** | | | | | | | |
| 9 | X:=-B+sqrt(B-A*C); | X | A,B,C | | | | 16 | |
| 10 | Y:=-B-sqrt(B-A*C); | Y | A,B,C | | | | | 17 |
| 11 | **end** | | | | | | | |
| 12 | **else** X:=-Y; | X | Y | | | | 16 | |
| 13 | **while** A-B > 0 | | A,B | | | | | |
| 14 | **do begin** | | | | | | | |
| 15 | A:= A-C; | A | A,C | 13, 15 | | | | |
| 16 | X:=X-1; | X | X | | | | 16 | |
| 17 | Y:=Y-2; | Y | Y | | | | | 17 |
| 18 | **end** | | | | | | | |
| 19 | **end**. | | | | | | | |

# Soluzione

I cammini da coprire sono quindi:

- 5-6-7-8-9-10-11-13-14-15 e 15-16-17-18-13-14-15 per la var. A
- 6-7-8-9-10-11-13 per la var. B
- 9-10-11-13-14-15-16, 12-13-14-15-16 e 16-17-18-13-14-15-16 per la var. X
- 10-11-13-14-15-16-17 e 17-18-13-14-15-16-17 per la var. Y

# Esercizio CD-2

- Dato il programma a fianco, determinare un insieme minimo di cammini da coprire per rispettare il criterio di copertura di tutti gli usi.

```
1   program primo;
2   var A,B: Integer;
3       X: real;
4       T: boolean;
5   begin
6       T:=false;
7       Read(A);
8       Read(B);
9       if A>B
10          then    begin
11              X:=(A-B)/A;
12              T:=true;
13              end
14      else X:=(B-A)/B;
15      while T
16      do  begin
17          A:= A-B;
18          if A<=0
19          then T:=false;
20          end
21  end.
```

# Soluzione

| Nodo | | def | use | du(A) | du(B) | du(T) | du(X) |
|---|---|---|---|---|---|---|---|
| 1 | program primo; | | | | | | |
| 2 | var A,B: Integer; | | | | | | |
| 3 | X: real; | | | | | | |
| 4 | T: boolean; | | | | | | |
| 5 | begin | | | | | | |
| 6 | T:=false; | T | | | | 15 | |
| 7 | Read(A); | A | | 9, 11, 14, 17 | | | |
| 8 | Read(B); | B | | | 9, 11, 14, 17 | | |
| 9 | if A>B | | A, B | | | | |
| 10 | then begin | | | | | | |
| 11 | X:=(A-B)/A; | X | A,B | | | | |
| 12 | T:=true; | T | | | | 15 | |
| 13 | end | | | | | | |
| 14 | else X:=(B-A)/B; | X | A, B | | | | |
| 15 | while T | | T | | | | |
| 16 | do begin | | | | | | |
| 17 | R1:= A-B; | | A, B | | | | |
| 17' | A:= R1; | A | | 18, 17 | | | |
| 18 | if A<=0 | | A | | | | |
| 19 | then T:=false; | T | | | | 15 | |
| 20 | end | | | | | | |
| 21 | end. | | | | | | |

# Soluzione

- Occorre coprire i cammini che comprendono percorsi da:
  - 6 a 15
  - 7 a ciascun nodo tra 9, 11, 14, 17
  - 8 a ciascun nodo tra 9, 11, 14, 17
  - 12 a 15
  - 17’ a ciascun nodo tra 18, 17
  - 19 a 15
- Insieme minimo di cammini:
  - 1, …6,7,8, 9, 10, 11 (then), 12, 15, 16, 17, 17’, 18 (then), 19, 20, 15 (seconda it. ciclo while), 17, 17’, 18 (else), 21
  - 1, …6,7,8, 9, 10, 14 (else), 21

# Esercizio CD-3

- Dato il programma a fianco, determinare un insieme minimo di cammini da coprire per rispettare il criterio di copertura di tutti gli usi.

```
1   program diciotto;
2   var      A,B,C: Integer;
3        X,Y: real;
4   begin
5        Read(A);
6        Read(B);
7        if (B-A*C)>0
8             then     begin
9                  X:=-B+sqrt(B-A*C);
10                 Y:=-B-sqrt(B-A*C);
11                 end
12       else X:=-Y;
13       while A-B > 0
14       do  begin
15            A:= A-C;
16            X:=X-1;
17            Y:=Y-2;
18            end
19  end.
```

# Soluzione

| | | def | use | du(A) | du(B) | du(X) | du(Y) |
|---|---|---|---|---|---|---|---|
| 1 | **program** diciotto; | | | | | | |
| 2 | **var** A,B,C: Integer; | | | | | | |
| 3 | X,Y: real; | | | | | | |
| 4 | **begin** | | | | | | |
| 5 | Read(A); | A | | 7,9,10 ,13,15 | | | |
| 6 | Read(B); | B | | | 7,9,1 0,13 | | |
| 7 | **if** (B-A*C)>0 | | A,B,C | | | | |
| 8 | **then begin** | | | | | | |
| 9 | X:=-B+sqrt(B-A*C); | X | A,B,C | | | 16 | |
| 10 | Y:=-B-sqrt(B-A*C); | Y | A,B,C | | | | 17 |
| 11 | **end** | | | | | | |
| 12 | **else** X:=-Y; | X | Y | | | 16 | |
| 13 | **while** A-B > 0 | | A,B | | | | |
| 14 | **do begin** | | | | | | |
| 15 | A:= A-C; | A | A,C | 13,15 | | | |
| 16 | X:=X-1; | X | X | | | 16 | |
| 17 | Y:=Y-2; | Y | Y | | | | 17 |
| 18 | **end** | | | | | | |
| 19 | **end**. | | | | | | |

# Cammini

- To do …

# Esercizio ES-1

Si descriva il risultato di una *esecuzione simbolica* della seguente procedura (in linguaggio Pascal) indicando i valori dello *stato* del programma e della *path condition* dopo ogni istruzione nell'ipotesi di voler eseguire ogni comando del programma (ramo *then*) ed il ciclo *while* una volta.

```
1.  function FATT (X: INTEGER):INTEGER
2.  RIS: INTEGER;
3.  POS_X, PARI_X: BOOL;
4.  begin
5.      POS_X := TRUE;
6.      PARI_X := TRUE;
7.      if X < 0 then
8.          if X mod 2 <> 0 then
9.              POS_X := FALSE;
10.             PARI_X := FALSE;
11.         end if
12.         X := -X;
13.     end if
14.     RIS := 1;
15.     while X > 0 loop
16.         RIS := RIS *X;
17.         X := X-1;
18.     end loop
19.     if not POS_X and not PARI_X
20.     then RIS := -RIS end if
21.     RETURN(RIS);
22. end --FATT
```

# Soluzione

| **Esecuzione** | **Stato simbolico** |
|---|---|
| inizio | X=valx, PC=true, FATT,RIS,POS_X,PARI_X=undef |
| 5,6 | X=valx, PC=true, POS_X=true, PARI_X=true, FATT,RIS=undef |
| 7 (scelta: vero) | X=valx, PC=valx<0, POS_X=true, PARI_X=true, FATT,RIS=undef |
| 8 (scelta: vero) | X=valx, PC=(valx<0) and (valx mod 2 <> 0), POS_X=true, PARI_X=true, FATT,RIS=undef |
| 9,10 | X=valx, PC=(valx<0)and(valx mod 2<>0), POS_X=false, PARI_X=false, FATT,RIS=undef |
| 11,12 | X=-valx,PC=(valx<0)and(valx mod 2<>0), POS_X=false, PARI_X=false, FATT,RIS=undef |
| 13,14 | X=-valx,PC=(valx<0)and(valx mod 2<>0), POS_X=false, PARI_X=false, RIS=1, FATT=undef |

# Soluzione

| Esecuzione | Stato simbolico |
|---|---|
| 15 (vero) | X=-valx,PC=(valx<0)and(valx mod 2<>0), POS_X=false, PARI_X=false, RIS=1, FATT=undef |
| 16,17 | X=-valx-1, PC=(valx<0)and(valx mod 2<>0), POS_X=false, PARI_X=false, RIS= - valx, FATT=undef |
| 15 (falso) | X= -valx-1, PC=(valx<0)and(valx mod 2<>0) and(\|valx\|<=1), POS_X=false, PARI_X=false, RIS=-valx, FATT=undef |
| 19 | X= -valx-1, PC=(valx<0) and (valx mod 2<>0) and (\|valx\|<=1), POS_X=false, PARI_X=false, RIS= +valx, FATT=undef |
| 21 | X= -valx-1, PC=(valx<0) and (valx mod 2<>0) and (\|valx\|<=1), POS_X=false, PARI_X=false, RIS= +valx, FATT= valx |

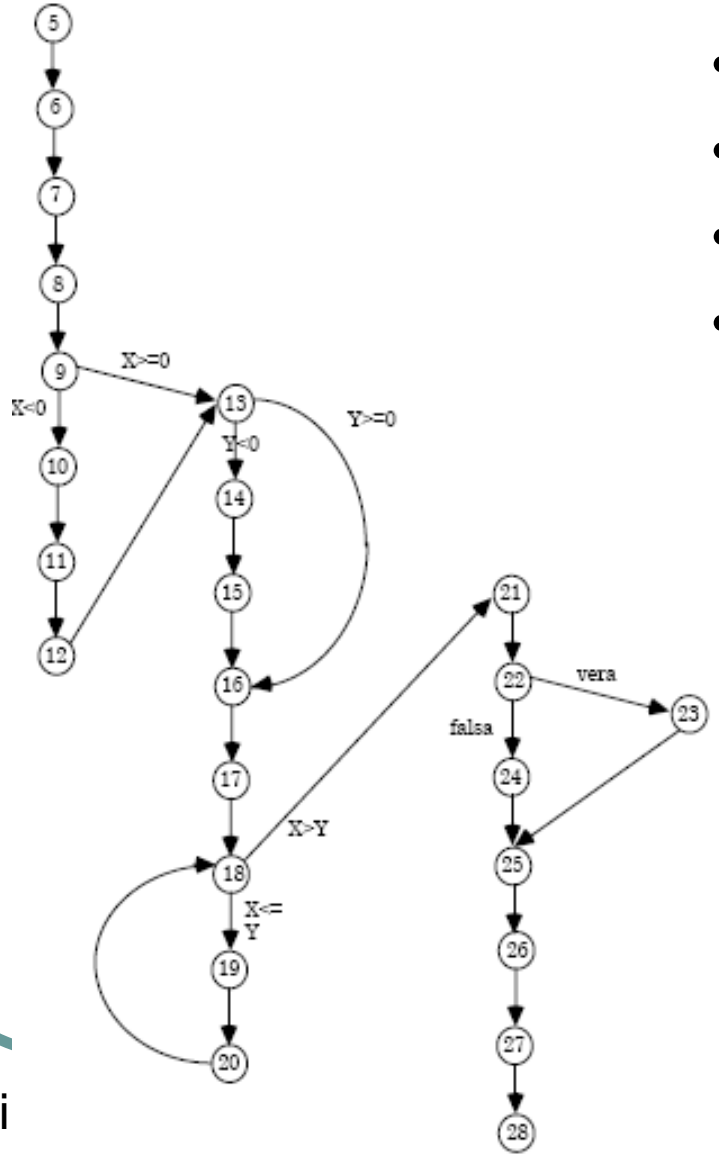
La path condition è
PC=(valx<0) and (valx mod 2 <> 0) and (|valx|<=1)
Un dato di test che la soddisfa è x=-1

# Esercizio ES-2

```
1.  procedure DIVIDI (X, Y : INTEGER)
2.  RIS, RESTO : INTEGER
3.  POS_X, POS_Y : BOOL
4.  begin
5.  POS_X := TRUE;
6.  POS_Y := TRUE;
7.  READ(X);
8.  READ(Y);
9.  if X < 0 then
10.     POS_X := FALSE;
11.     X := -X;
12. end if
13. if Y < 0 then
14.     POS_Y := FALSE;
15.     Y := -Y;
16. end if
17. RIS := 0;
18. while X <= Y loop
19.     X := X - Y;
20.     RIS := RIS - 1;
21. end loop
22. if not (POS_X or POS_Y)
23. then RESTO := -X
24. else RESTO = X
25. end if
26. WRITE(RIS);
27. WRITE(RESTO);
28. end --DIVIDI
```

- Disegnare il grafo di controllo del programma
- Individuare le condizioni per eseguire il cammino: 1-21, 18, 22, 23, 25-28

# Soluzione




- X<0
- Y<0
- X<=Y
- (X-Y) > Y
  - *Inconsistenza*

# Esercizio ES-3

Si esegua simbolicamente la seguente procedura, nell'ipotesi di voler eseguire ogni comando del programma (ramo then) ed il ciclo while una volta, mostrando come varia la path condition e il valore delle variabili. Cosa calcola la funzione A ?

```
    function A ( N: Integer): Integer
    var X, Y: Integer
    begin
1.      X:=N;
2.      if N < 0
3.      then
4.          while N < 0 do
5.              X:= X+2;
6.              N:= N+1;
7.          endwhile
8.      endif
9.      Y:=X;
10.     return(Y);
    end
```

# Soluzione

| Esecuzione | Stato simbolico |
|---|---|
| Inizio | N=valn, PC=true, X,Y,A=undef |
| 1 | N=valn, PC=true, X= valn, Y,A=undef |
| 2 (scelta: true) | N=valn, PC=valn<0, X= valn, Y,A=undef |
| 4 (vero) | N=valn, PC=valn<0, X= valn, Y,A=undef |
| 5, 6 | N=valn+1, PC=valn<0, X= valn+2, Y,A=undef |
| 4 (scelta: falso) | N=valn+1, PC=(valn<0) and (valn+1>=0), X= valn+2, Y,A=undef |
| 9, 10 | N=valn+1, PC=(valn<0) and (valn+1>=0), X= valn+2, Y,A=valn+2 |

# Soluzione

- La path condition è

  PC=(valn<0) and (valn>=-1)
- L'unico test che soddisfa PC è

  T = {(N=-1)}
- La funzione calcolata è

  valn + 2

# Domande (un esempio …)

- Per individuare all'interno di un programma l'uso di una variabile non inizializzata non è necessario eseguire il programma. Vero o falso? (motivare)

# Domande (un esempio …)

- Per individuare all'interno di un programma l'uso di una variabile non inizializzata non è necessario eseguire il programma. Vero o falso? (motivare)

**Soluzione:**

- Vero, è sufficiente eseguire un'analisi statica del flusso delle variabili (analisi D-U-A).