DOKAZOVANJE PRAVILNOSTI PROGRAMOV

Pravilnost programa lahko preverjamo :

- z branjem programa in intuitivnim razumevanjem, kaj program dela,
- s testiranjem programa na izbrani množici testnih podatkov,
- s formalnim dokazom pravilnosti programa

Zakaj formalno dokazovanje pravilnosti?

- zahteva natančno analizo programa in formalen opis pomembnih relacij med podatkovnimi objekti med izvajanjem programa,

- dokazi pravilnosti programa so tipično daljši od samega programa,

- formalno zahtevna metoda,

+ omogoča boljše razumevanje, kaj je pravilnost programa,

+ formalen dokaz enkrat za vselej pokaže pravilnost algoritma,

+ pri kritičnih delih kode s formalnim dokazom pravilnosti zagotovimo zanesljivost programske opreme.

Dokazovanje pravilnosti preprostih programov

- zaporedje stavkov,
- prireditveni stavek,
- izbira **if-else**,
- zanka **while**

Formalizacija:

Program definiramo kot preslikavo:

$$f : X \longrightarrow Z$$

ki preslika vhodne podatke $< x_1, \ldots, x_n > \in X$ v izhodne podatke $< z_1, \ldots, z_m > \in Z$. Pri tem morajo vhodni podatki izpolnjevati *začetni pogoj*:

$$\phi(x_1, \ldots, x_n)$$

Izhodni podatki pa morajo izpolnjevati zaključni pogoj:

$$\psi(z_1, \ldots, z_m, x_1, \ldots, x_n)$$

Pravimo, da je program:

parcialno pravilen, če v primeru, da se za vhodne podatke, ki izpolnjujejo začetni pogoj ϕ, ustavi, izhodni podatki izpolnjujejo zaključni pogoj ψ;

totalno pravilen, če je parcialno pravilen, in če se za vse vhodne podatke, ki izpolnjujejo začetni pogoj ϕ, po končnem številu korakov ustavi.

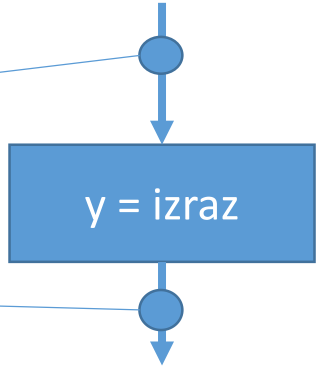
Pri dokazovanju pravilnosti programa iz pogojev P, ki veljajo pred izvrševanjem stavka, izpeljujemo pogoje Q, ki veljajo po izvršitvi stavka:

// $P(Y)$
Stavek;
// $Q(Y)$

Pravila izpeljevanja pogojev

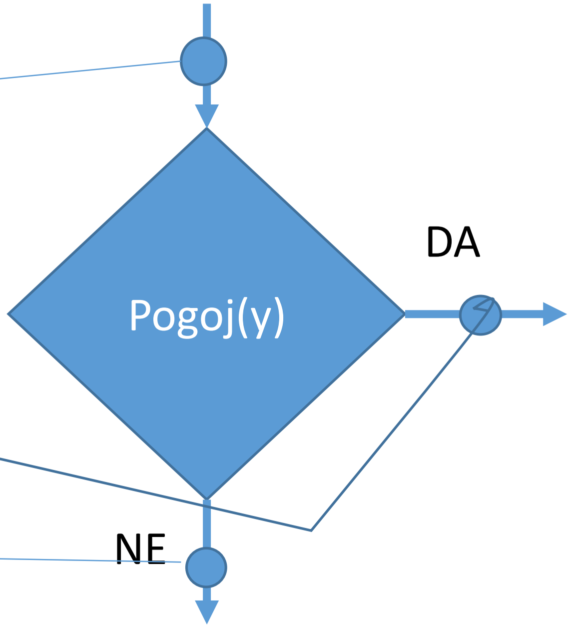
Prireditev:

// P(izraz)
y = izraz;
// P(y)



Izbira:

//P(y)
if (Pogoj(y))
// P(y) & Pogoj(y)
...;
else
// P(y) & !Pogoj(y)
...;



Pravila izpeljevanja pogojev

Zaporedje:

// P0(y)

{ S1; S2; ...; Sk }

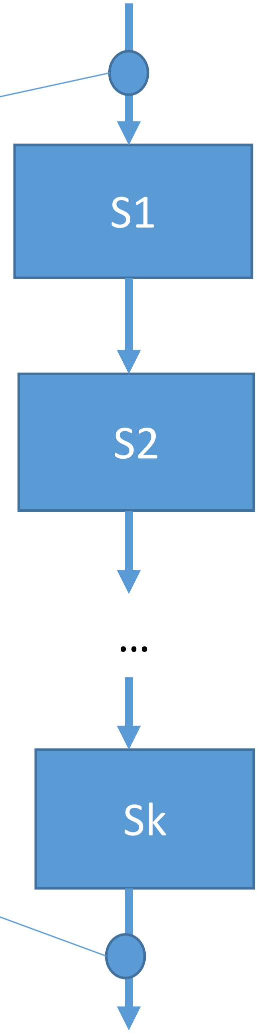
// Pk(y)

pri čemer velja

// P_(i−1)(y)

Si;

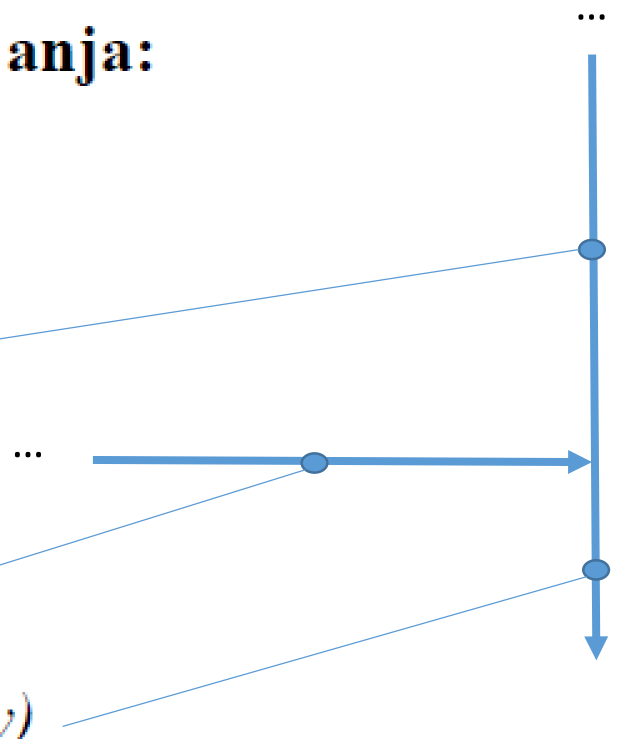
// Pi(y)



Pravila izpeljevanja pogojev

Združitev več poti izvajanja:

if (...)
{ ... }
// P1(y)
else
{...}
// P2(y)
; *// P1(y) or P2(y)*



Pravila izpeljevanja pogojev

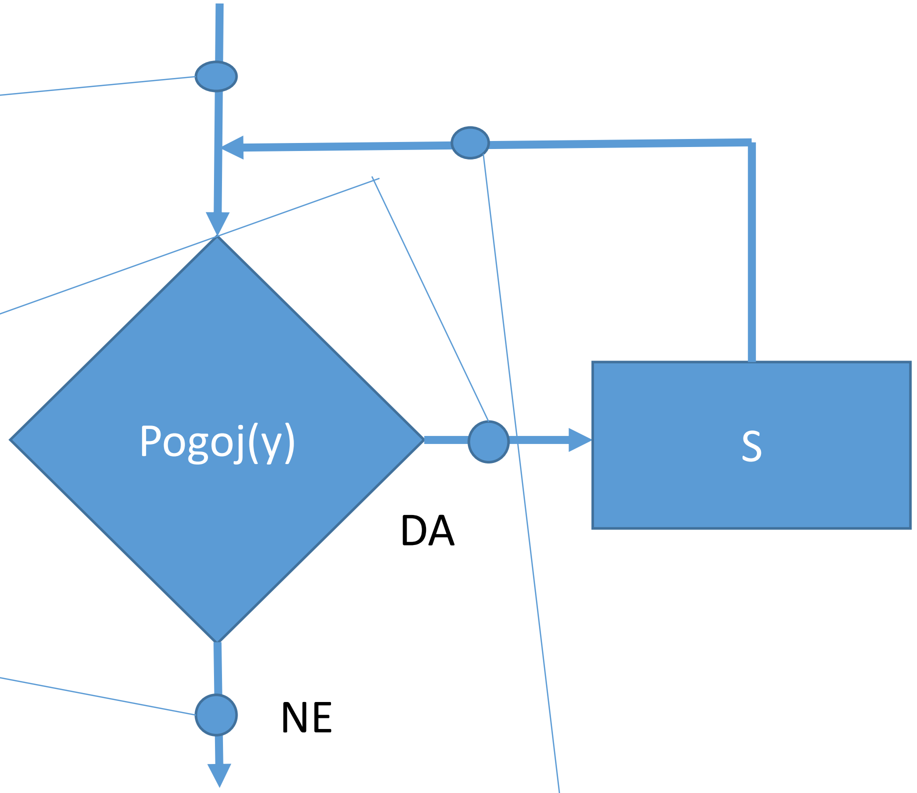
Zanka:

// P1(y)
while (Pogoj(y)) {
// za i–to izvajanje zanke: Pi(y) & Pogoj(y)
S;
} *// while*
// Pk(y) & !Pogoj(y)

Pri čemer velja

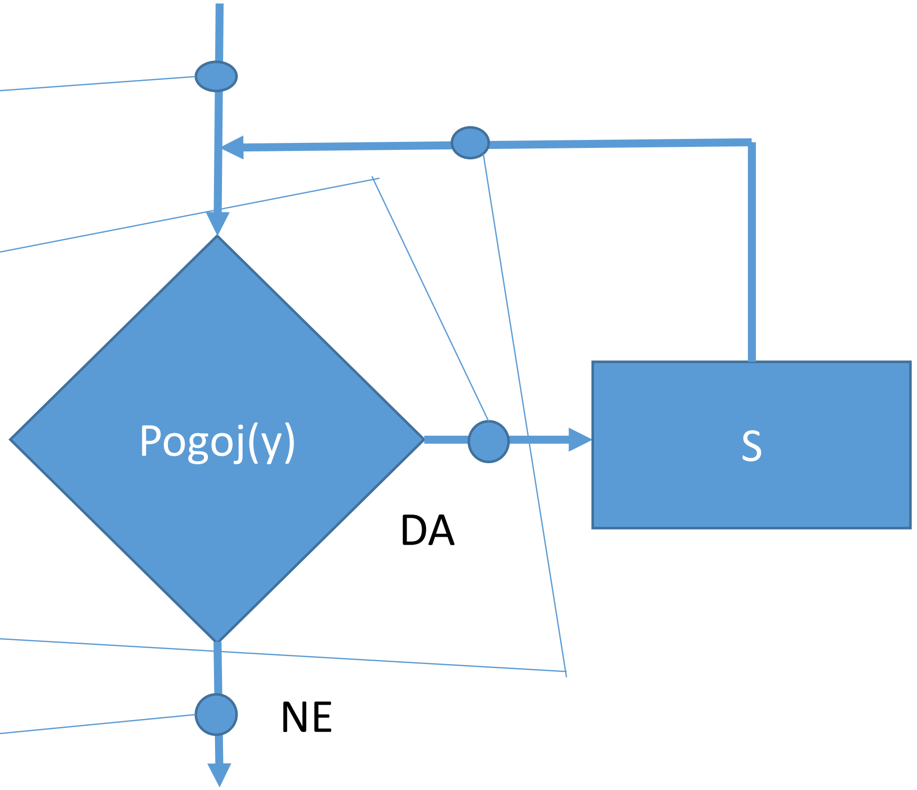
// Pi(y) & Pogoj(y)
S
// P_(i+1)(y)

in je pogoj $P_k(y)$ odvisen od števila izvajanj zanke.



Zančna invarianta: *I(y)*

```
// I(y)
while (Pogoj(y)) {
  // I(y) & Pogoj(y)
    S;
  // I(y)
  } // while
  // I(y) & !Pogoj(y)
```



Parcialna pravilnost programa:

1. Pri zankah definiramo dovolj močne zančne invariante
2. Uporabljamo pravila izpeljevanja pogojev
3. Začnemo pri začetku programa (α) z začetnim pogojem ϕ
4. Izpeljujemo vse do zaključka programa (ω), kjer mora biti izpolnjen zaključni pogoj ψ

Parcialna pravilnost programa:

1. Pri zankah definiramo dovolj močne zančne invariante
2. Uporabljamo pravila izpeljevanja pogojev
3. Začnemo pri začetku programa (α) z začetnim pogojem ϕ
4. Izpeljujemo vse do zaključka programa (ω), kjer mora biti izpolnjen zaključni pogoj ψ
5. Če iz začetnega pogoja izpeljemo zaključni pogoj, je program **parcialno pravilen:**

V primeru, da se program za vhodne podatke, ki izpolnjujejo začetni pogoj φ, ustavi, izhodni podatki izpolnjujejo zaključni pogoj ψ.

Parcialna pravilnost programa:

1. Pri zankah definiramo dovolj močne zančne invariante
2. Uporabljamo pravila izpeljevanja pogojev
3. Začnemo pri začetku programa (α) z začetnim pogojem ϕ
4. Izpeljujemo vse do zaključka programa (ω), kjer mora biti izpolnjen zaključni pogoj ψ
5. Če iz začetnega pogoja izpeljemo zaključni pogoj, je program **parcialno pravilen:**

V primeru, da se program za vhodne podatke, ki izpolnjujejo začetni pogoj φ, ustavi, izhodni podatki izpolnjujejo zaključni pogoj ψ.

Ustvarjalni del dokaza so zančne invariante, ostalo je rutinsko delo, ki ga lahko opravi računalnik (specializirani programski jeziki).

Totalna pravilnost programa

- Program je totalno pravilen, če
  - je parcialno pravilen
  - se za vse vhodne podatke, ki izpolnjujejo začetni pogoj ϕ, po končnem številu korakov ustavi.
- Pri preverjanju ustavljivosti programa so problematične samo zanke.

Za vsako zanko potrebujemo:

- zančno spremenljivko l,
- *dobro utemeljeno množico* (delno urejeno množico brez neskončnih padajočih zaporedij) D (ponavadi vzamemo kar množico naravnih števil),
- zančno invarianto $l \in D$,

1. Treba je dokazati resničnost zančne invariante.
2. Treba je dokazati, da se zančna spremenljivka *l* po vsaki izvršitvi zanke zmanjša: *ll* < *l*.
3. Ker vrednost *l* ne more v nedogled padati, se bo zanka po končnem številu korakov iztekla.

```
// l pripada D
while (Pogoj(y)) {
   // l pripada D & Pogoj(y)
   S;
   // ll pripada D & (ll < l)
} // while
```

Primer: Izračun faktoriele

- Začetni pogoj: $\phi(n) = (n \in \mathrm{N} \cup \{0\})$

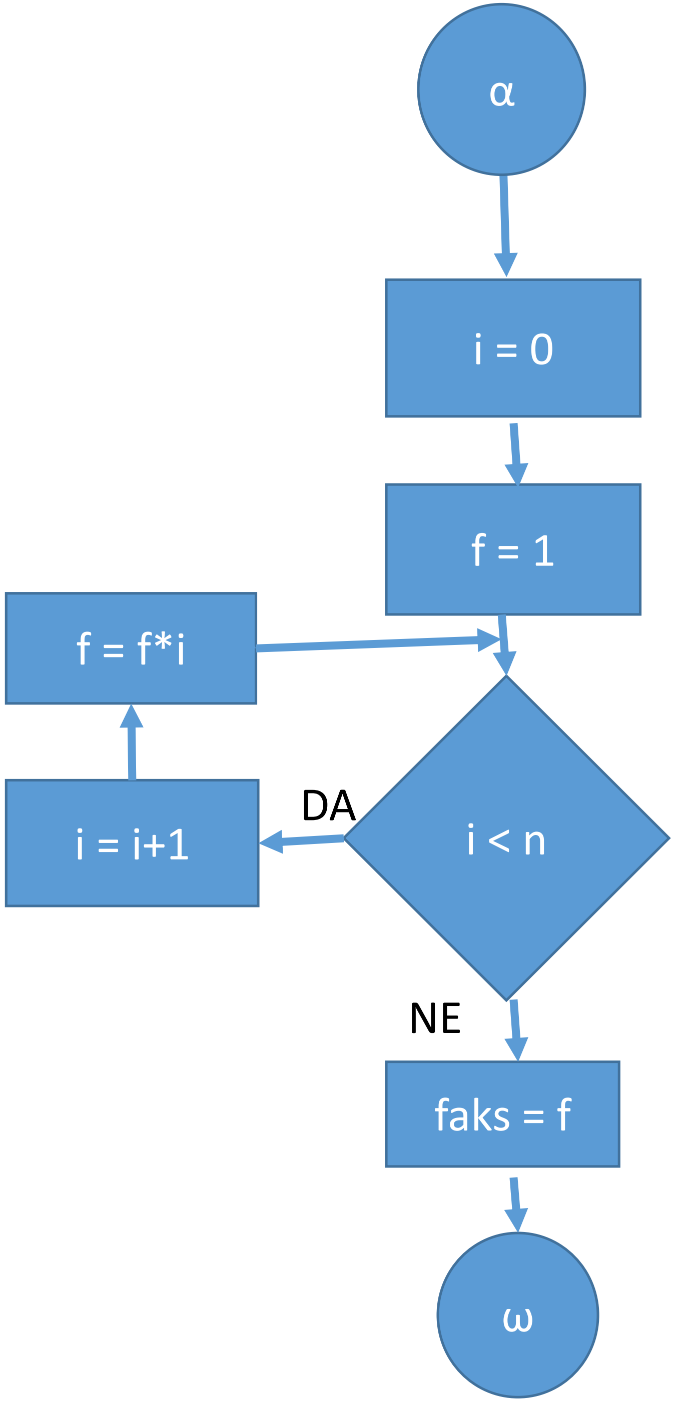
- Zaključni pogoj: $\psi(faks, n) = (faks = n!)$

Primer: Izračun faktoriele

```
static public int faks(int n) {
// fi(n) = (n >= 0)
    int i=0,f=1 ;

    while (i < n) {
        i++ ;
        f *= i ;
    } // while
    return f ;
    // psi(faks, n) = (faks = n!)
} // faks
```

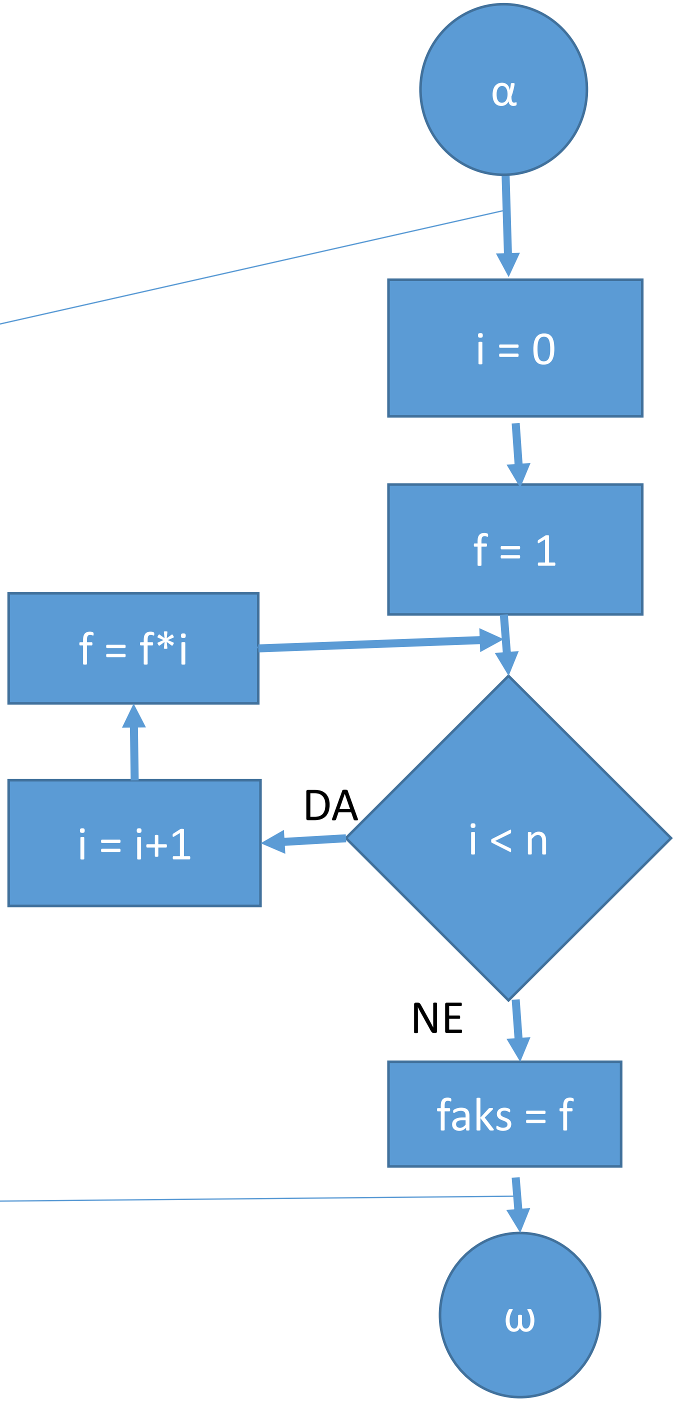
Primer: Izračun faktoriele

```
static public int faks(int n) {
// fi(n) = (n >= 0)
    int i=0,f=1 ;

    while (i < n) {
      i++ ;
      f *= i ;
    } // while
    return f ;
    // psi(faks, n) = (faks = n!)
} // faks
```

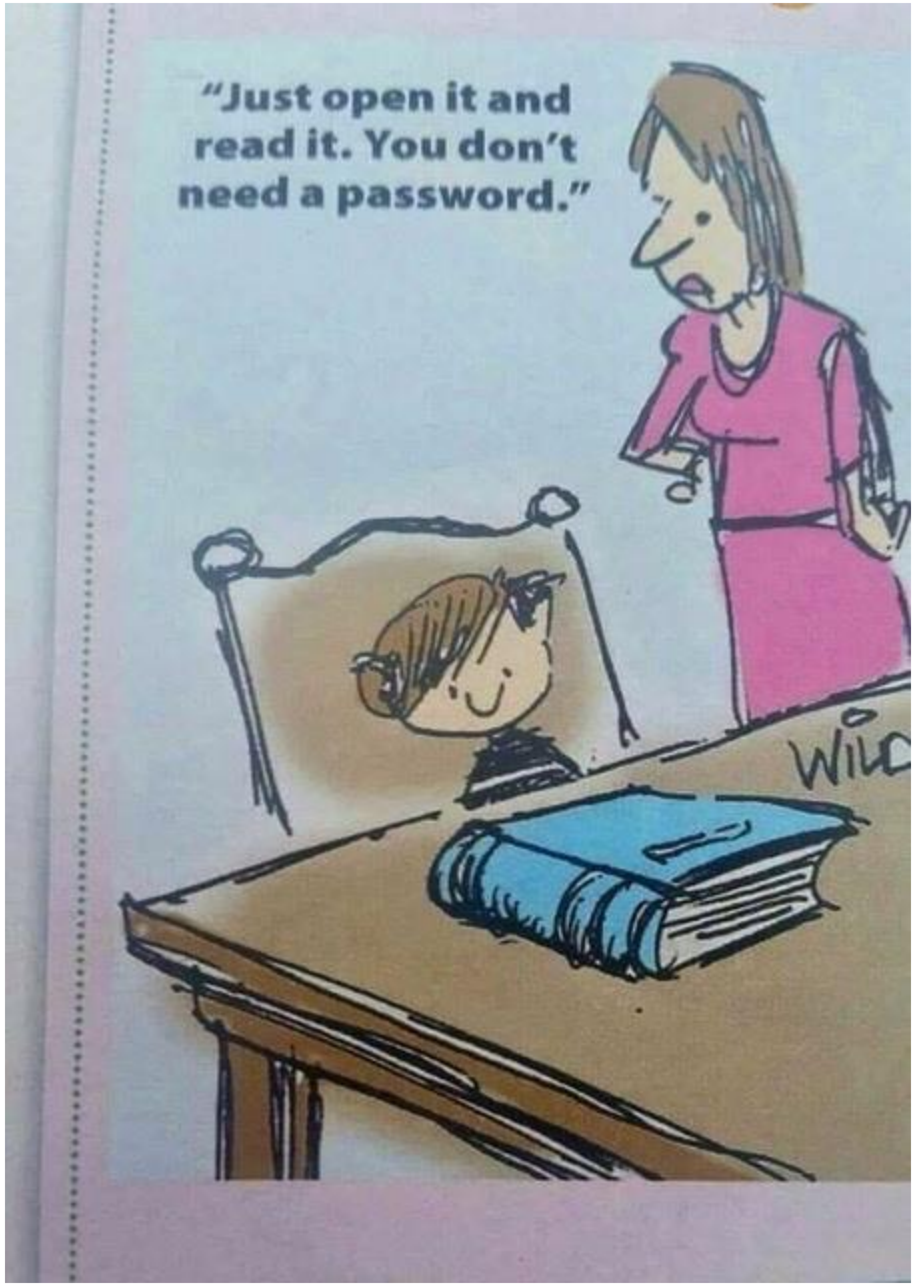


Primer: Izračun faktoriele

```
static public int faks(int n) {
// fi(n) = (n >= 0)
    int i=0,f=1 ;

    while (i < n) {
      i++ ;
      f *= i ;
    } // while
    return f ;
    // psi(faks, n) = (faks = n!)
} // faks
```

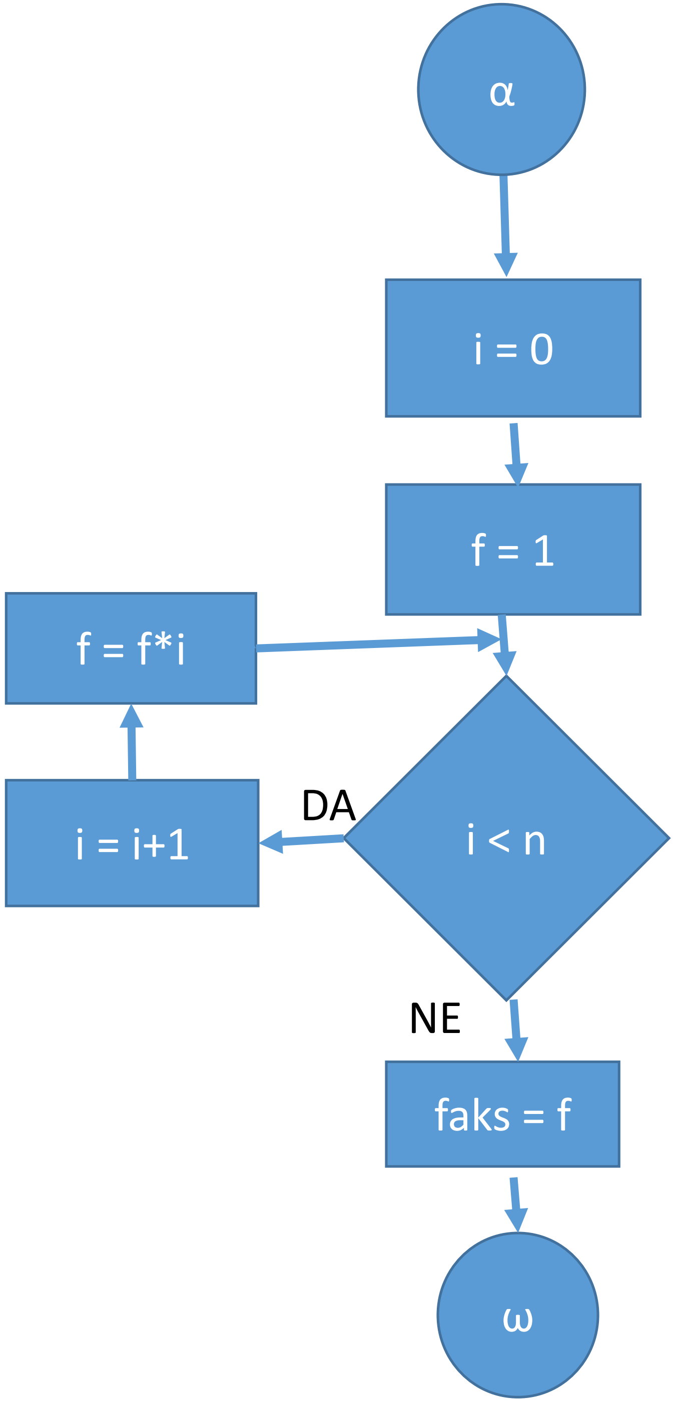



IT'S CALLED 'A BOOK'.
...NOT SURE WHERE THE BATTERIES GO.
GABLE GLOBE+MAIL


"Just open it and read it. You don't need a password."

Primer: Izračun faktoriele

Kaj je primerna zančna invarianta?

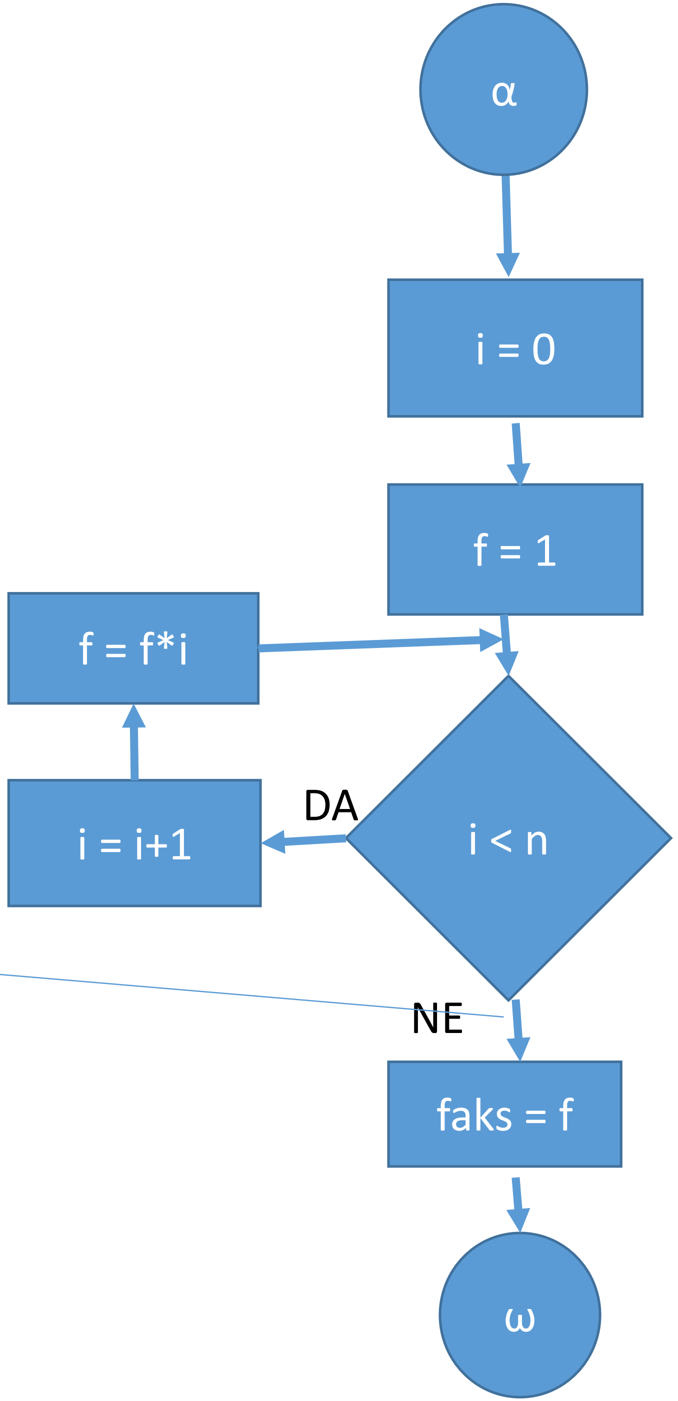


Primer: Izračun faktoriele

Kaj je primerna zančna invarianta?

Dokazati moramo, da je *faks = n!*

Torej pred stavkom *faks = f* mora veljati *f = n!*



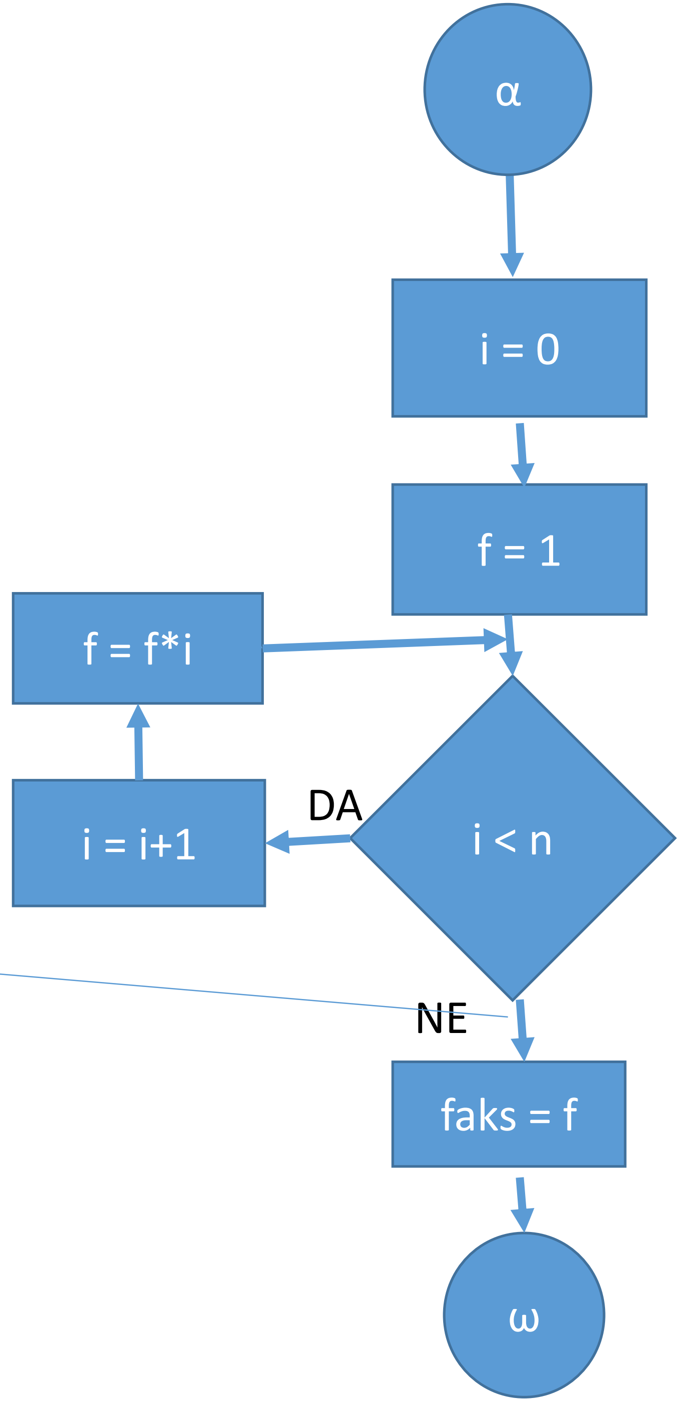
Primer: Izračun faktoriele

Kaj je primerna zančna invarianta?

Dokazati moramo, da je *faks = n!*

Torej pred stavkom *faks = f* mora veljati *f = n!*

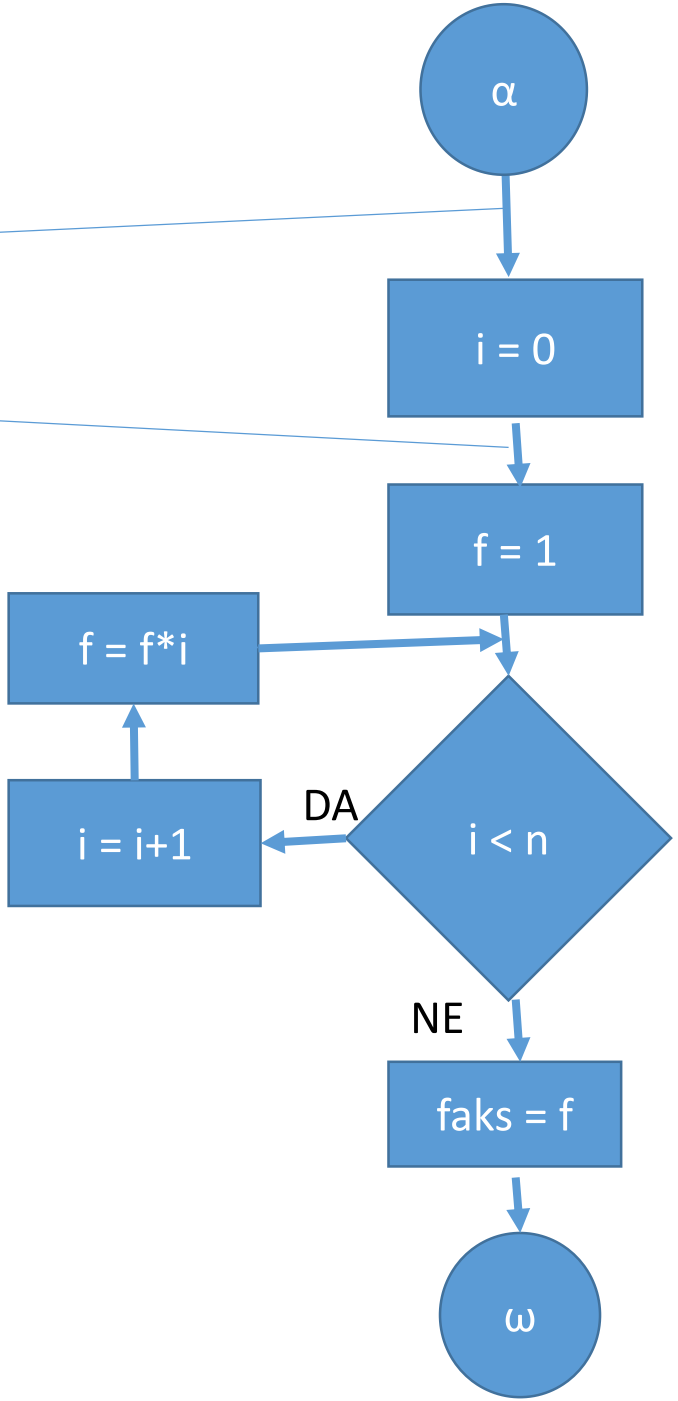
Ker v tej točki ne velja več *i < n*, mora torej veljati *i = n*. Torej je primerna zančna invarianta lahko ***f = i!***



Primer: Izračun faktoriele

n >= 0

i = 0

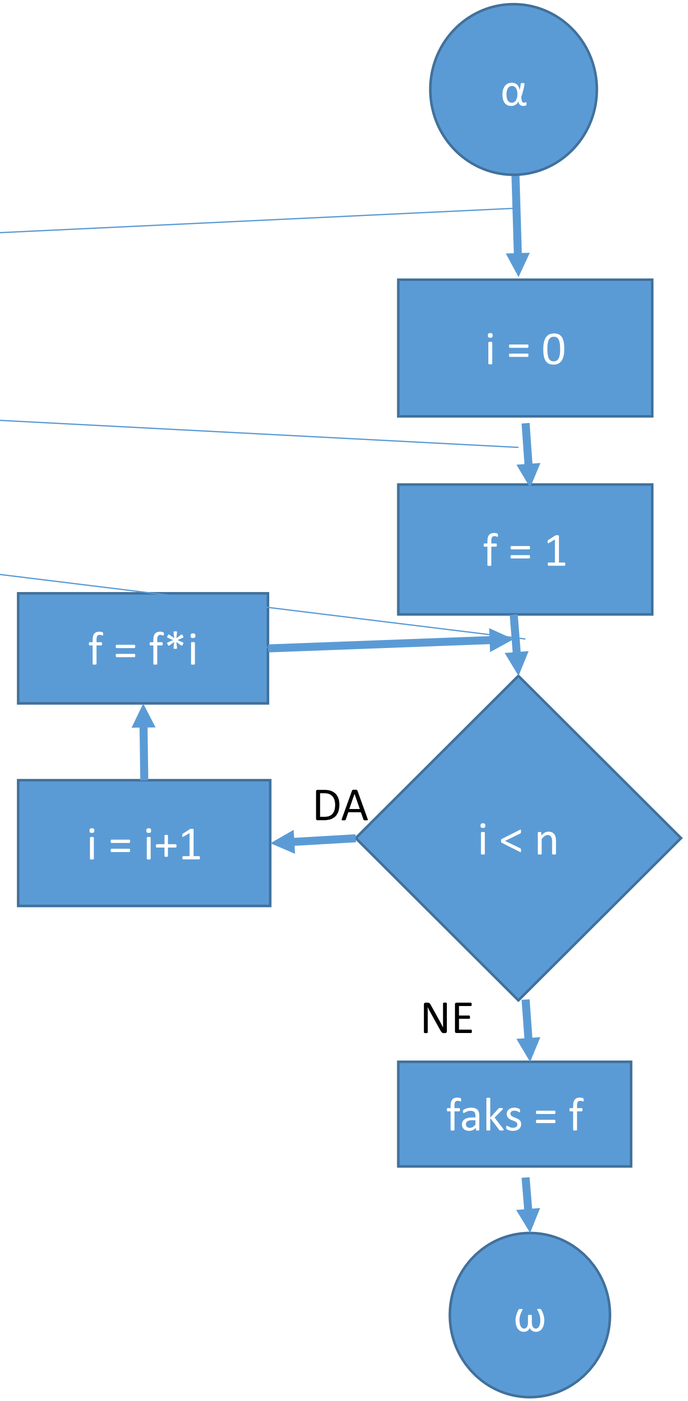


Primer: Izračun faktoriele

n >= 0

i = 0

f = 1



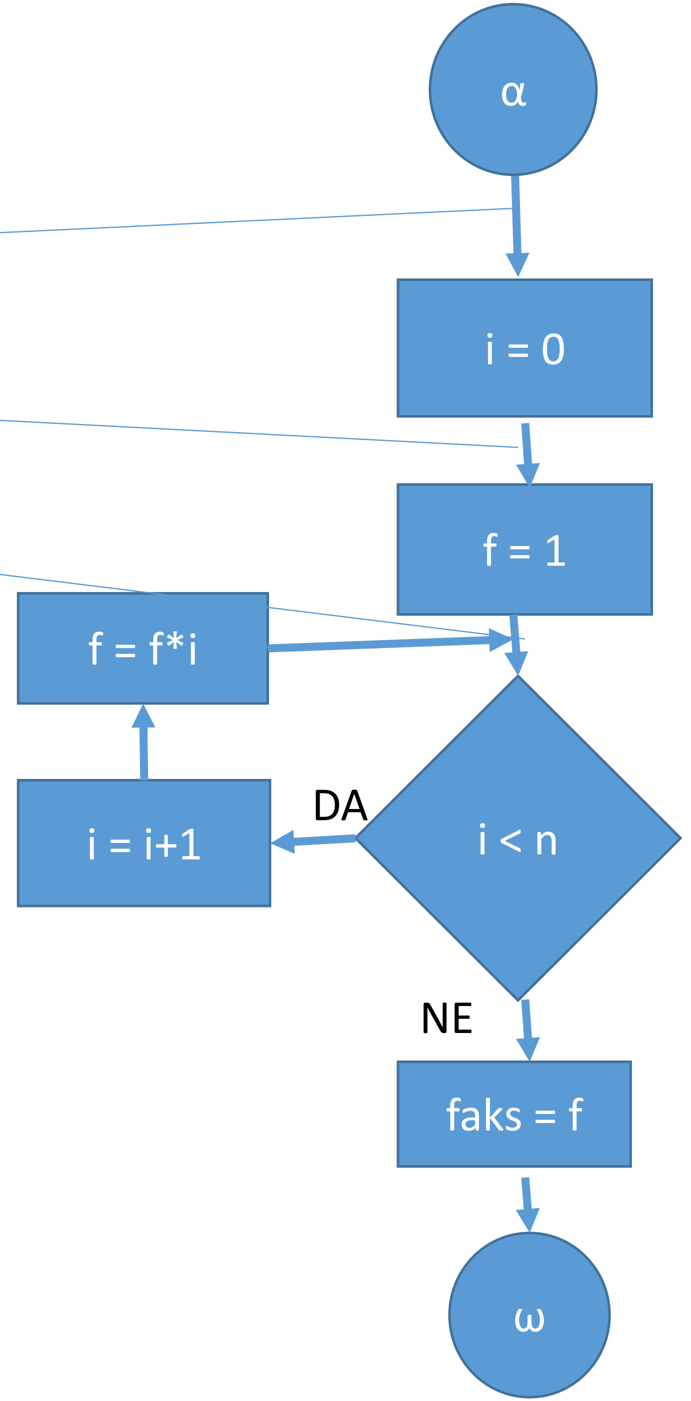
Primer: Izračun faktoriele

n >= 0

i = 0

f = 1

torej velja: f = i!



Primer: Izračun faktoriele

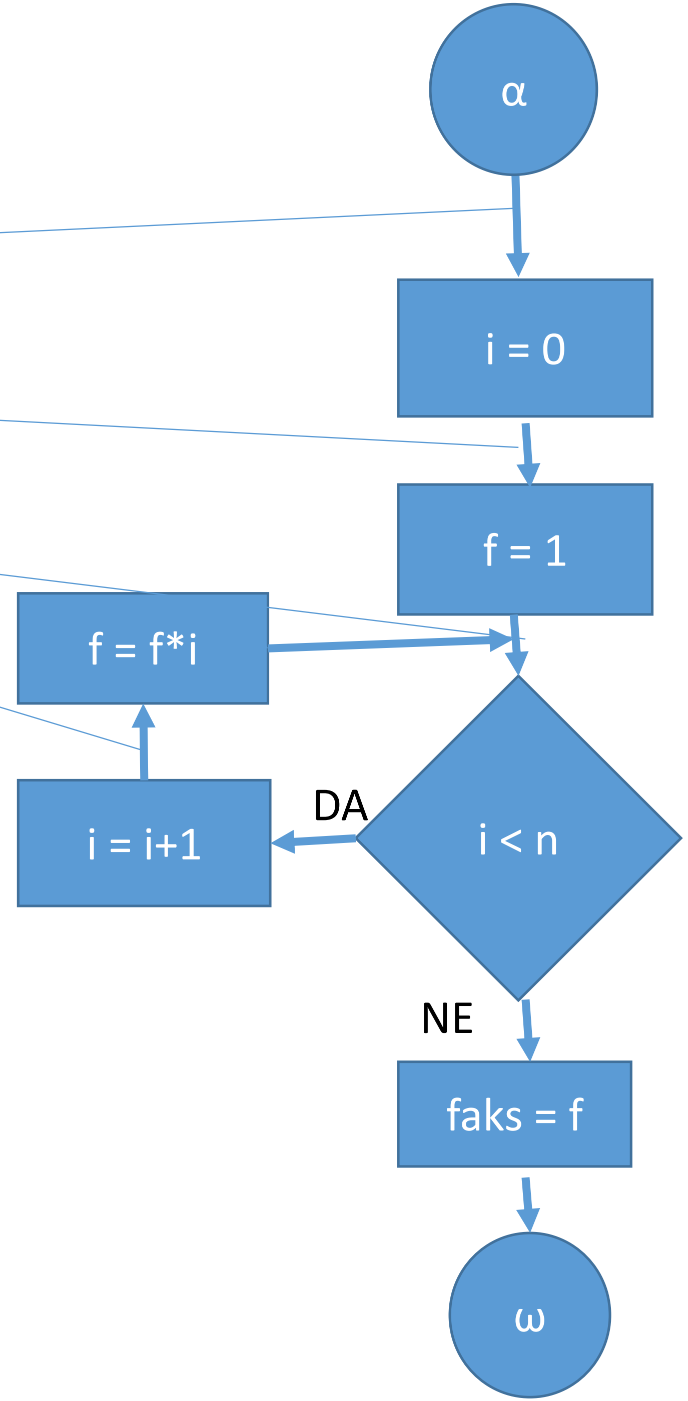
n >= 0

i = 0

f = 1

torej velja: f = i!

i < n, i-1 < n → i <= n, f = (i-1)!



Primer: Izračun faktoriele

n >= 0

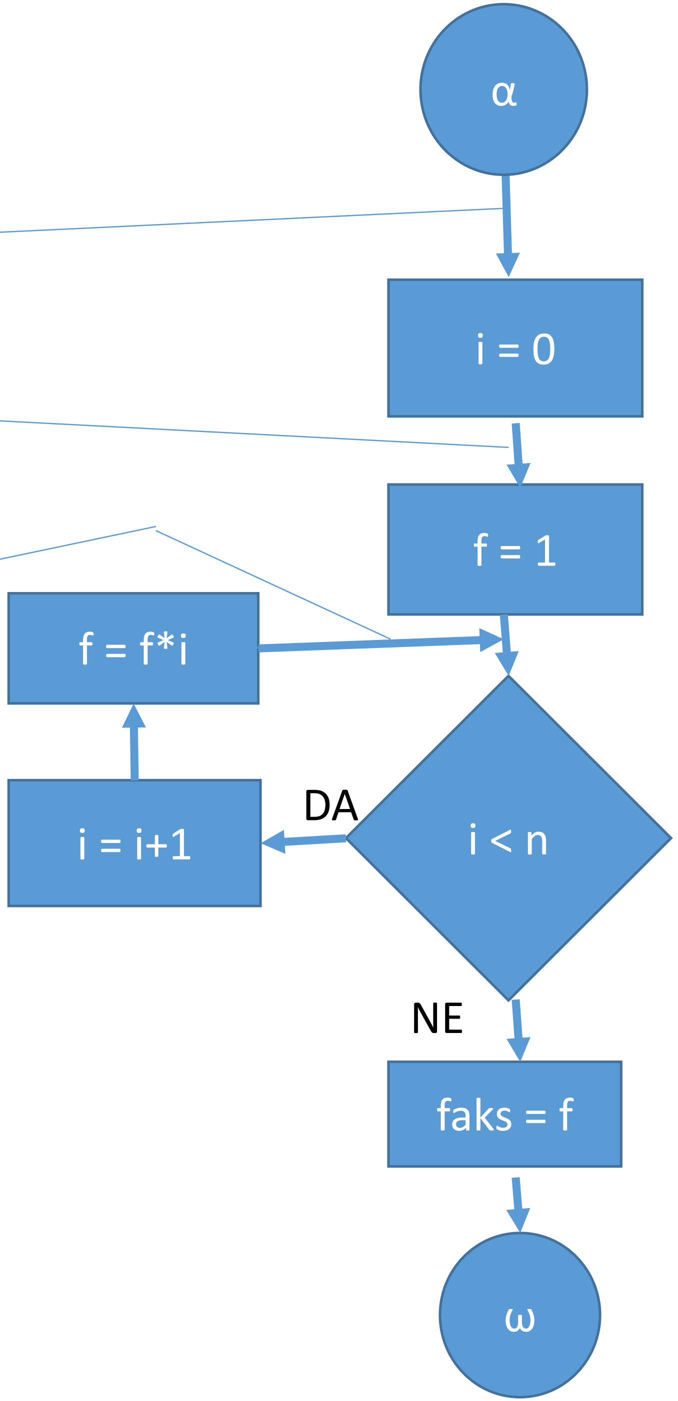
i = 0

f = 1

torej velja: f = i!

i < n, i-1 < n → i <= n, f = (i-1)!

po množenju velja f = i!



Primer: Izračun faktoriele

n >= 0

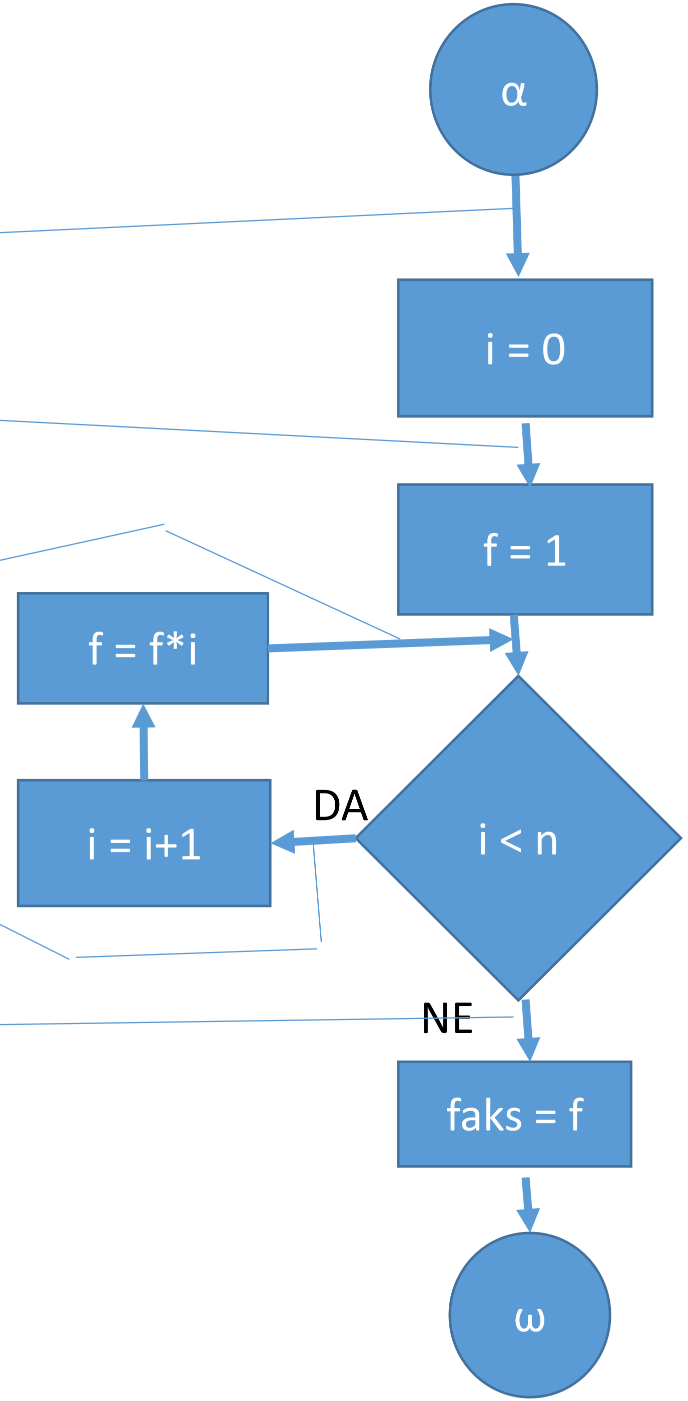
i = 0

f = 1

torej velja: f = i!

i < n, i-1 < n → i <= n, f = (i-1)!

invarianta torej velja: f = i!



Primer: Izračun faktoriele

n >= 0

i = 0

f = 1

torej velja: f = i!

i < n, i-1 < n → i <= n, f = (i-1)!

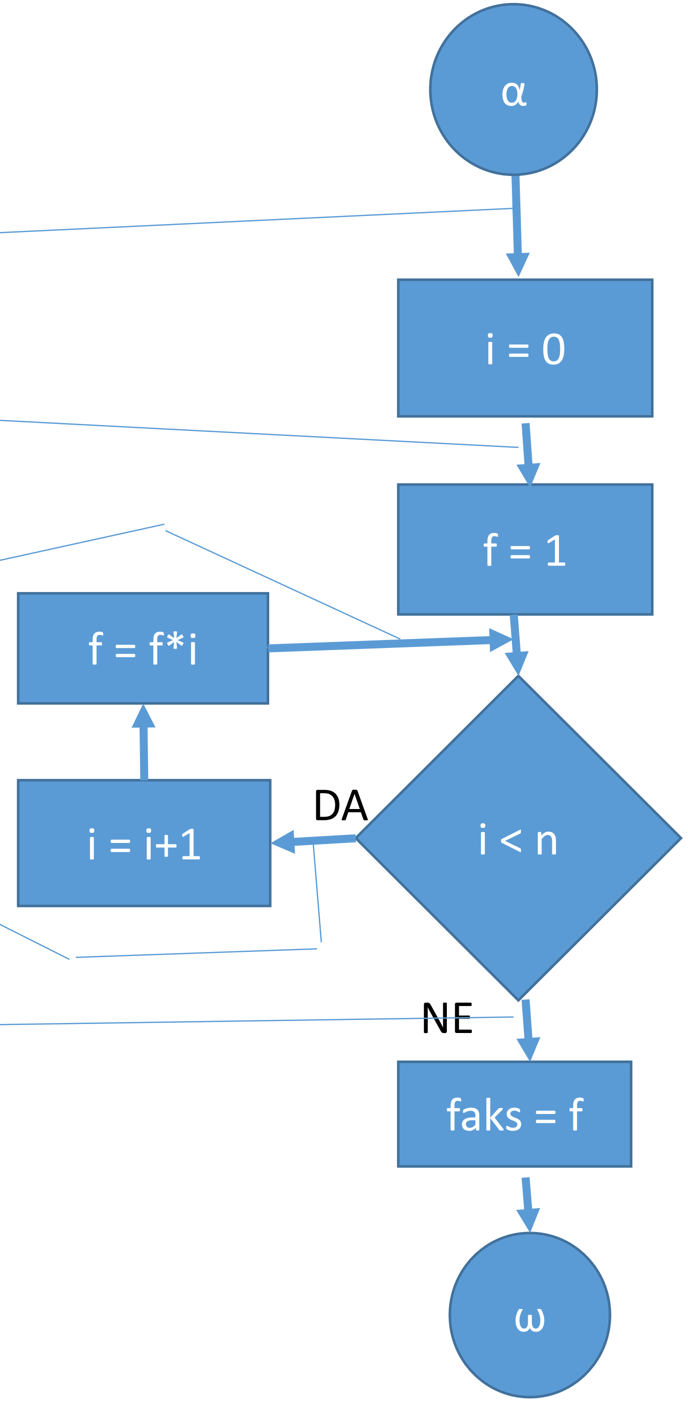
invarianta torej velja: f = i!

V izstopni točki zanke velja:

1) f = i!

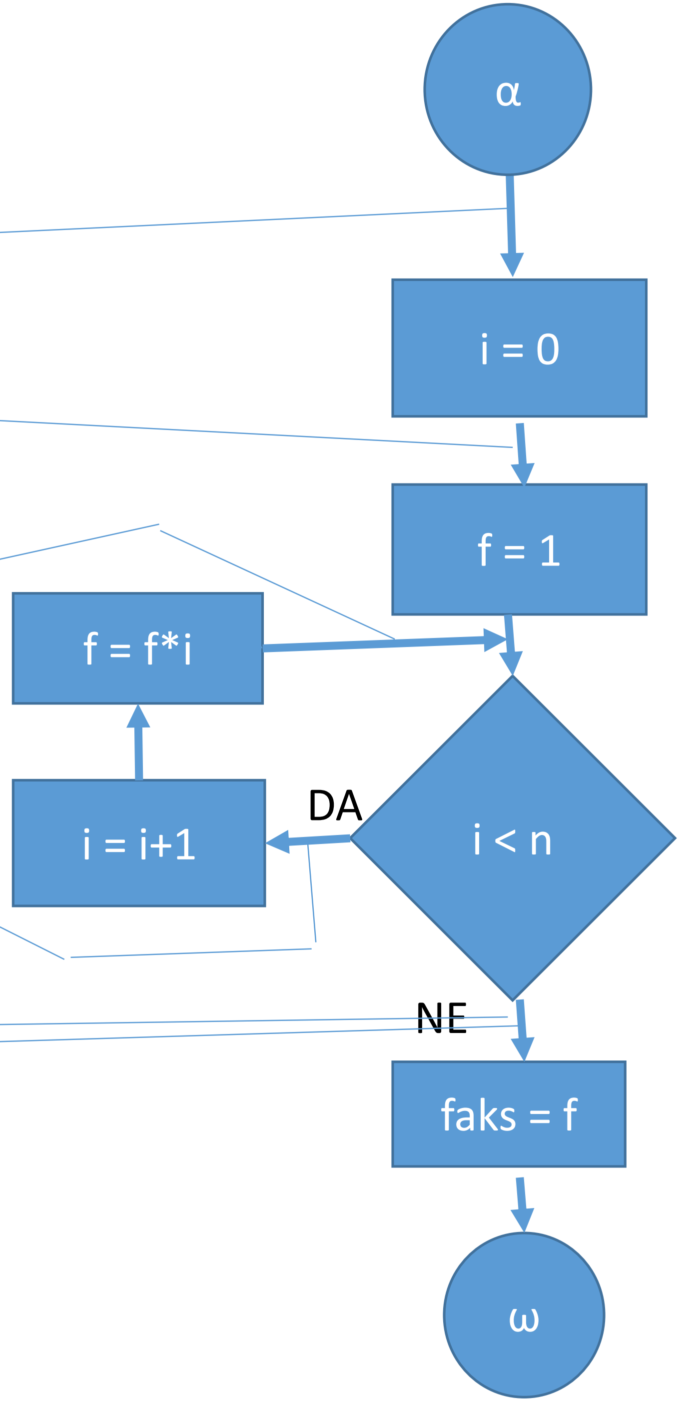
2) i >= n

3) i <= n



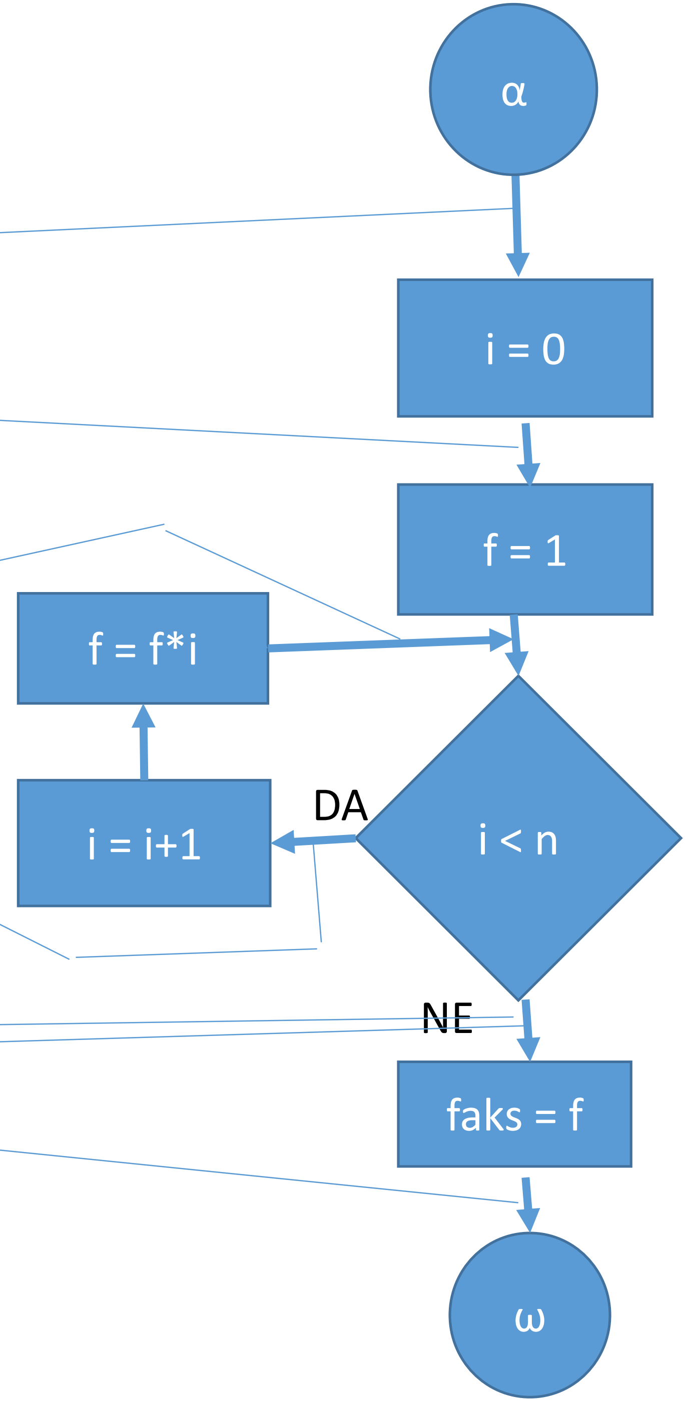
Primer: Izračun faktoriele

n >= 0
i = 0
f = 1
torej velja: f = i!
i < n, i-1 < n → i <= n, f = (i-1)!
invarianta torej velja: f = i!
V izstopni točki zanke velja:
1) f = i!
2) i >= n
3) i <= n
torej → i = n, f = n!



Primer: Izračun faktoriele

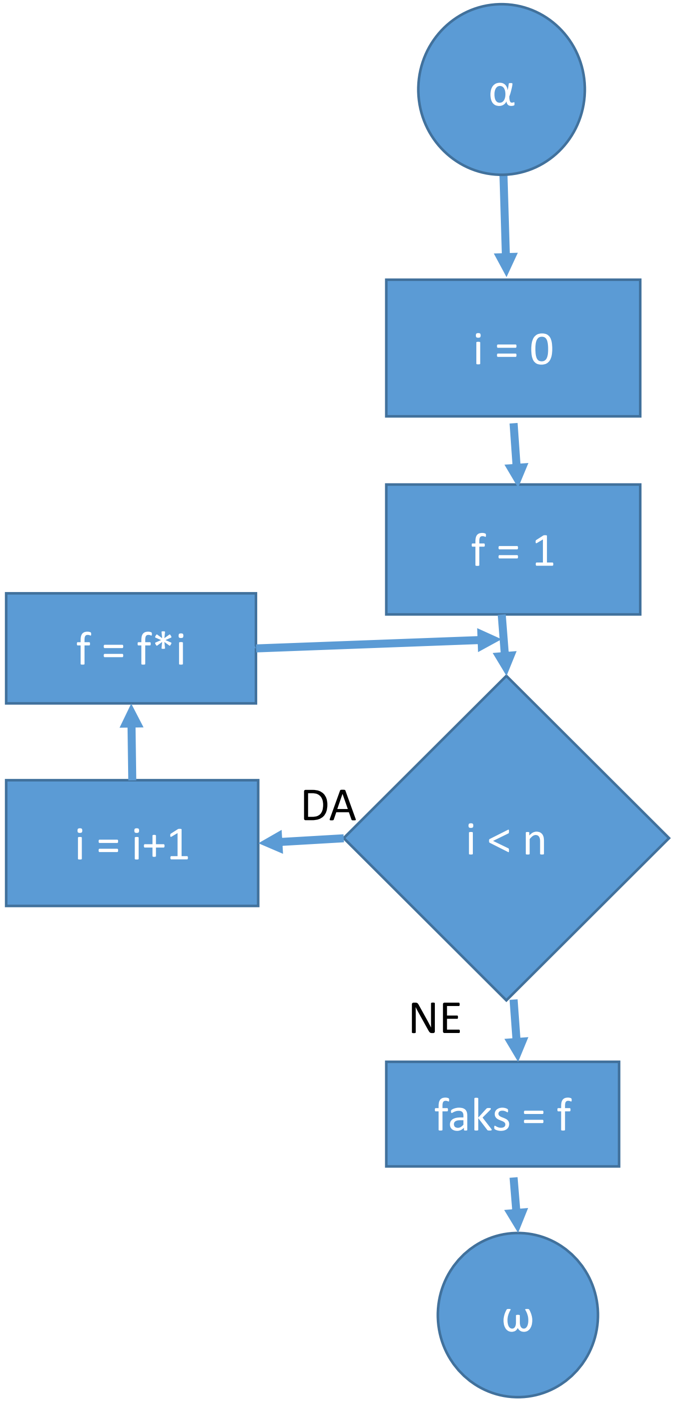
n >= 0
i = 0
f = 1
torej velja: f = i!
i < n, i-1 < n → i <= n, f = (i-1)!
invarianta torej velja: f = i!
V izstopni točki zanke velja:
1) f = i!
2) i >= n
3) i <= n
torej → i = n, f = n!
Torej v zaključni točki velja faks = f = n!
PROGRAM JE PARCIALNO PRAVILEN.



Primer: Izračun faktoriele

Treba je dokazati še ustavljivost programa.

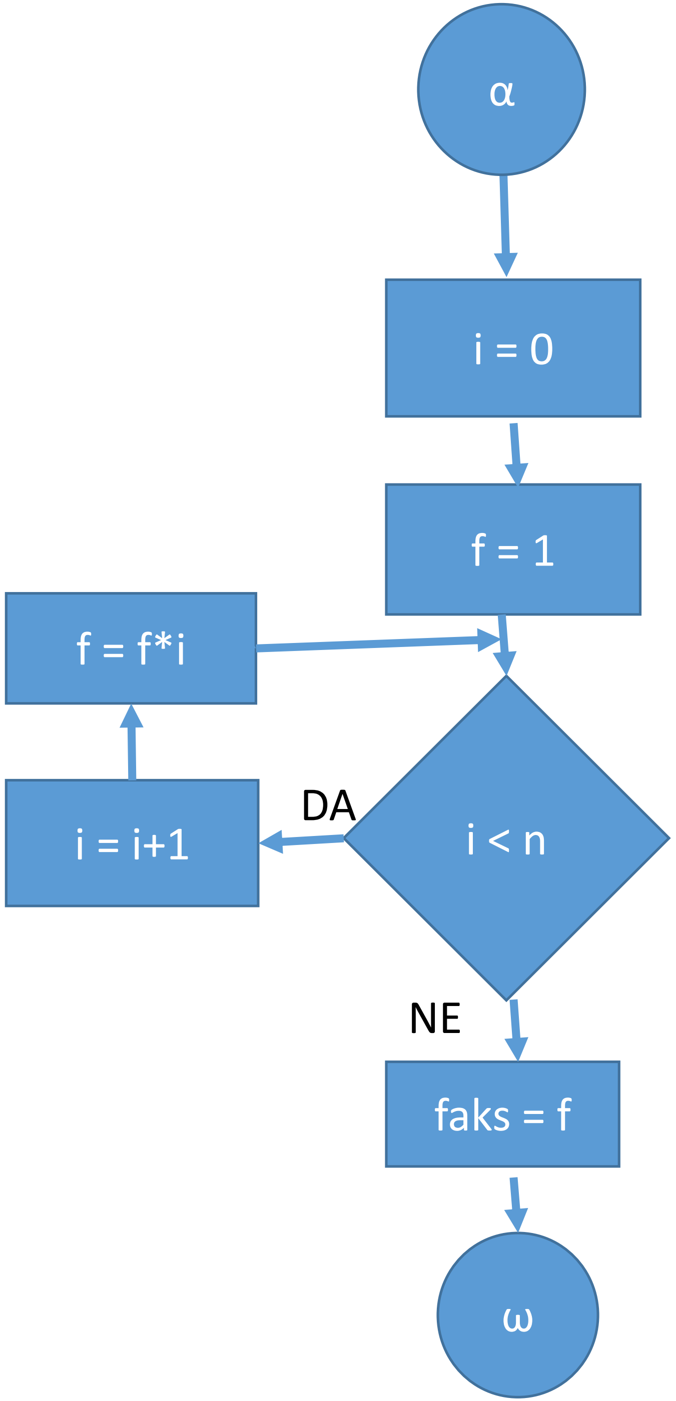
1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka:
3) Zančna invarianta:



Primer: Izračun faktoriele

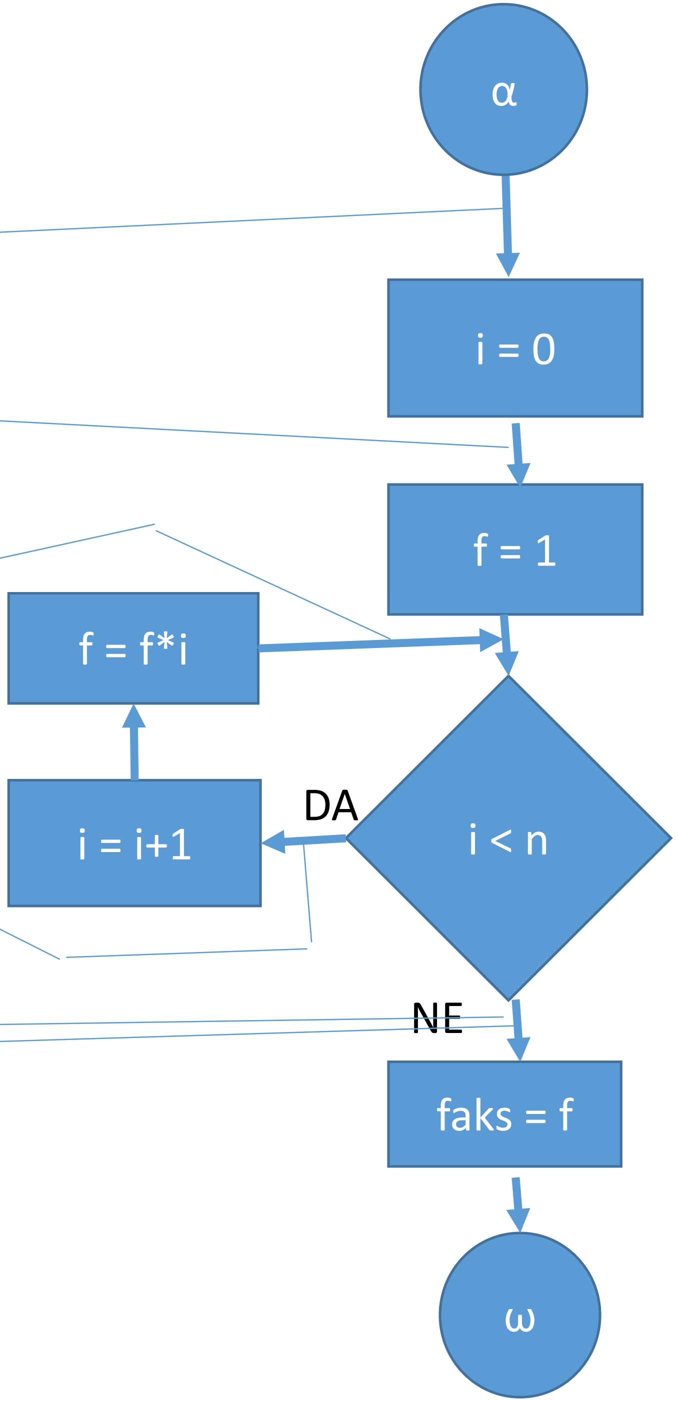
Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: $l = n - i$
3) Zančna invarianta: $n - i \in N$



Primer: Izračun faktoriele

n >= 0
i = 0
f = 1
torej velja: f = i!
i < n, i-1 < n → i <= n, f = (i-1)!
invarianta torej velja: f = i!
V izstopni točki zanke velja:
1) f = i!
2) i >= n
3) i <= n
torej → i = n, f = n!



Primer: Izračun faktoriele

$n >= 0$

$i = 0$

torej velja: $n - i \in N$

$i < n$, $i-1 < n \rightarrow i <= n$

invarianta torej velja: $n - i \in N$

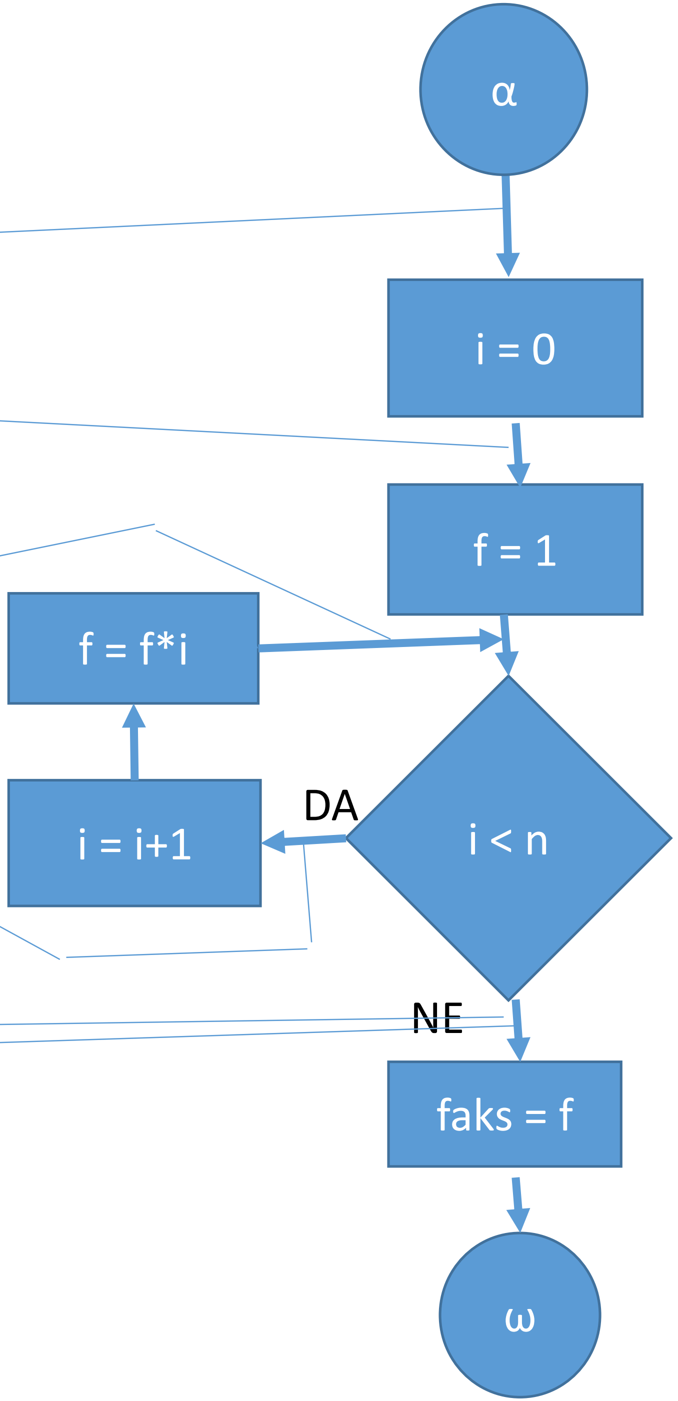
V izstopni točki zanke velja:

1) $f = i!$

2) $i >= n$

3) $i <= n$

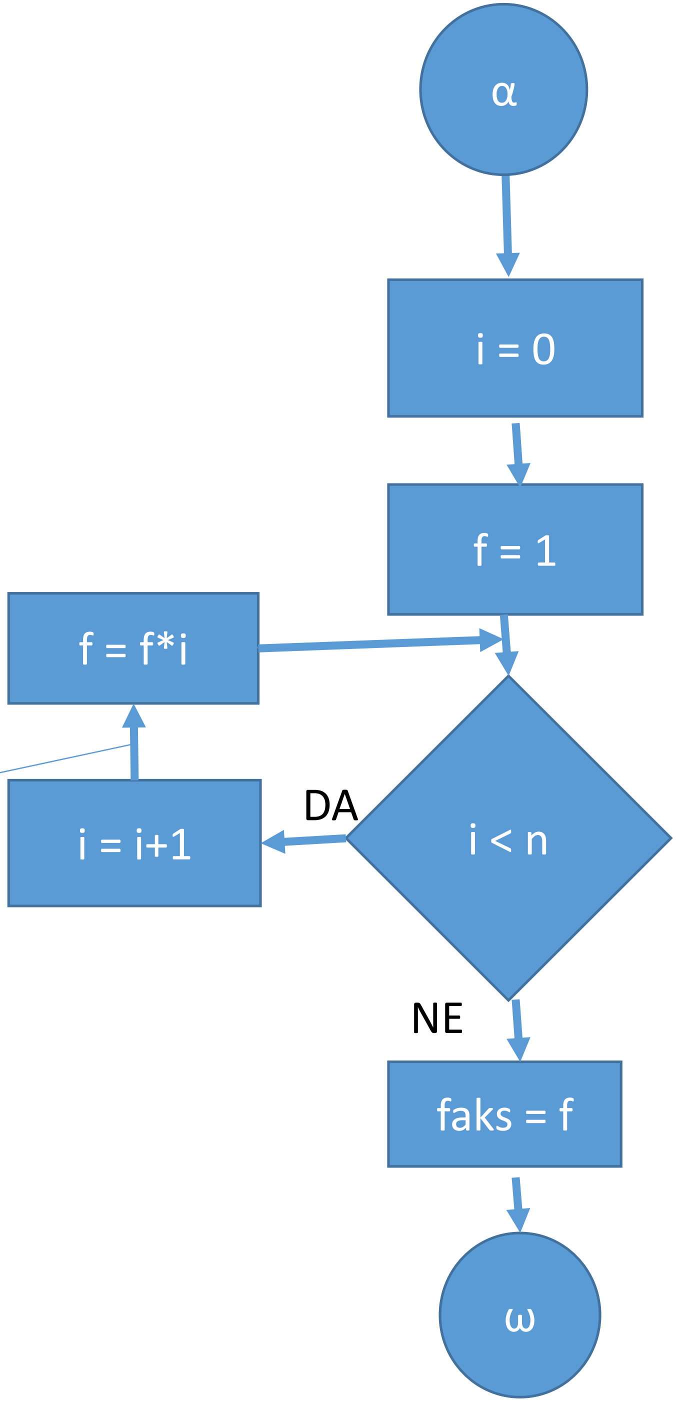
torej $\rightarrow$ $i = n$, $n - i \in N$



Primer: Izračun faktoriele

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: $l = n - i$
3) Zančna invarianta: $n - i \in N$ VELJA
4) Zančna spremenljivka se zmanjšuje:
   $ll = n - i - 1 < l = n - i$



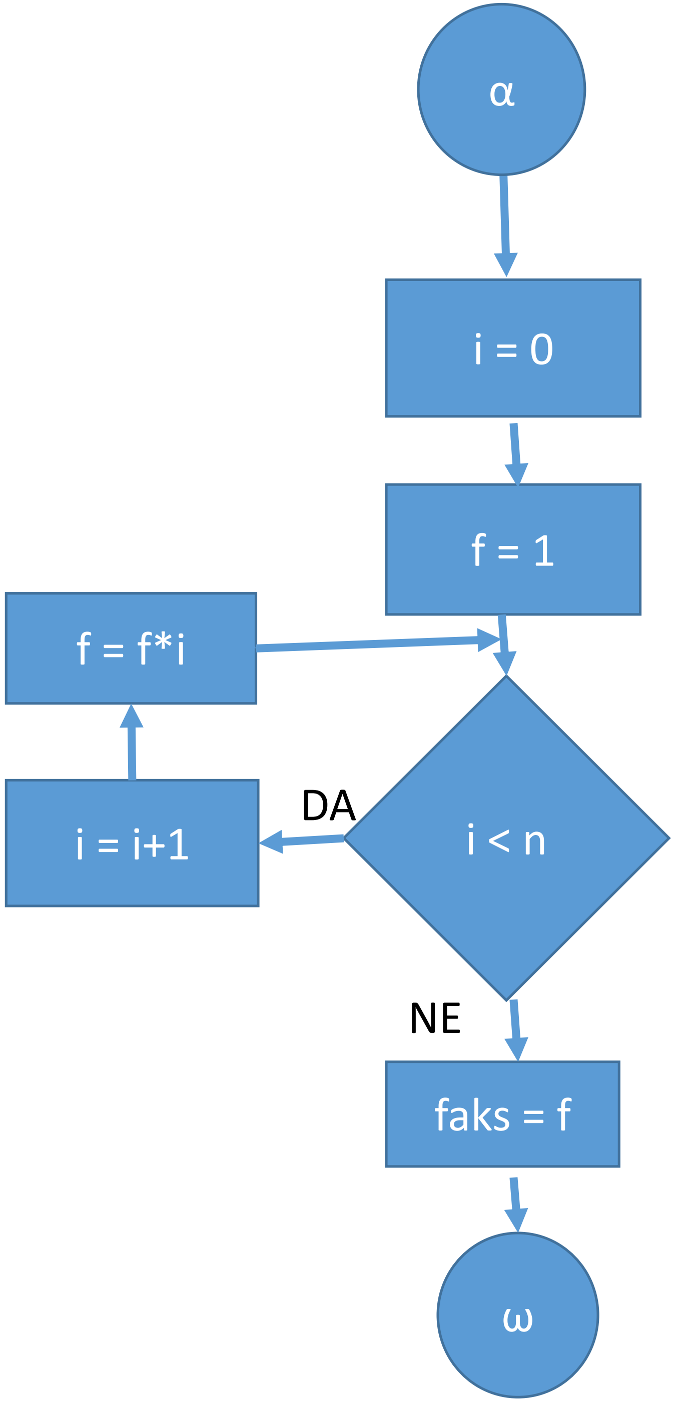
Primer: Izračun faktoriele

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: $l = n - i$
3) Zančna invarianta: $n - i \in N$ VELJA
4) Zančna spremenljivka se zmanjšuje:
   $ll = n - i - 1 < l = n - i$

PROGRAM JE TOTALNO PRAVILEN:

- *Je parcialno pravilen*
- *Se vedno ustavi*



Primer: Množenje z inkrementom

- Začetni pogoj: $\phi(x,y) = (x,y \in \mathrm{N} \cup \{0\})$

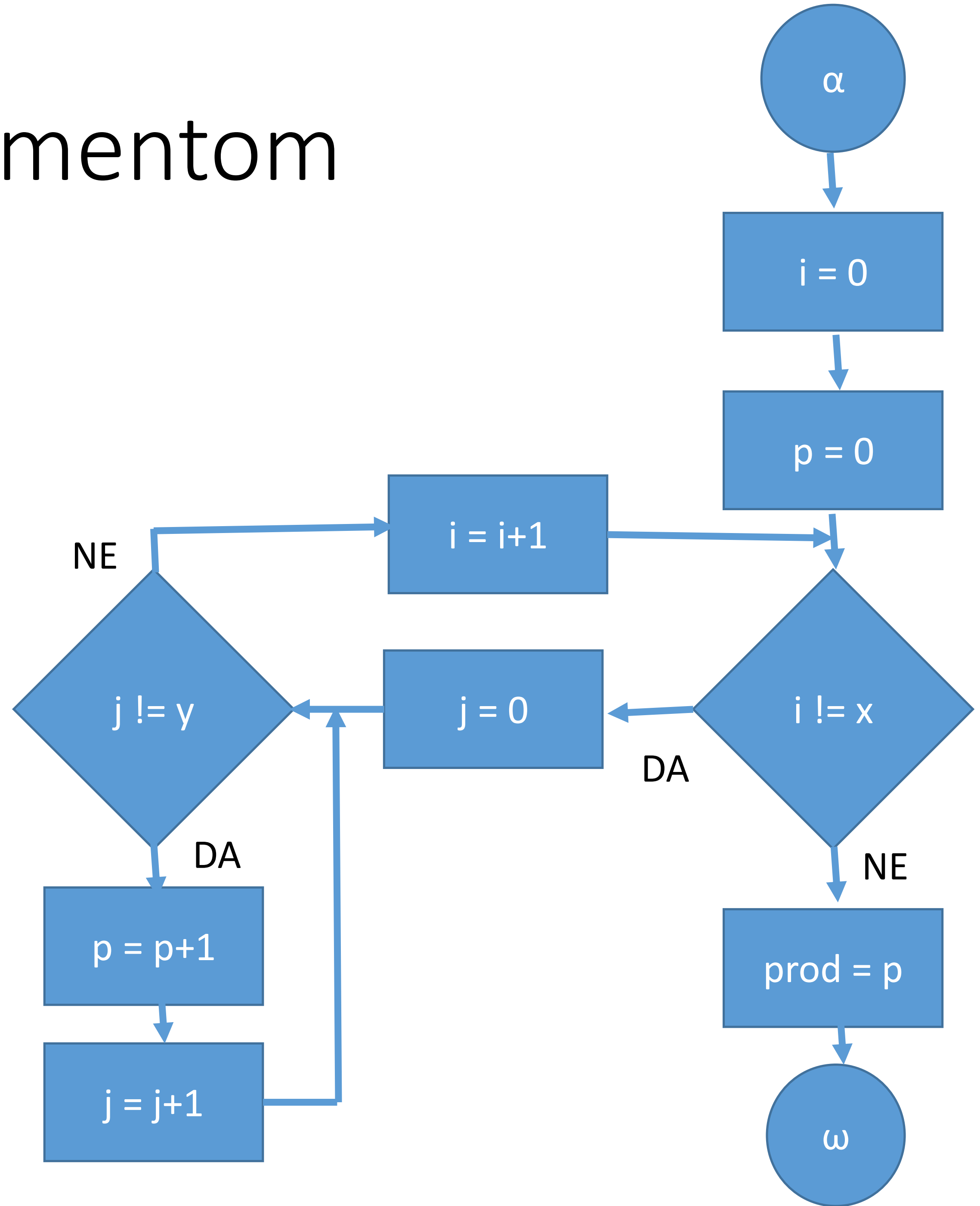
- Zaključni pogoj: $\psi(prod,x,y) = (prod = x \times y)$

Primer: Množenje z inkrementom

```
static public int prod(int x, int y) {
  // fi(x, y) = (x,y >= 0)
  int i, j, p ;

  i=0 ;
  p=0 ;
  while (i != x) {
    j=0 ;
    while (j != y) {
      p++ ;
      j++ ;
    } // while j
    i++ ;
  } // while i
  return p ;
  // psi(x, y, prod) = (prod = x * y)
} // prod
```

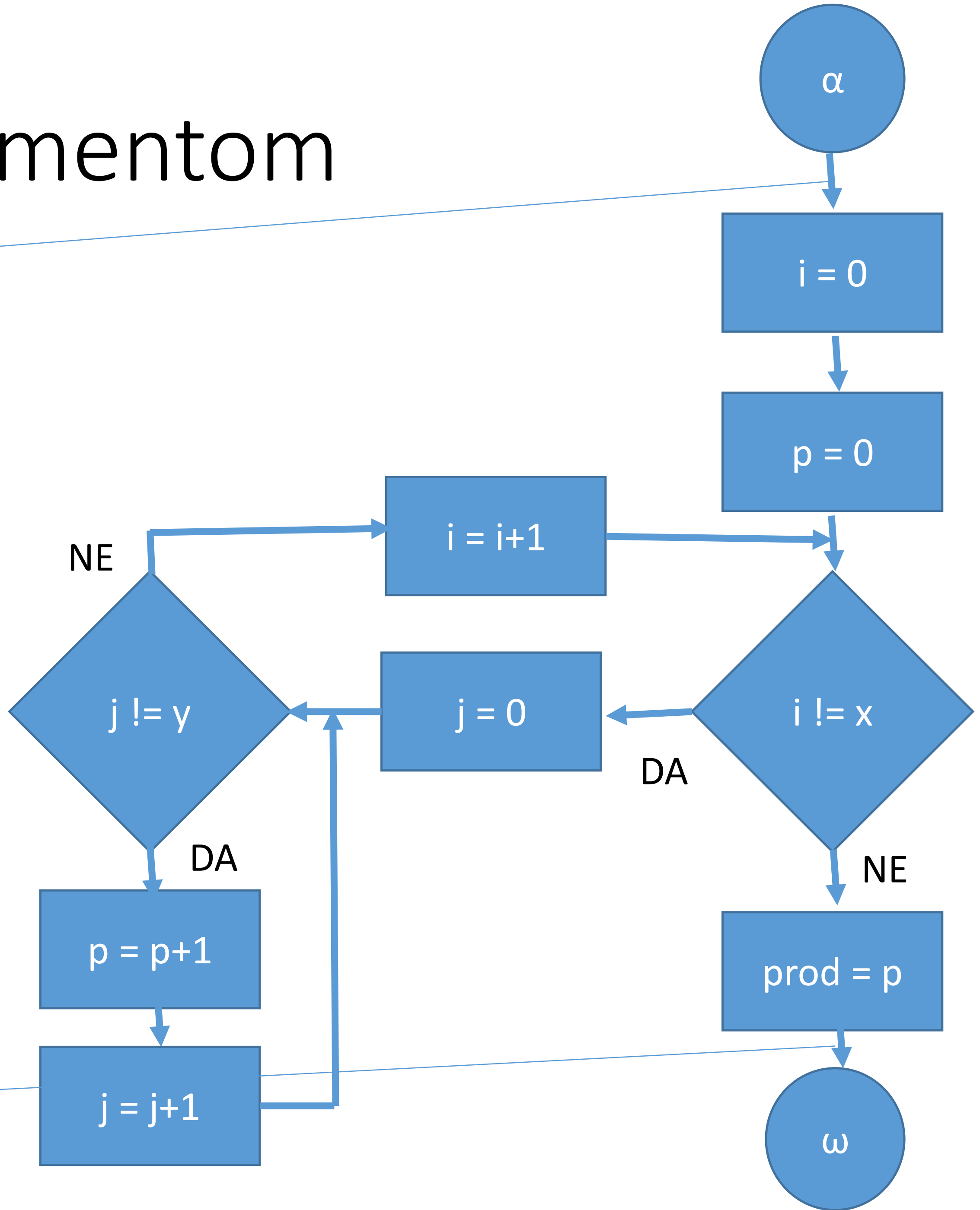
Primer: Množenje z inkrementom

```
static public int prod(int x, int y) {
    // fi(x, y) = (x,y >= 0)
    int i, j, p ;

    i=0 ;
    p=0 ;
    while (i != x) {
        j=0 ;
        while (j != y) {
            p++ ;
            j++ ;
        } // while j
        i++ ;
    } // while i
    return p ;
    // psi(x, y, prod) = (prod = x * y)
} // prod
```



Primer: Množenje z inkrementom

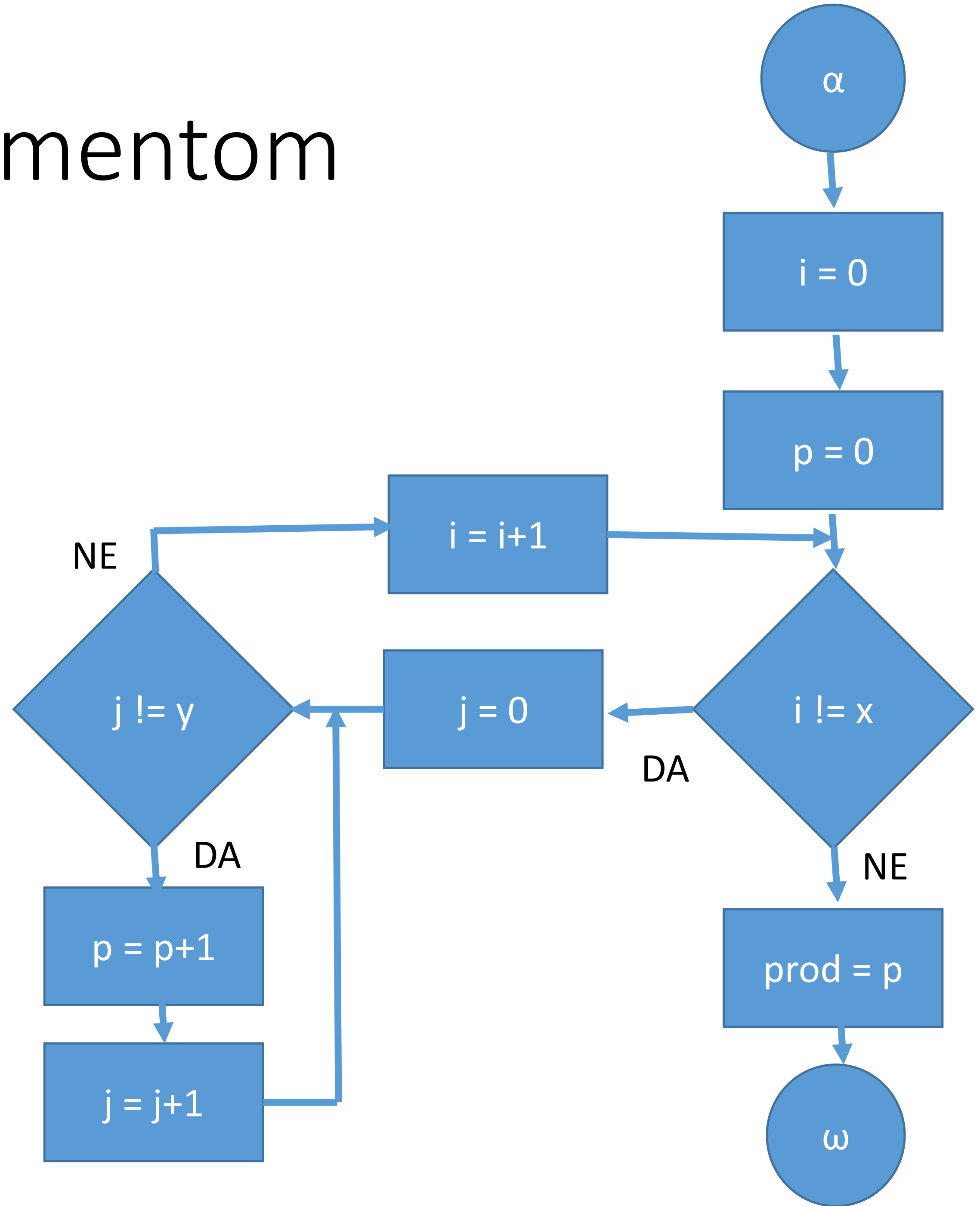
```
static public int prod(int x, int y) {
   // fi(x, y) = (x,y >= 0)
   int i, j, p ;

   i=0 ;
   p=0 ;
   while (i != x) {
      j=0 ;
      while (j != y) {
         p++ ;
         j++ ;
      } // while j
      i++ ;
   } // while i
   return p ;
   // psi(x, y, prod) = (prod = x * y)
} // prod
```



Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta?



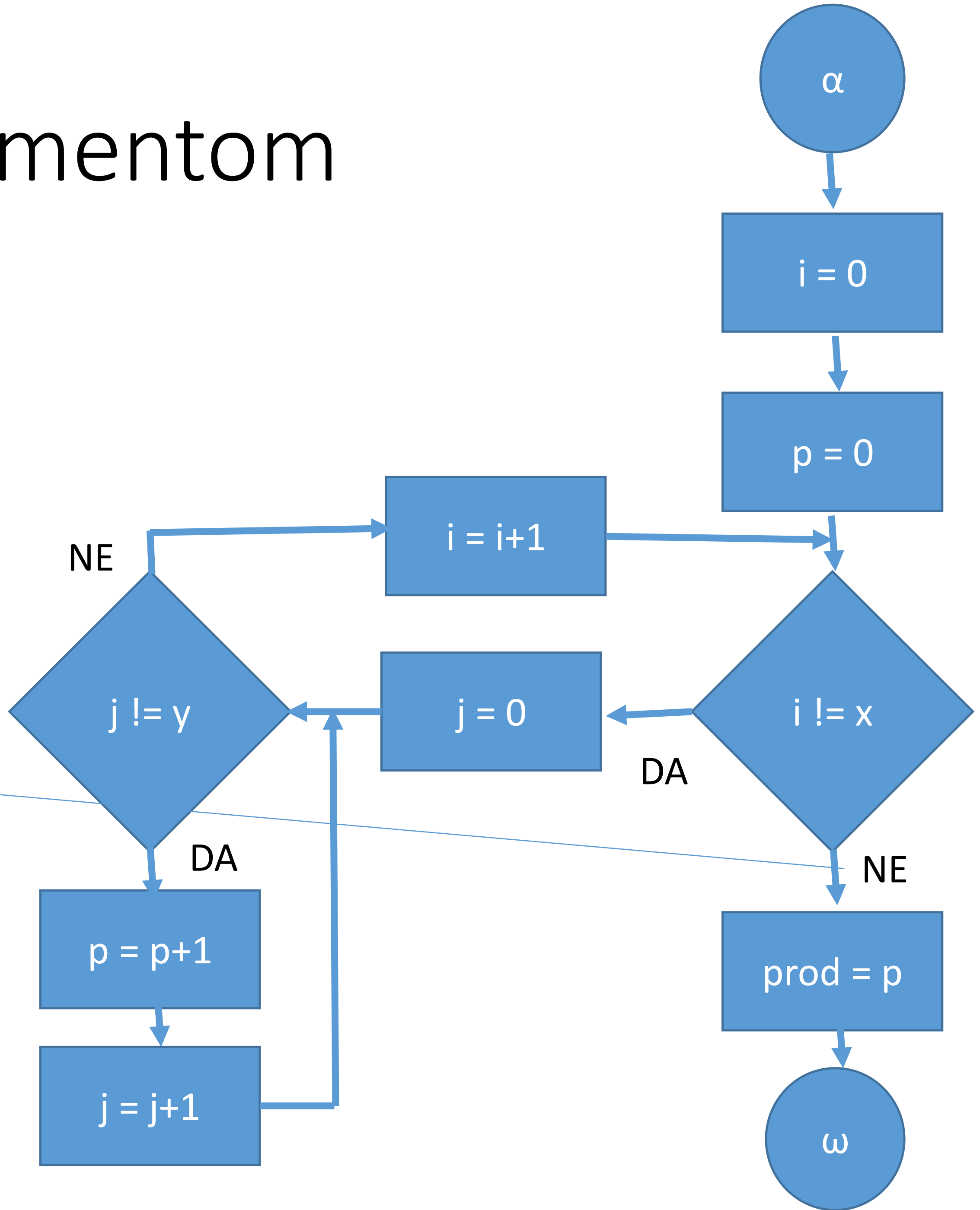
Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta?

Dokazati moramo, da je *prod = x*y*

Torej pred stavkom *prod = p* mora veljati *p = x*y*



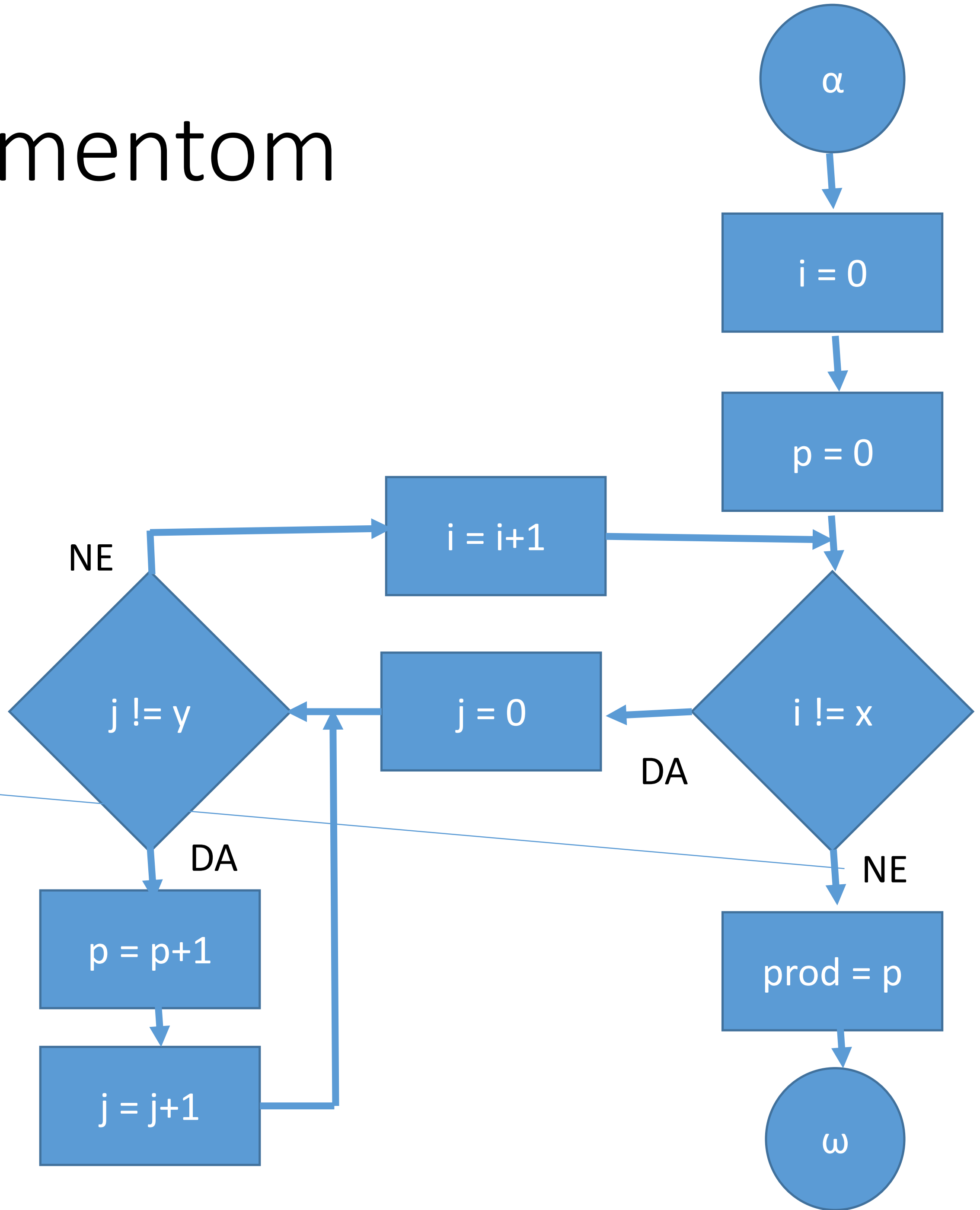
Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.
Kaj je primerna zančna invarianta?

Dokazati moramo, da je *prod = x*y*

Torej pred stavkom *prod = p* mora veljati *p = x*y*

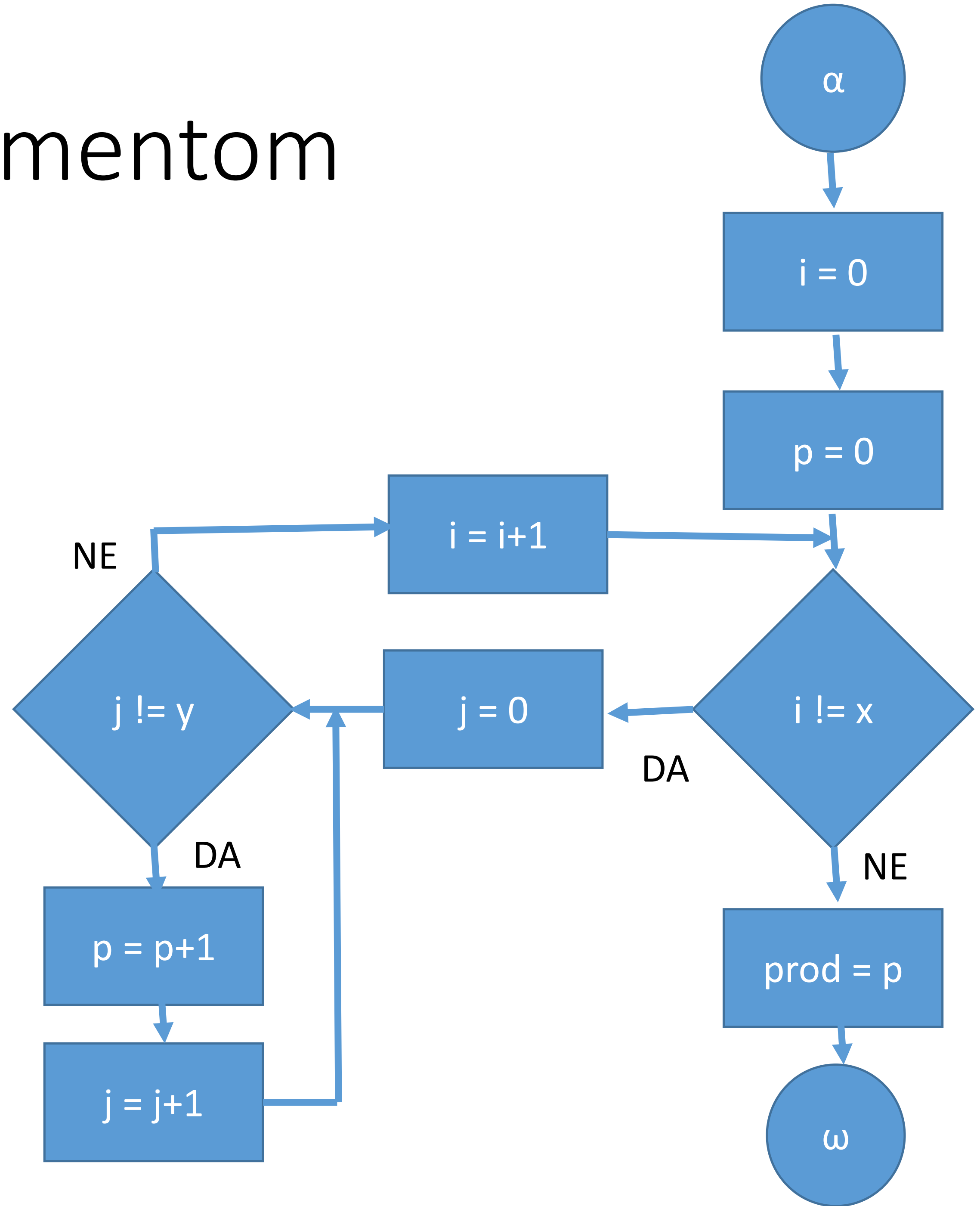
Ker v tej točki ne velja več *i != x*, mora torej veljati *i = x*. Torej je primerna zančna invarianta za zunanjo zanko lahko ***p = i*y***



Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta za notranjo zanko?



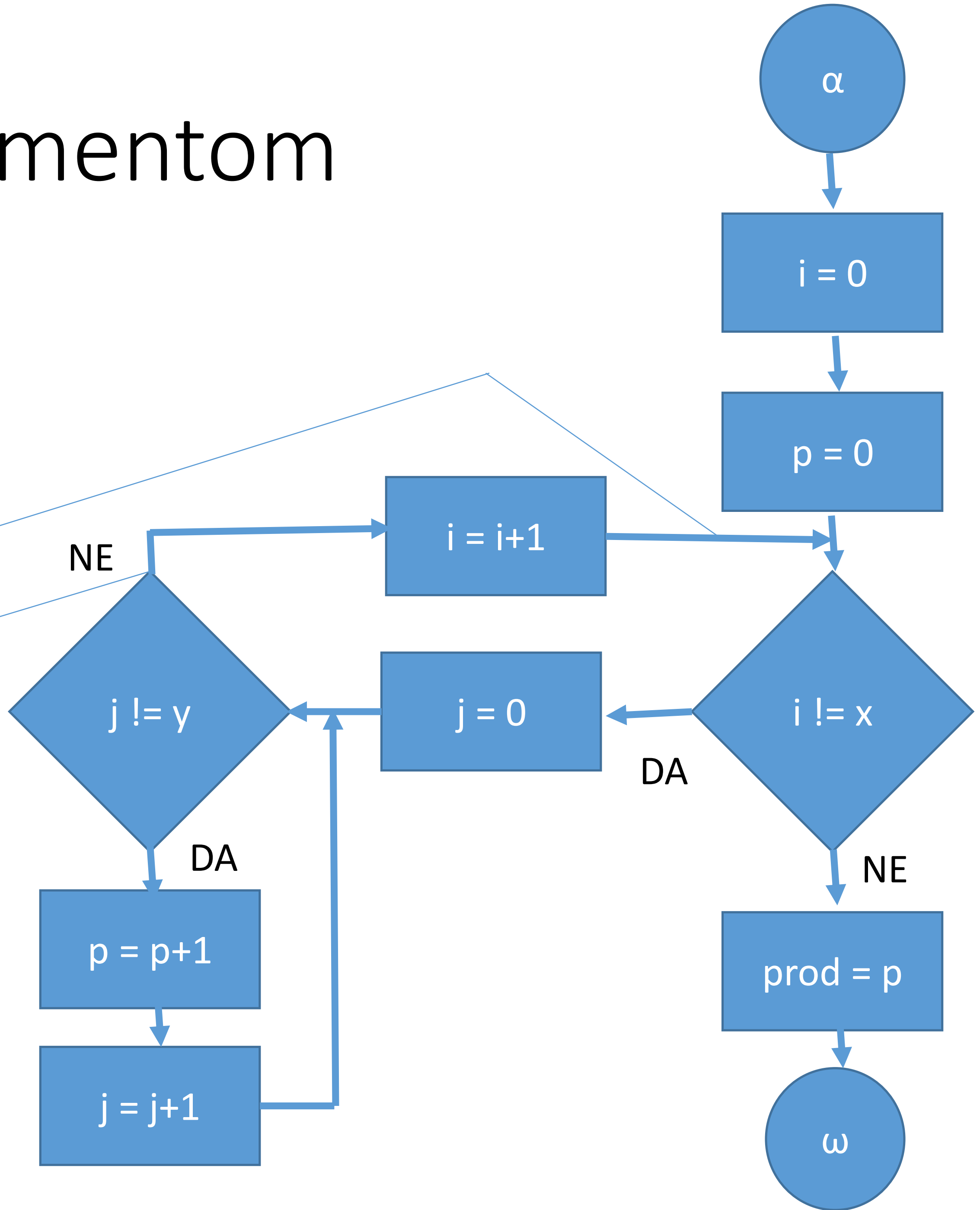
Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta za notranjo zanko?

Dokazati moramo, da je $p = i*y$

Torej pred stavkom $i = i+1$ mora veljati $p = (i+1)*y$



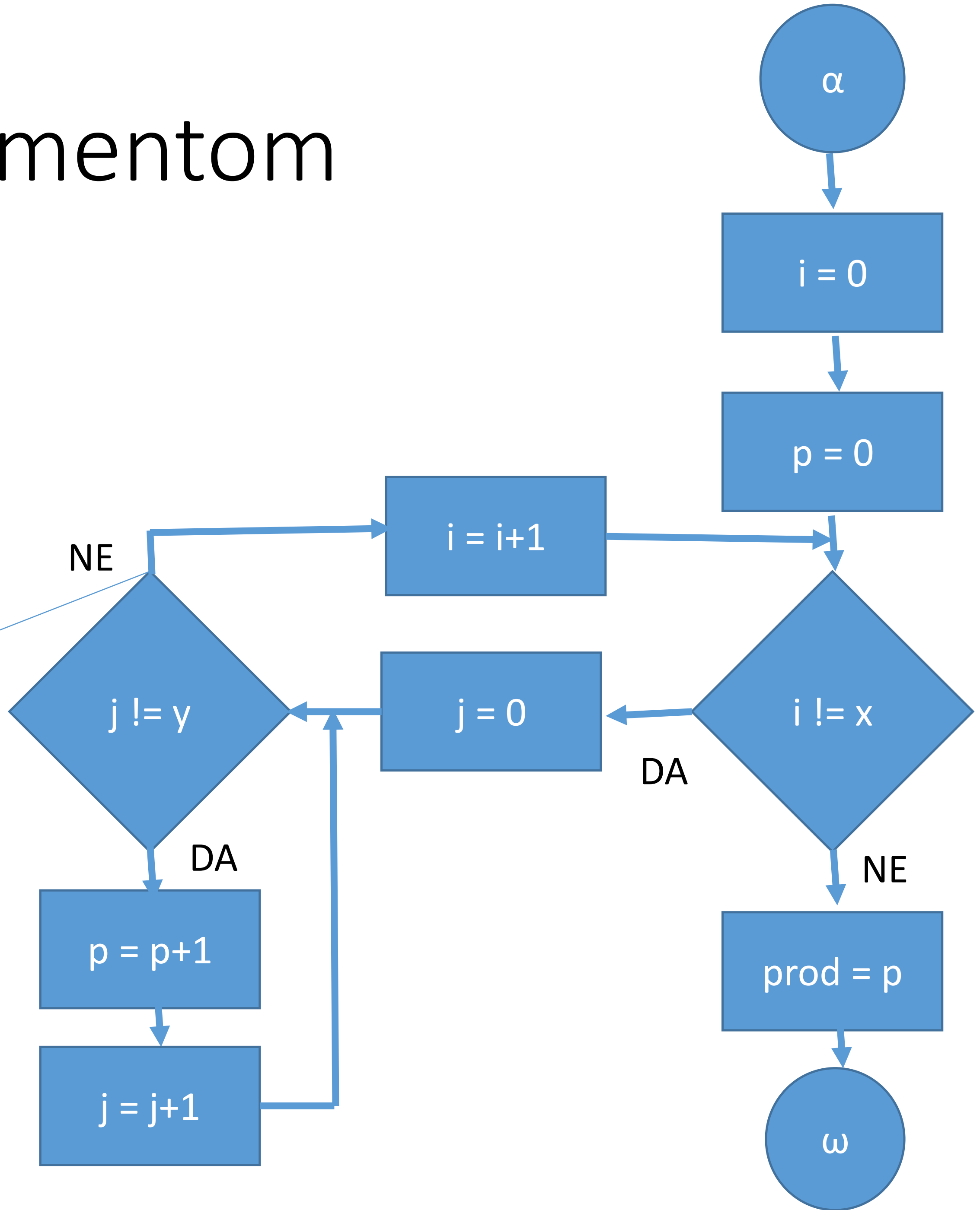
Primer: Množenje z inkrementom

Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta za notranjo zanko?

Dokazati moramo, da je $p = i*y$

Torej pred stavkom $i = i+1$ mora veljati
$p = (i+1)*y = i*y + y$
Ker v tej točki ne velja več $j \mathrel{!=} y$, mora torej veljati $j = y$.



Primer: Množenje z inkrementom

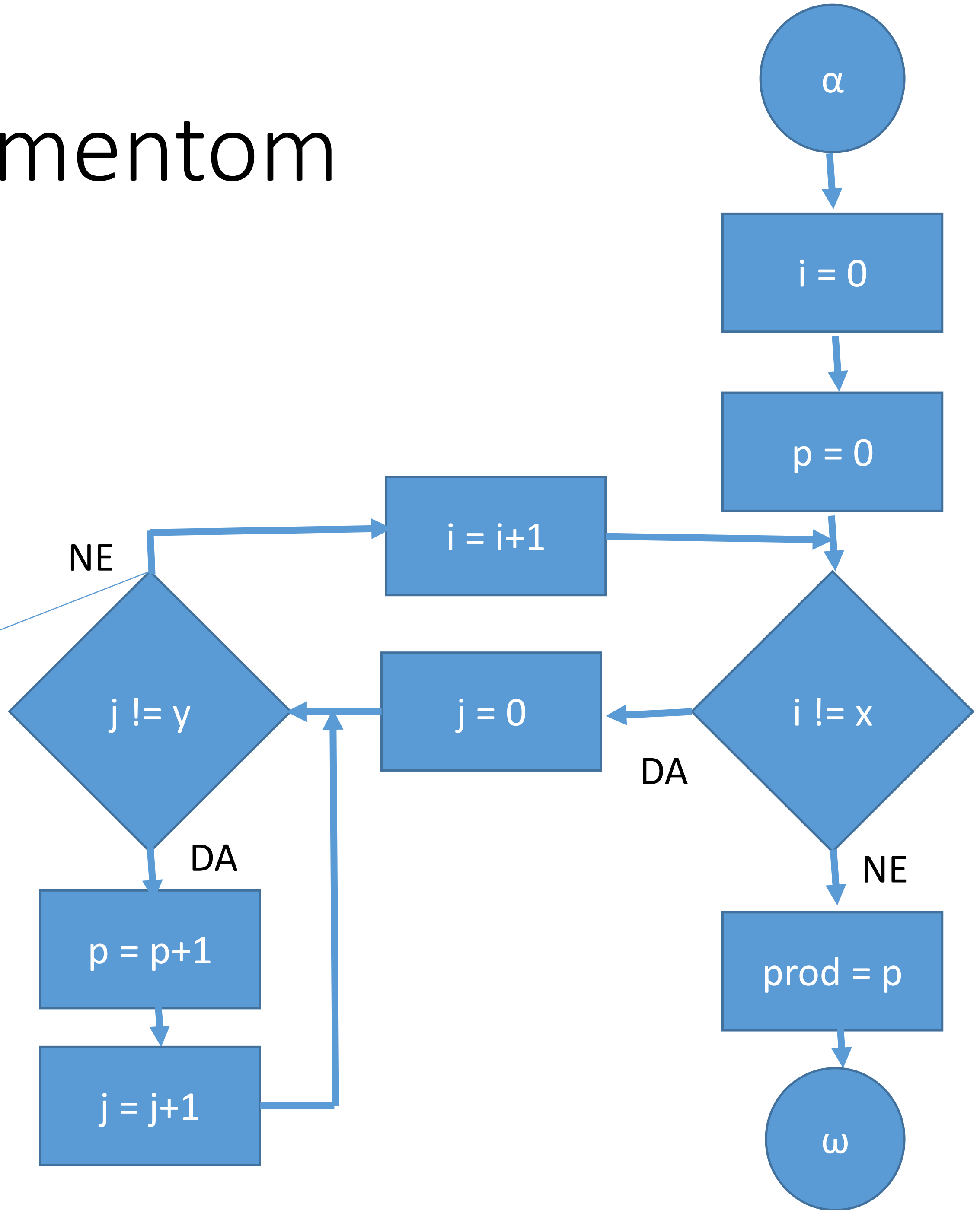
Dve zanki: dve začni invarianti.

Kaj je primerna zančna invarianta za notranjo zanko?

Dokazati moramo, da je $p = i*y$

Torej pred stavkom $i = i+1$ mora veljati $p = (i+1)*y = i*y + y$

Ker v tej točki ne velja več $j\ != y$, mora torej veljati $j = y$. Torej je primerna zančna invarianta za notranjo zanko lahko $p = i*y + j$

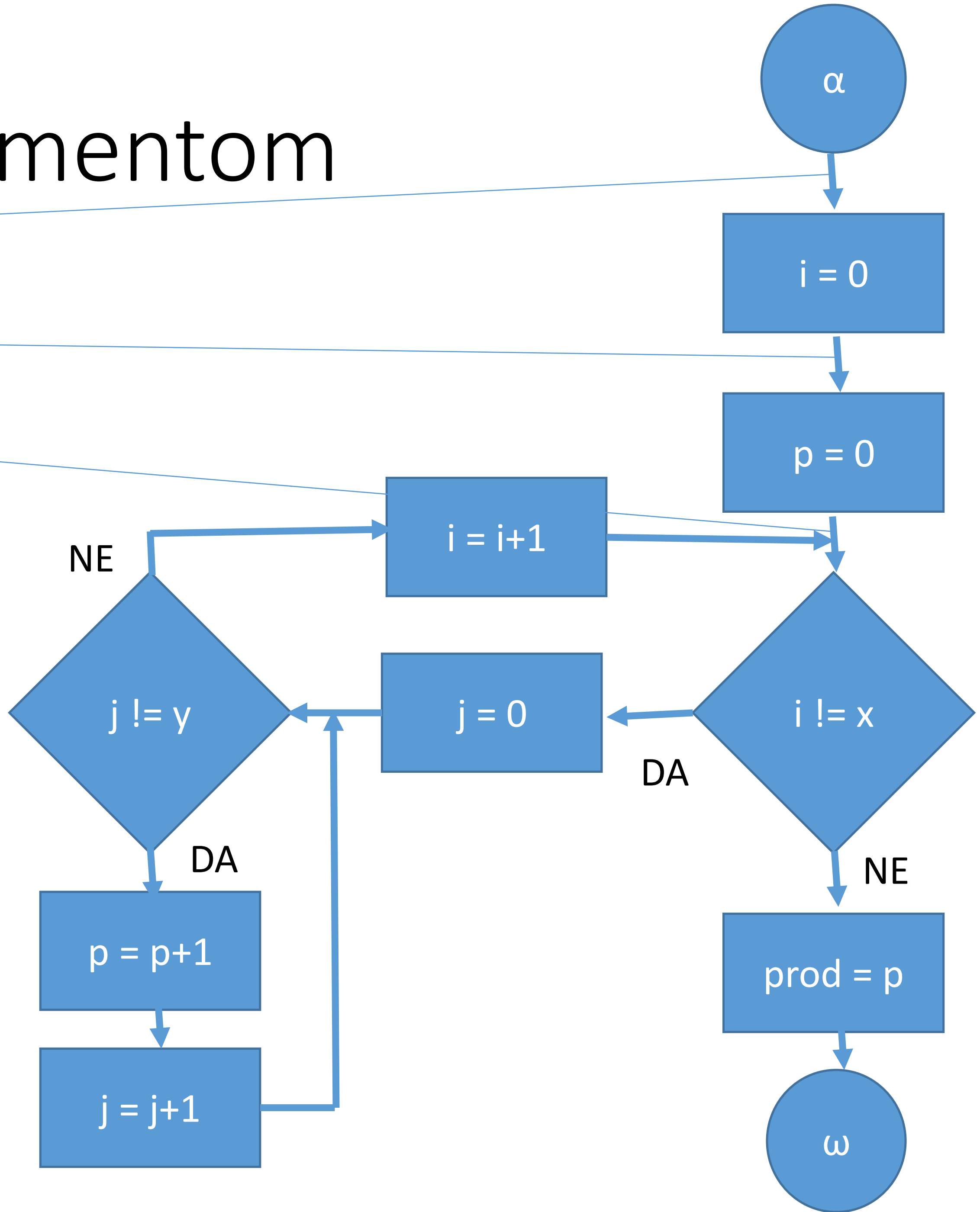


Primer: Množenje z inkrementom

x,y >= 0

i = 0

p = 0



Primer: Množenje z inkrementom

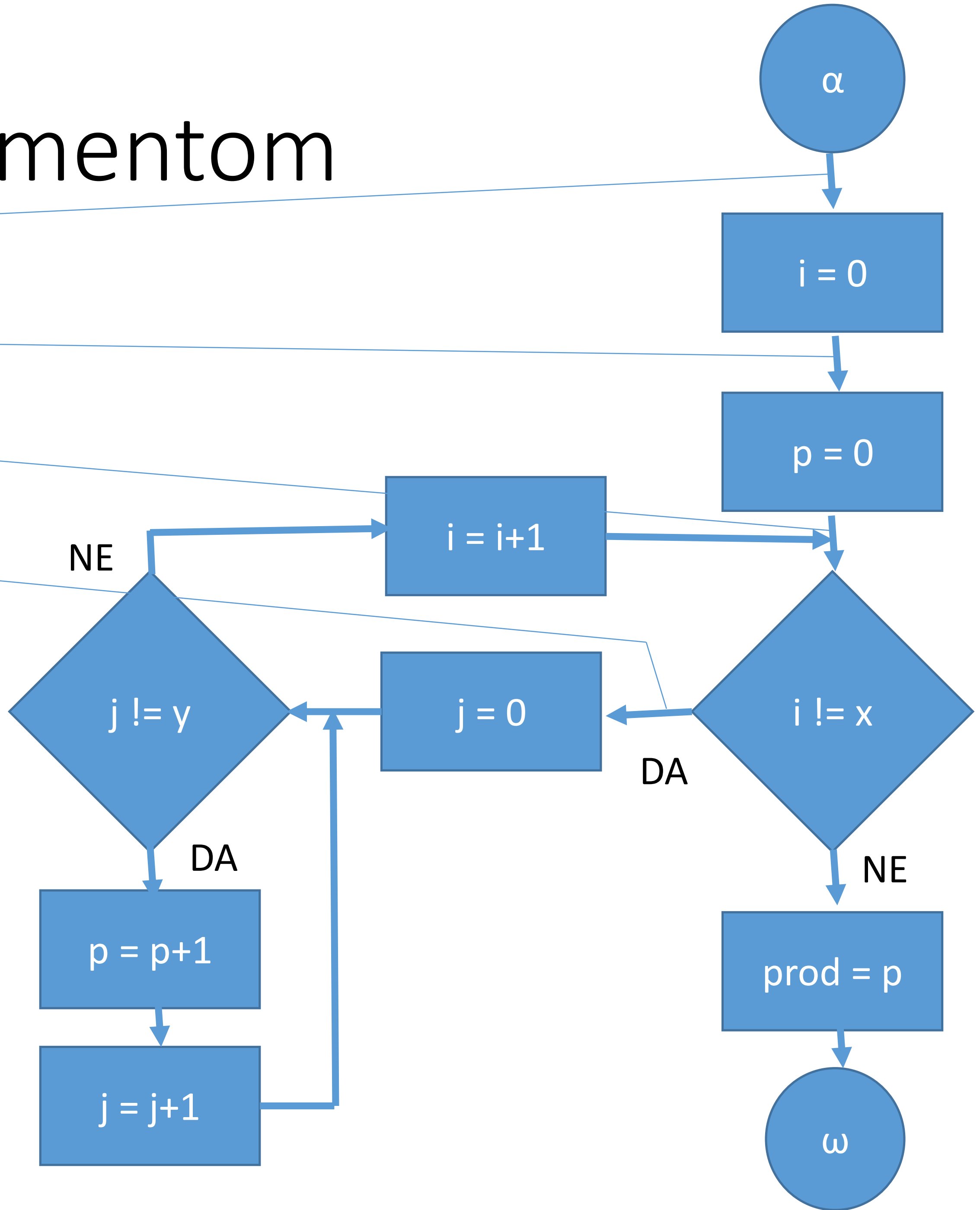
x,y >= 0

i = 0

p = 0

Torej velja: p = i*y

i != x → i < x in p = i*y



Primer: Množenje z inkrementom

x,y >= 0

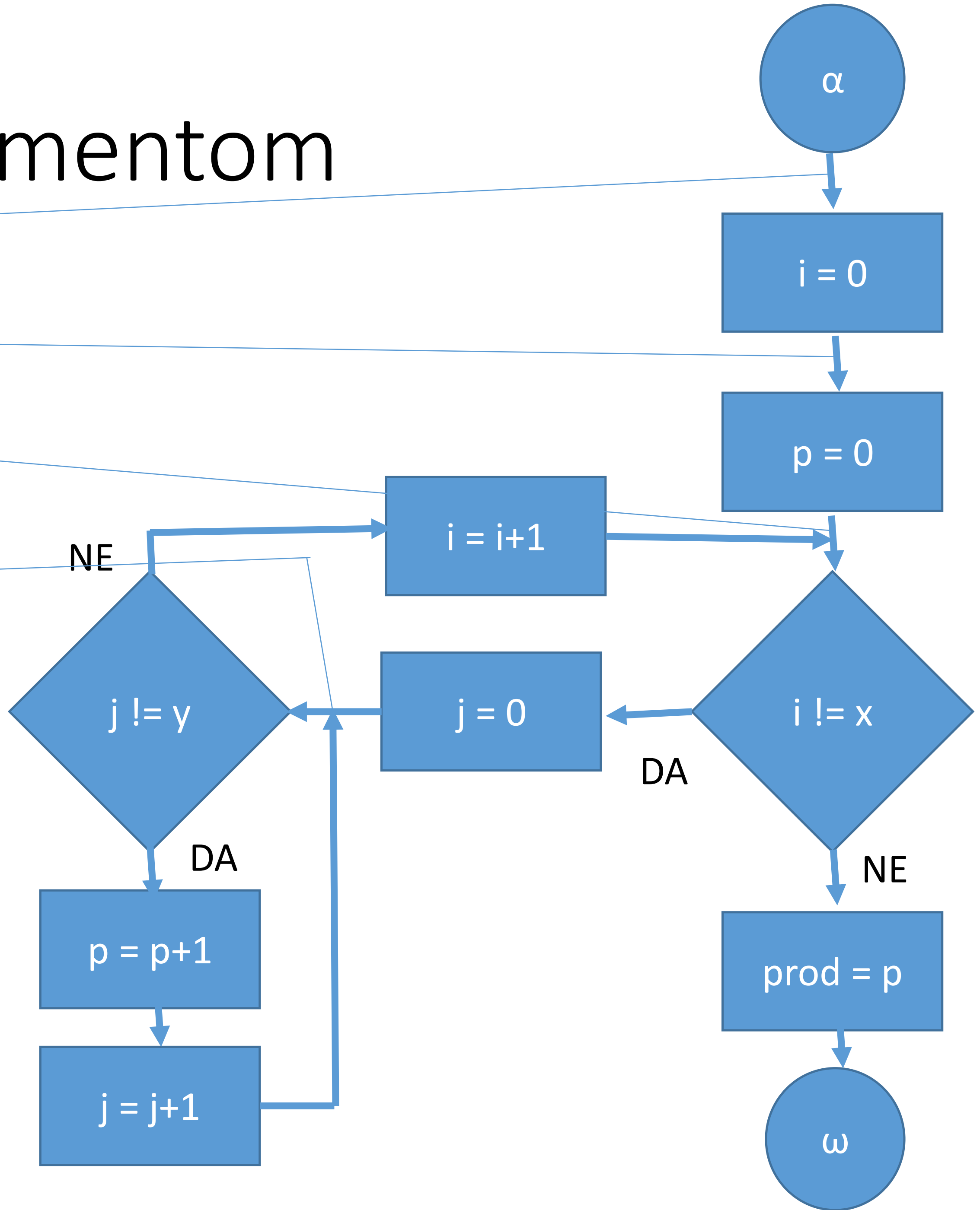
i = 0

p = 0

Torej velja: p = i*y

i != x → i < x in p = i*y

j = 0 → p = i*y + j



Primer: Množenje z inkrementom

x,y >= 0

i = 0

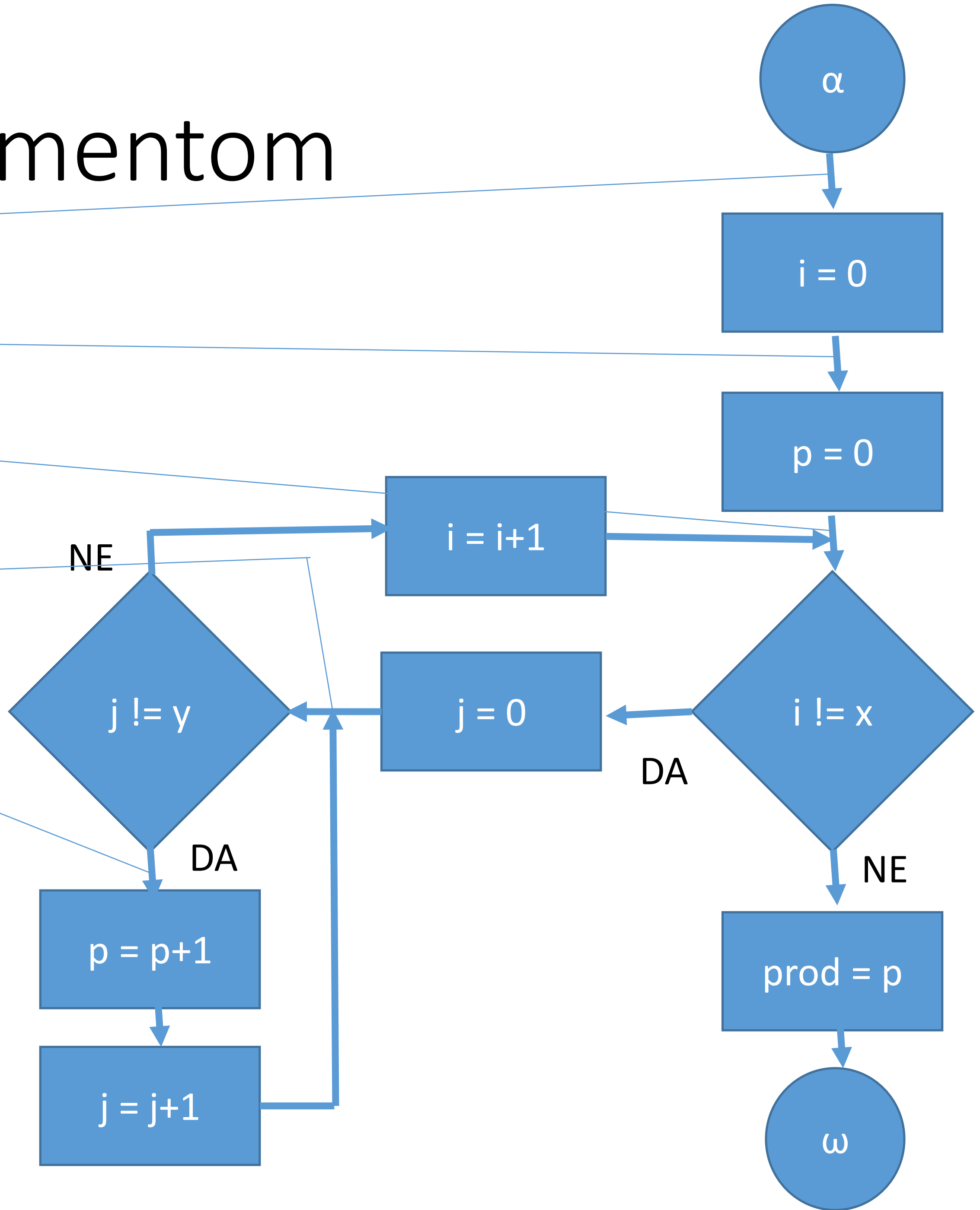
p = 0

Torej velja: p = i*y

i != x → i < x in p = i*y

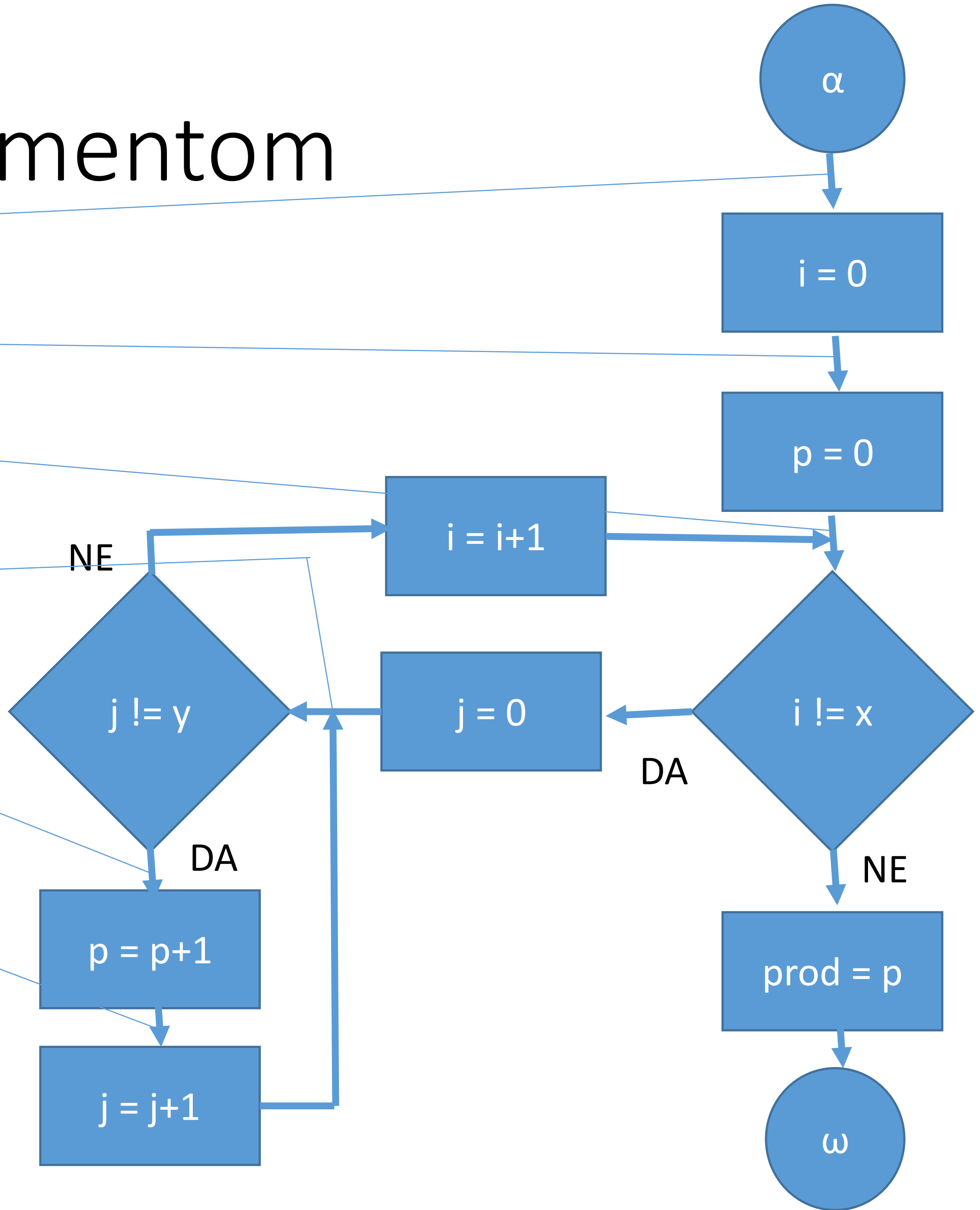
j = 0 → p = i*y + j

j != y → j < y in p = i*y + j



Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1



Primer: Množenje z inkrementom

x,y >= 0

i = 0

p = 0

Torej velja: p = i*y

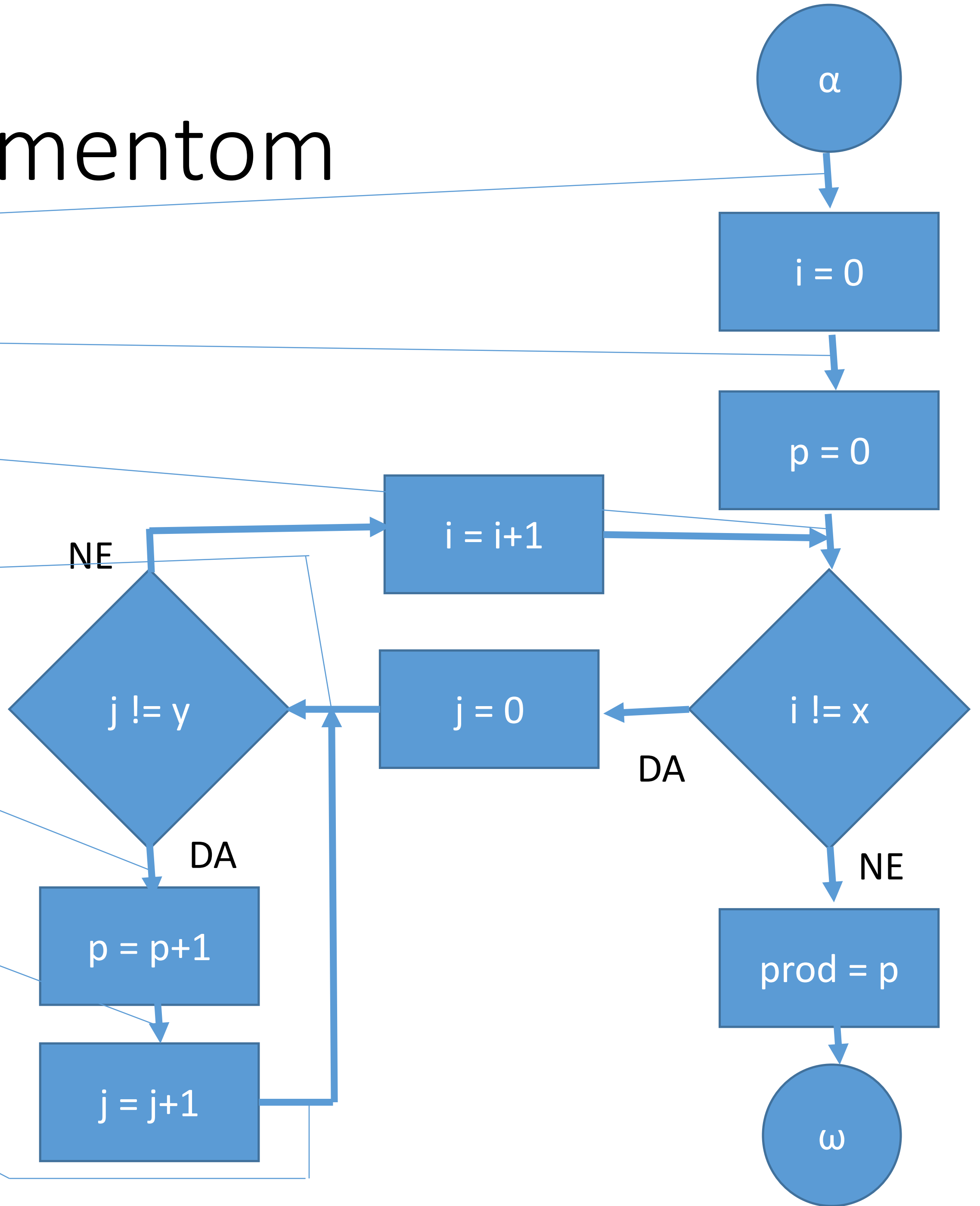
i != x → i < x in p = i*y

j = 0 → p = i*y + j

j != y → j < y in p = i*y + j

p = i*y + j + 1

p = i*y + j



Primer: Množenje z inkrementom

x,y >= 0

i = 0

p = 0

Torej velja: p = i*y

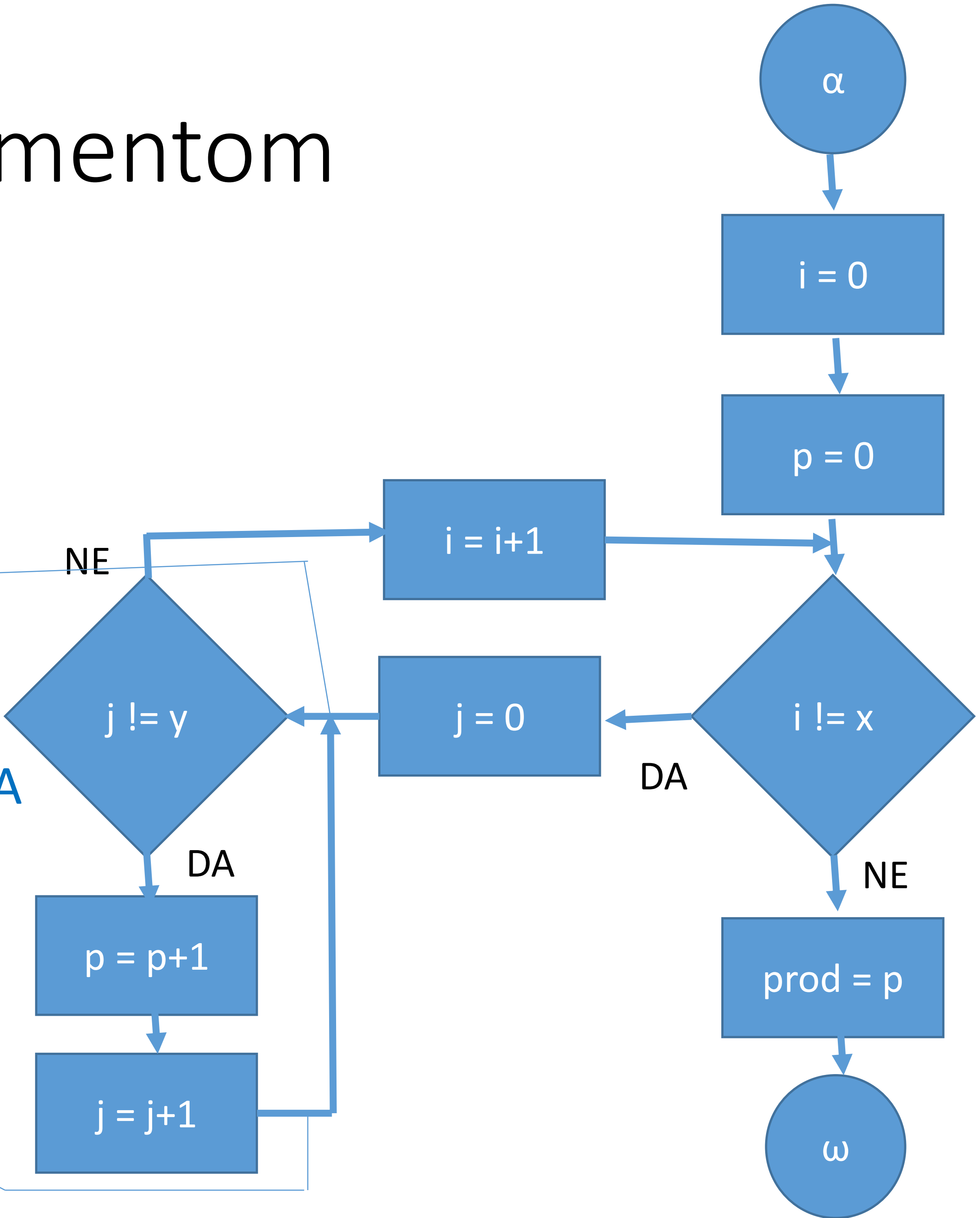
i != x → i < x in p = i*y

j = 0 → p = i*y + j

j != y → j < y in p = i*y + j

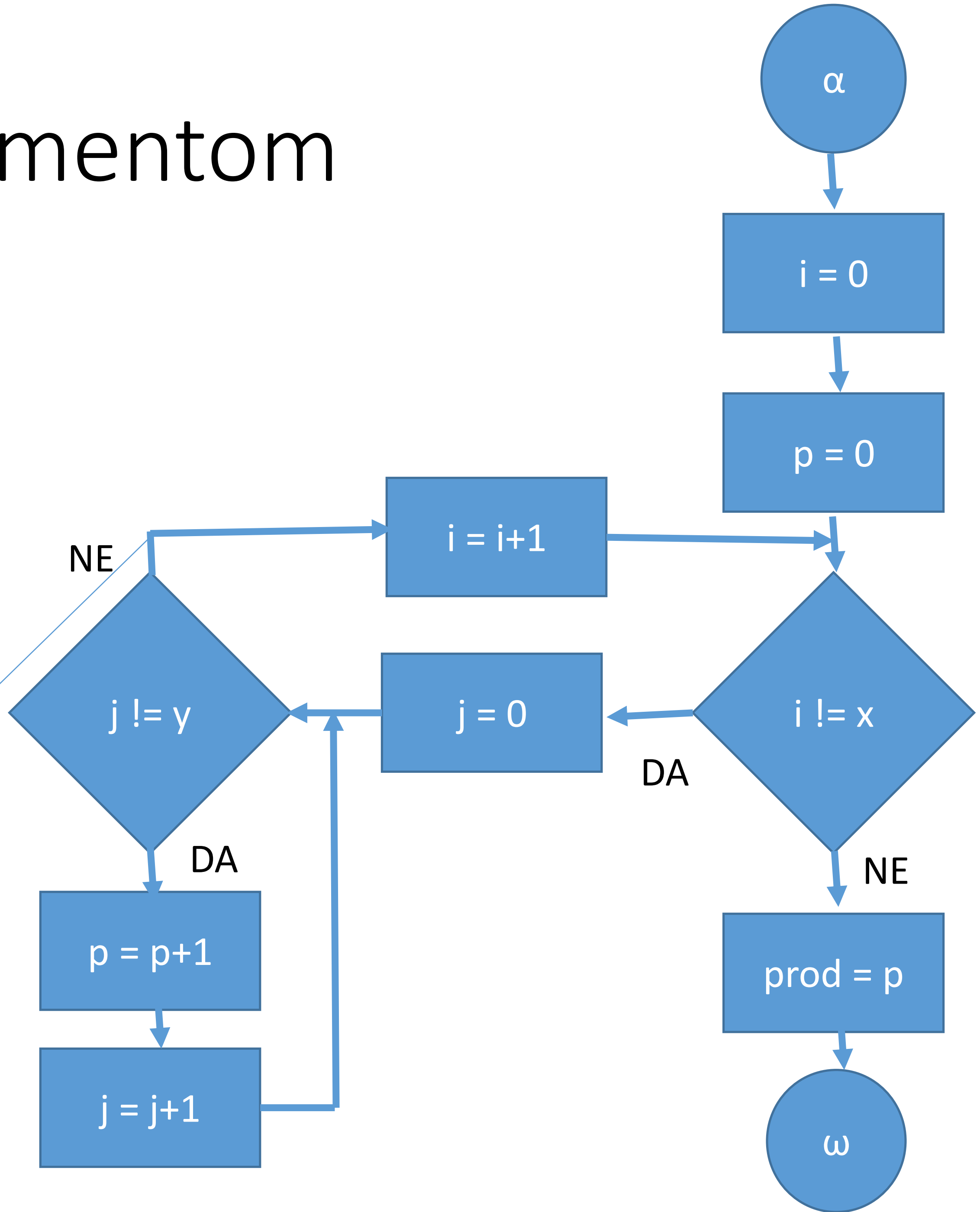
p = i*y + j + 1

p = i*y + j   NOTRANJA ZANČNA INVARIANTA



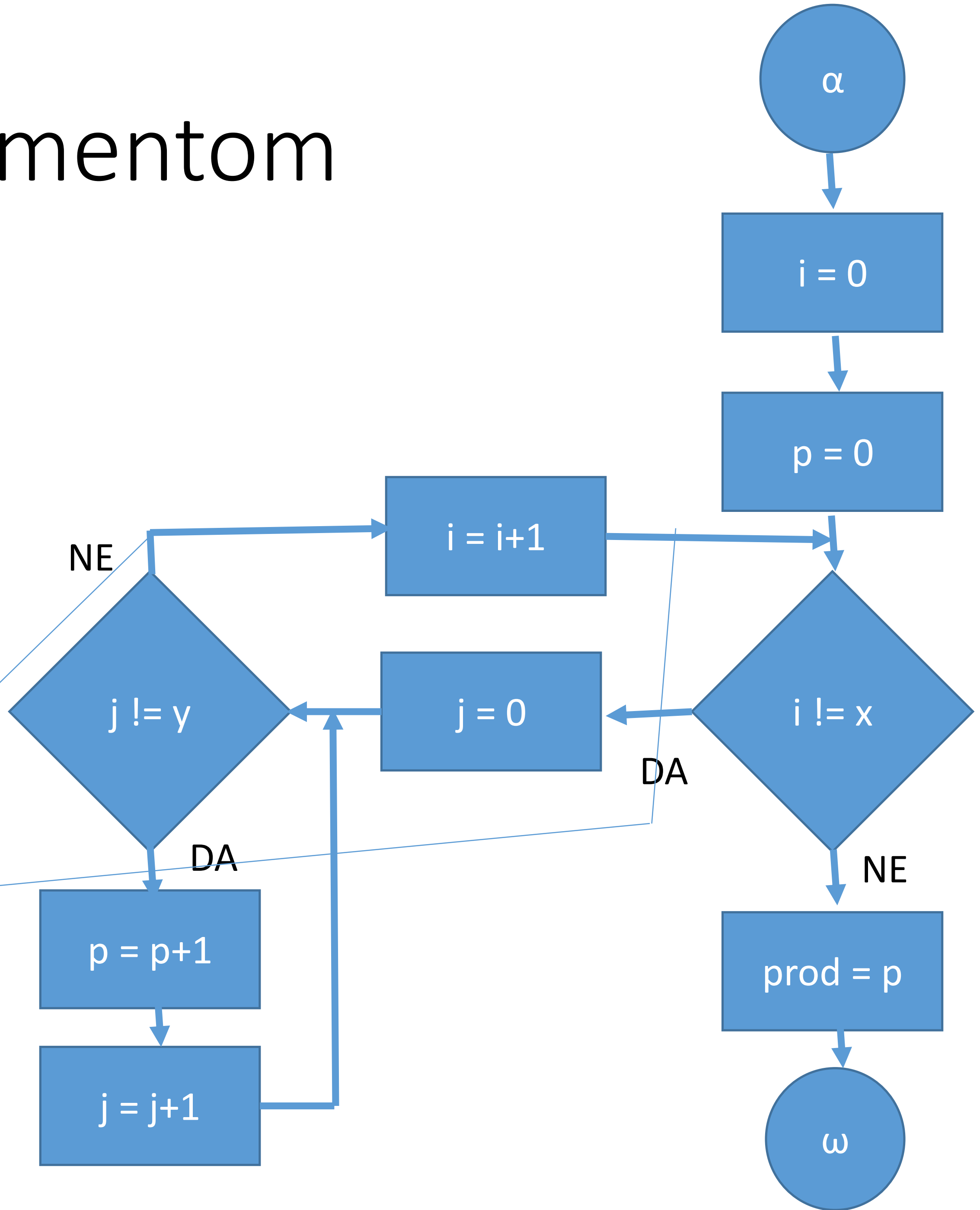
Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y



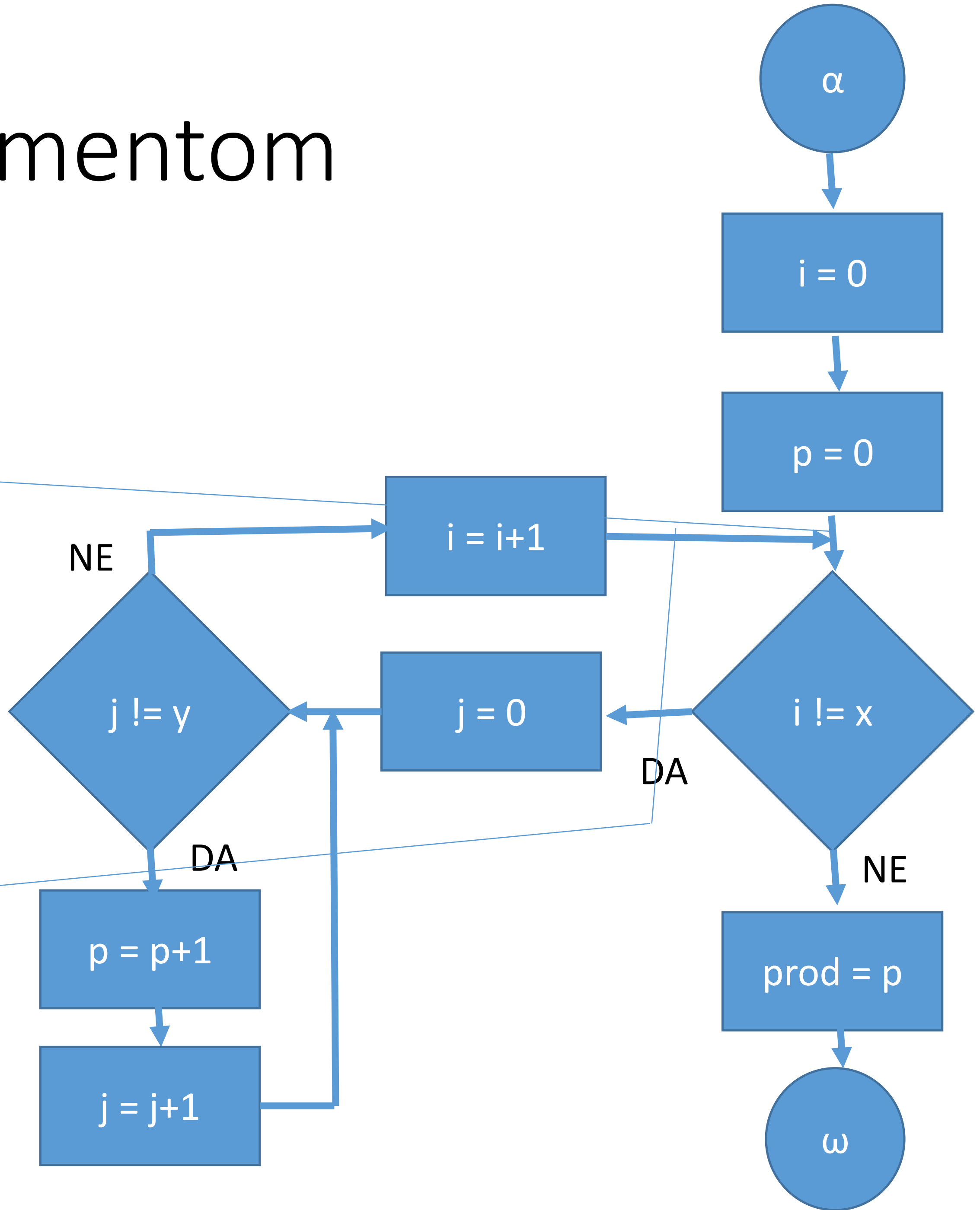
Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: p = i*y



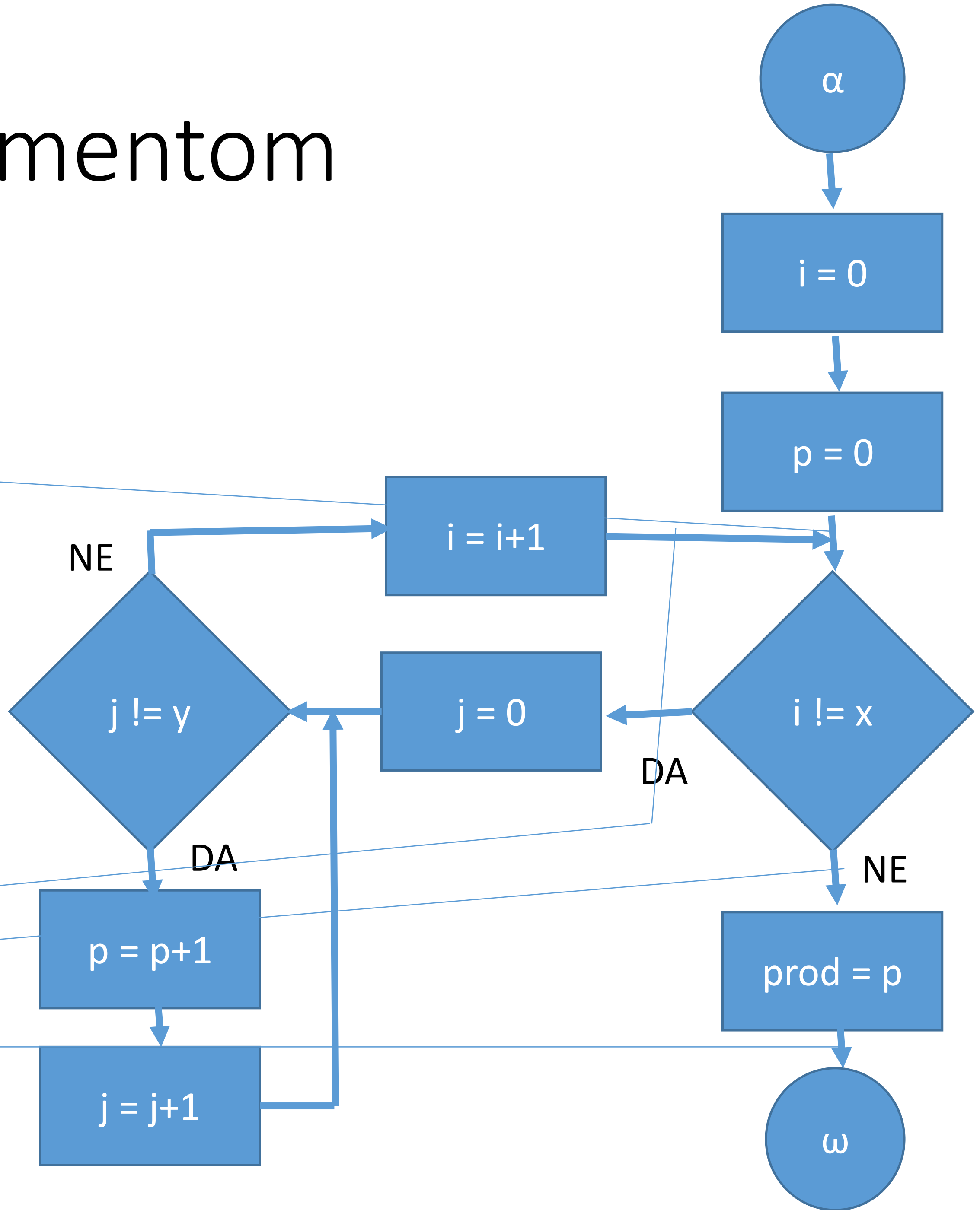
Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: p = i*y
ZUNANJA ZANČNA INVARIANTA



Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: p = i*y
i = x → p = x*y
prod = p → prod = x*y



Primer: Množenje z inkrementom

x,y >= 0

i = 0

p = 0

Torej velja: p = i*y

i != x → i < x in p = i*y

j = 0 → p = i*y + j

j != y → j < y in p = i*y + j

p = i*y + j + 1

p = i*y + j

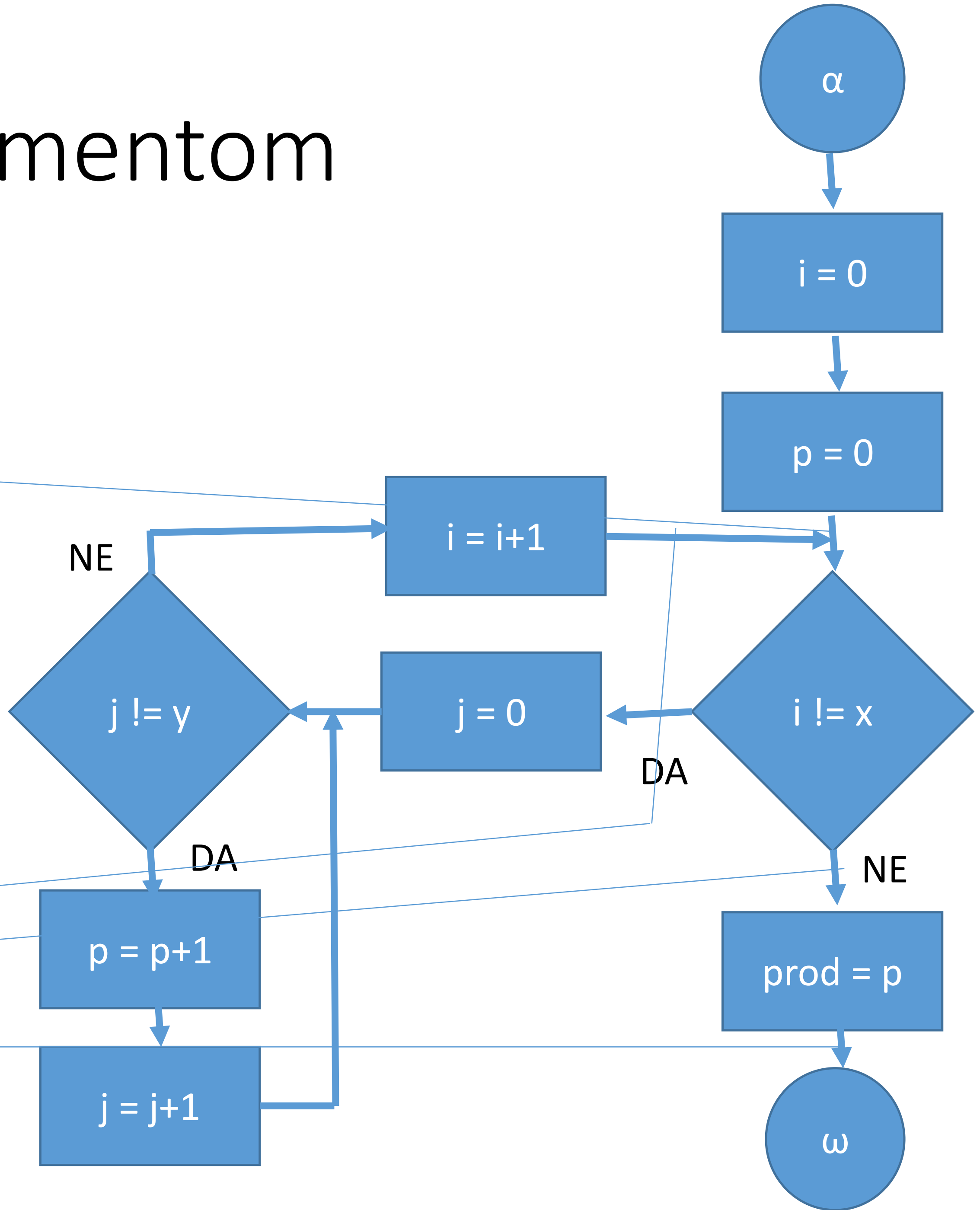
j = y in p = i*y + j → p = i*y+y = (i+1)*y

Torej po stavku i = i + 1 velja: p = i*y

i = x → p = x*y

prod = p → prod = x*y

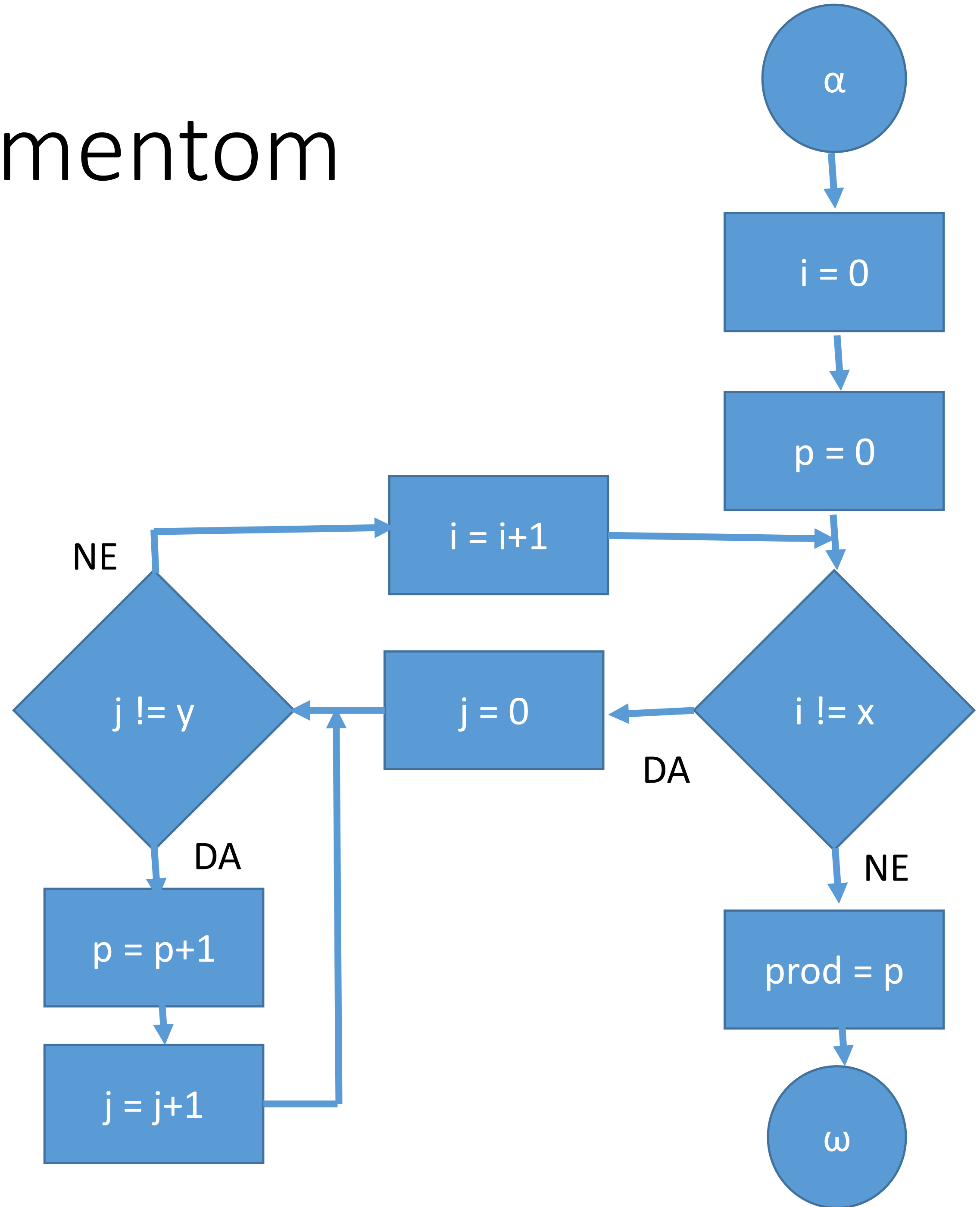
PROGRAM JE PARCIALNO PRAVILEN



Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

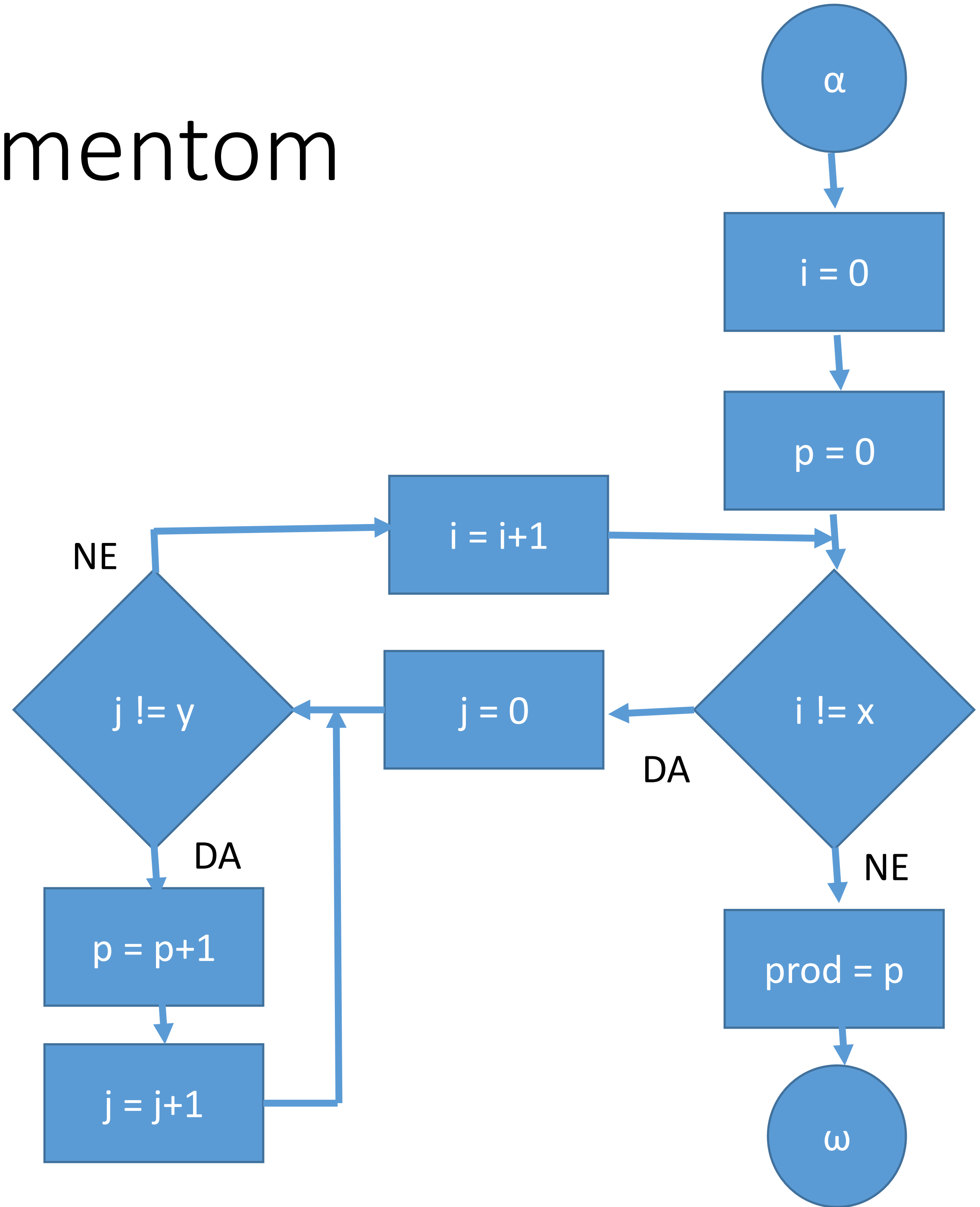
1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1:
3) Zančna invarianta 1:
4) Zančna spremenljivka 2:
5) Zančna invarianta 2:



Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

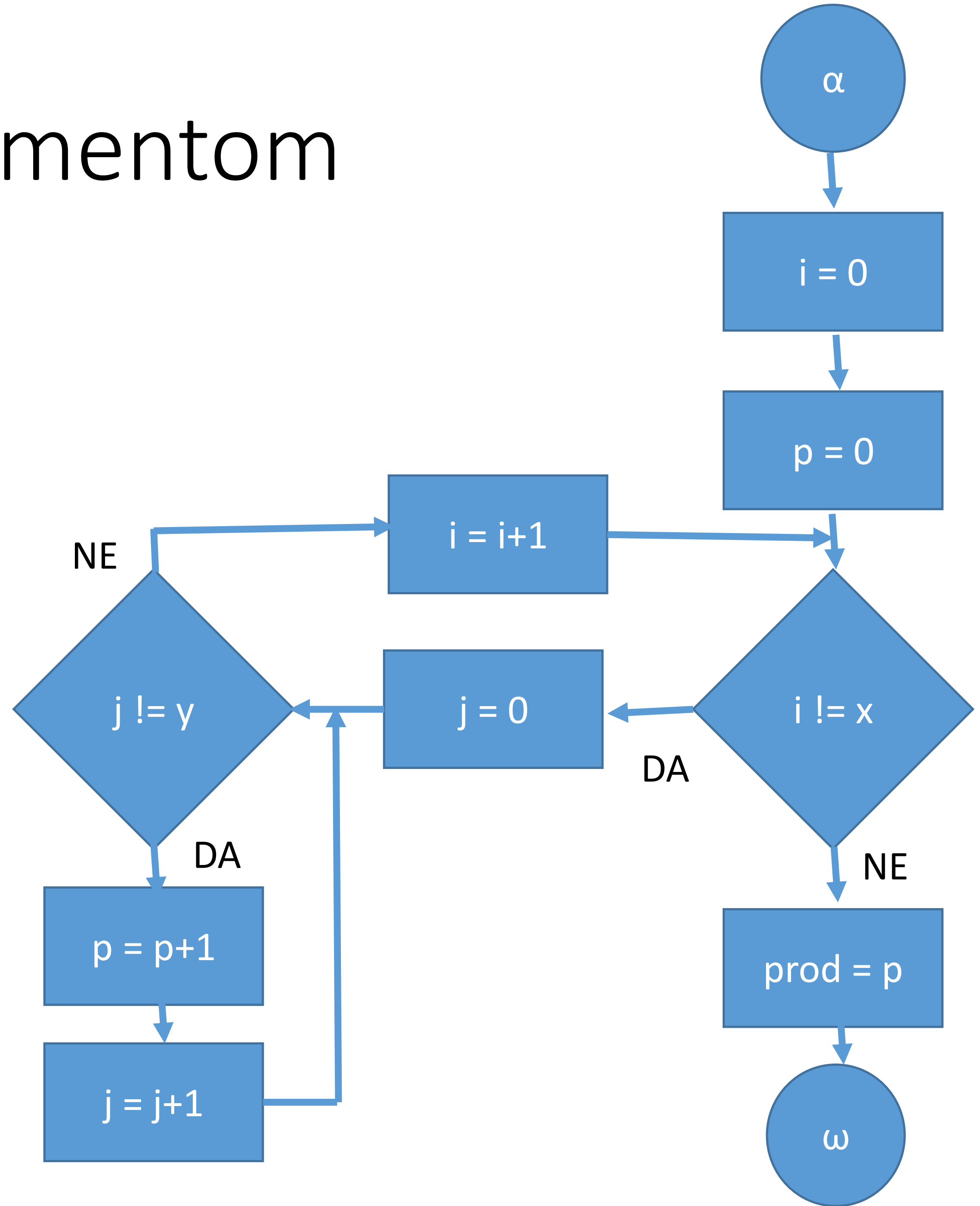
1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: $l1 = y\text{-}j$
3) Zančna invarianta 1: $y - j \in N$
4) Zančna spremenljivka 2:
5) Zančna invarianta 2:



Primer: Množenje z inkrementom

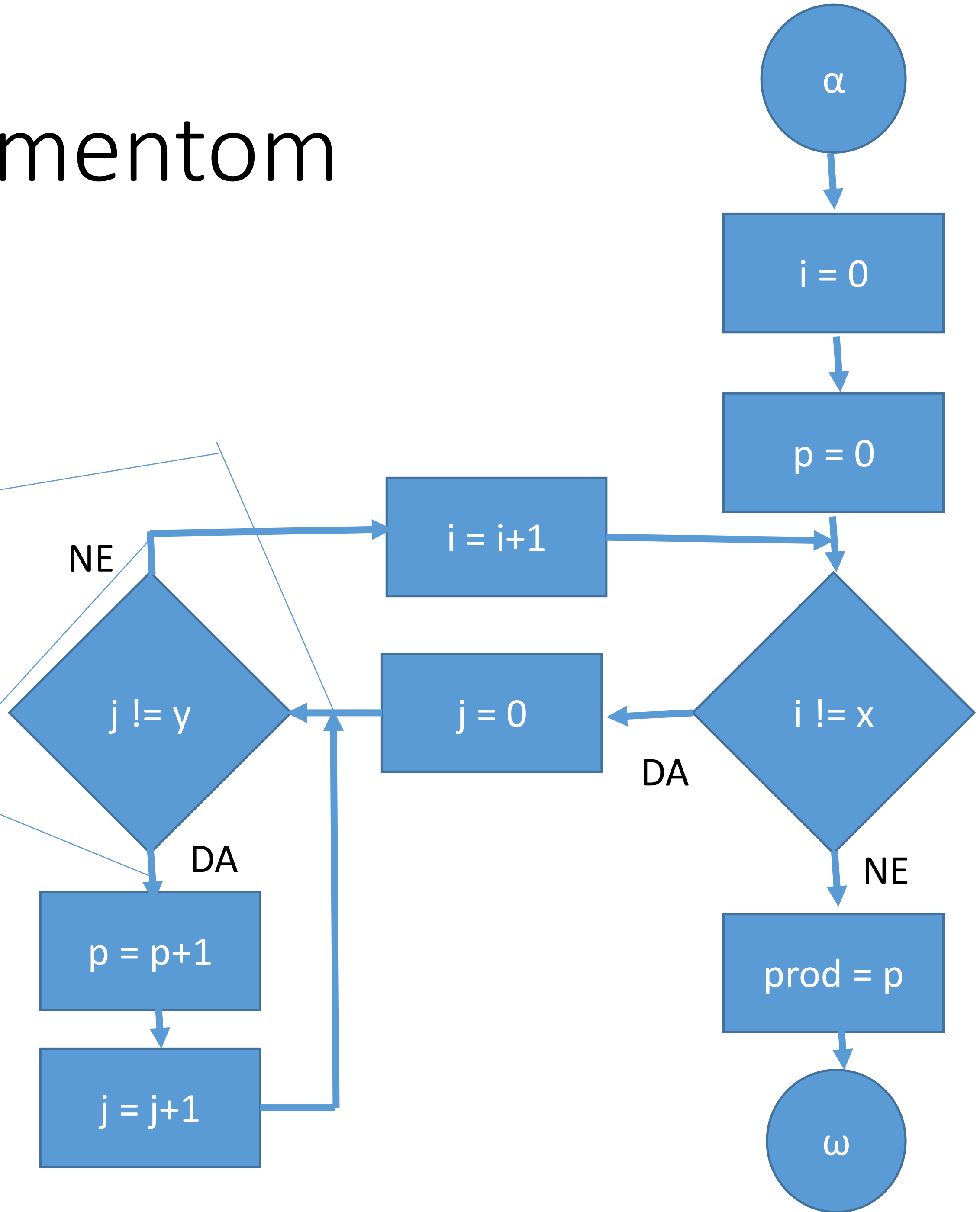
Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: $l1 = y-j$
3) Zančna invarianta 1: $y - j \in N$
4) Zančna spremenljivka 2: $l2 = x-i$
5) Zančna invarianta 2: $x - i \in N$



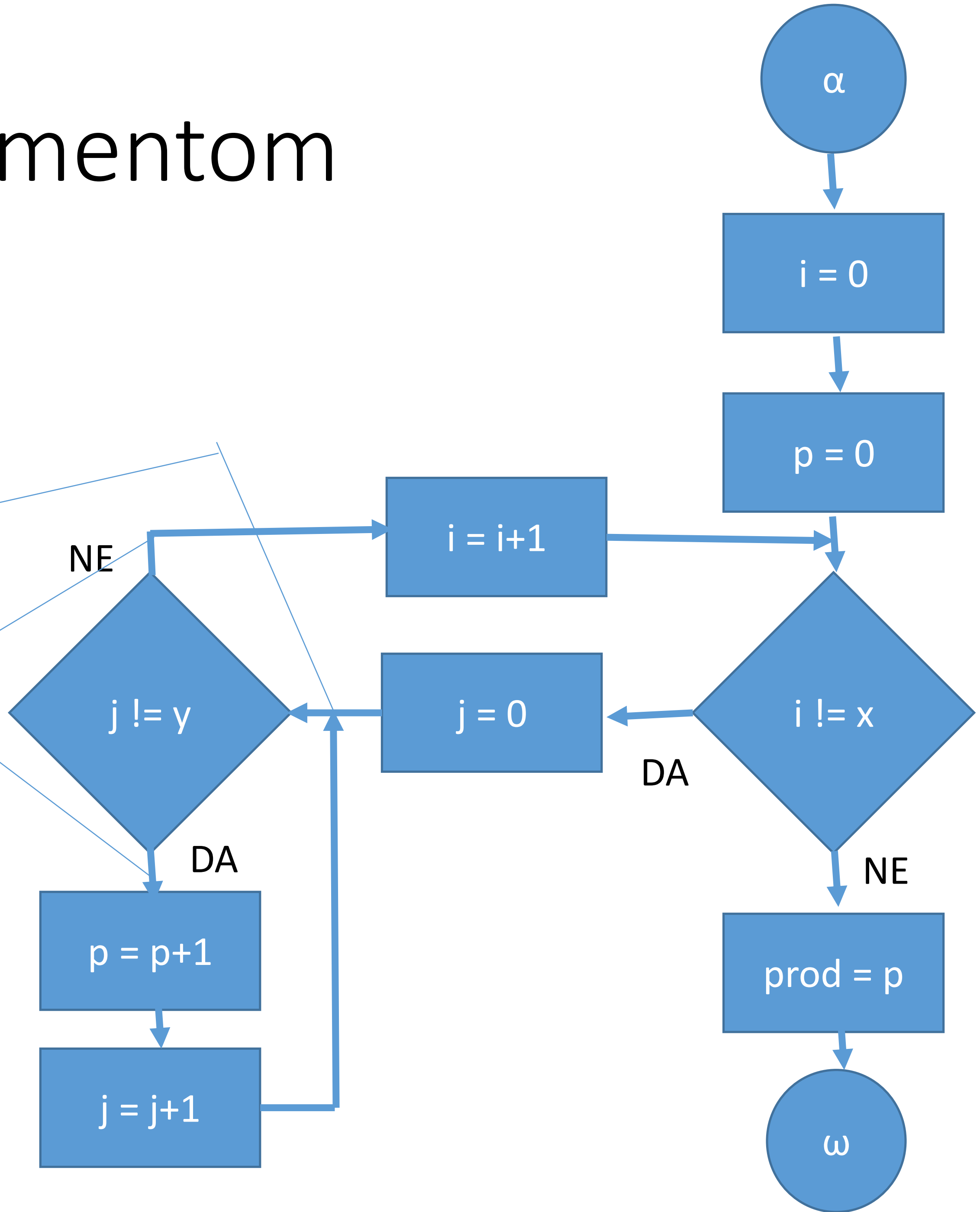
Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: p = i*y
i = x → p = x*y
prod = p → prod = x*y
PROGRAM JE PARCIALNO PRAVILEN



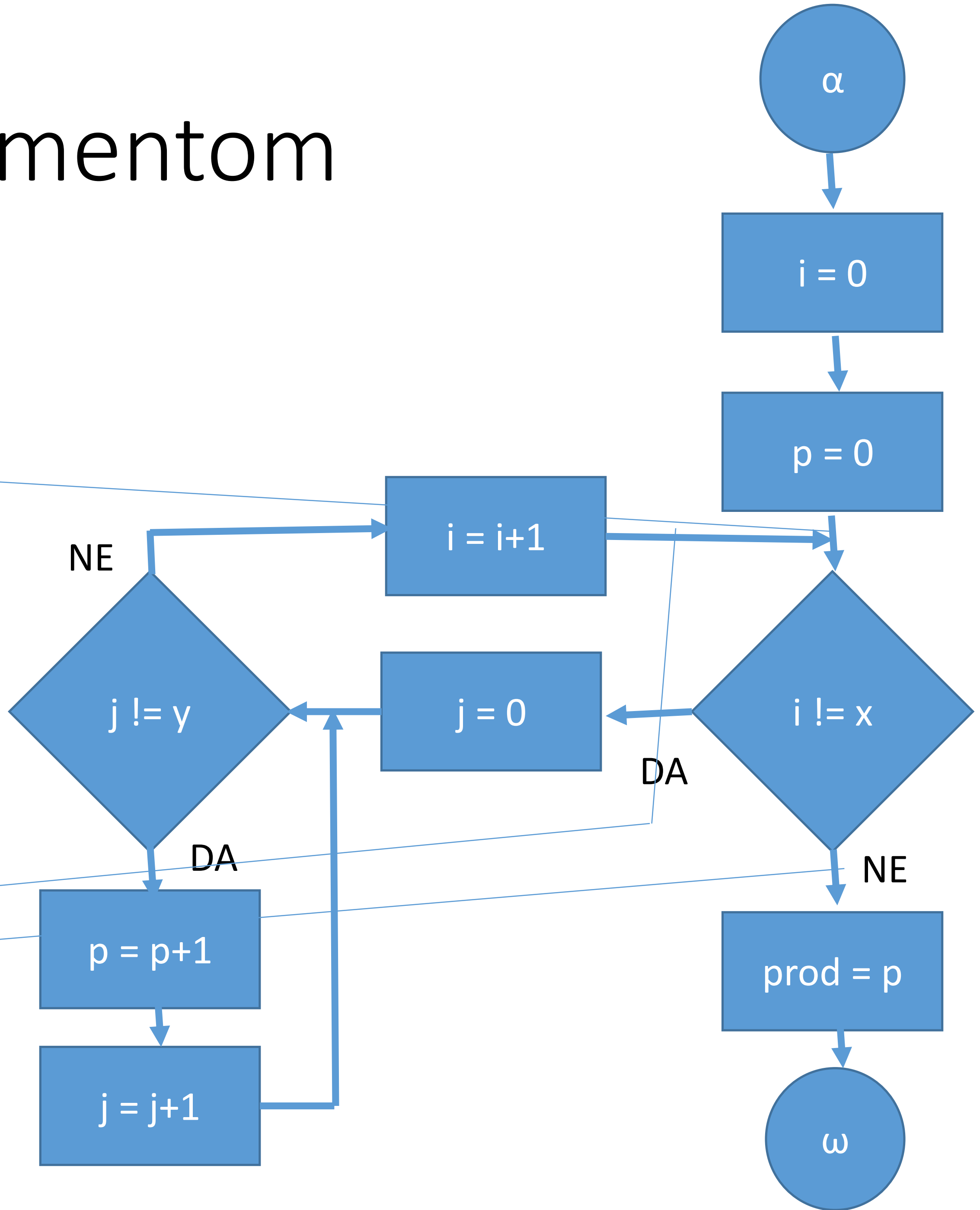
Primer: Množenje z inkrementom

$x, y >= 0$
$i = 0$
$p = 0$
Torej velja: $p = i*y$
$i \,!= x \rightarrow i < x$ in $p = i*y$
$j = 0 \rightarrow p = i*y + j \rightarrow y - j \in N$
$j \,!= y \rightarrow j < y$ in $p = i*y + j \rightarrow y - j \in N$
$p = i*y + j + 1$
$p = i*y + j$
$j = y$ in $p = i*y + j \rightarrow y - j \in N$
Torej po stavku $i = i + 1$ velja: $p = i*y$
$i = x \rightarrow p = x*y$
$prod = p \rightarrow prod = x*y$
PROGRAM JE PARCIALNO PRAVILEN



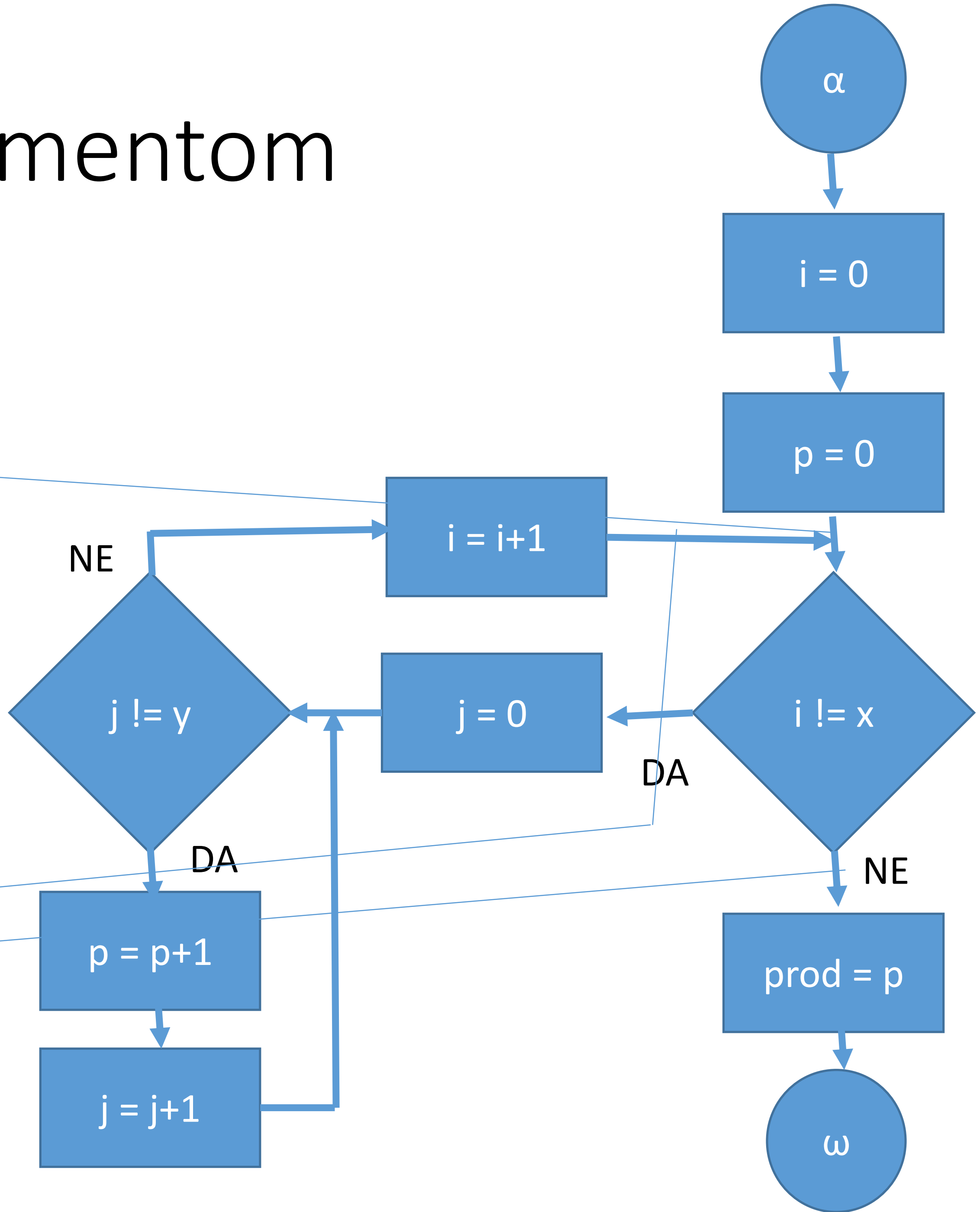
Primer: Množenje z inkrementom

x,y >= 0
i = 0
p = 0
Torej velja: p = i*y
i != x → i < x in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: p = i*y
i = x → p = x*y
prod = p → prod = x*y
PROGRAM JE PARCIALNO PRAVILEN



Primer: Množenje z inkrementom

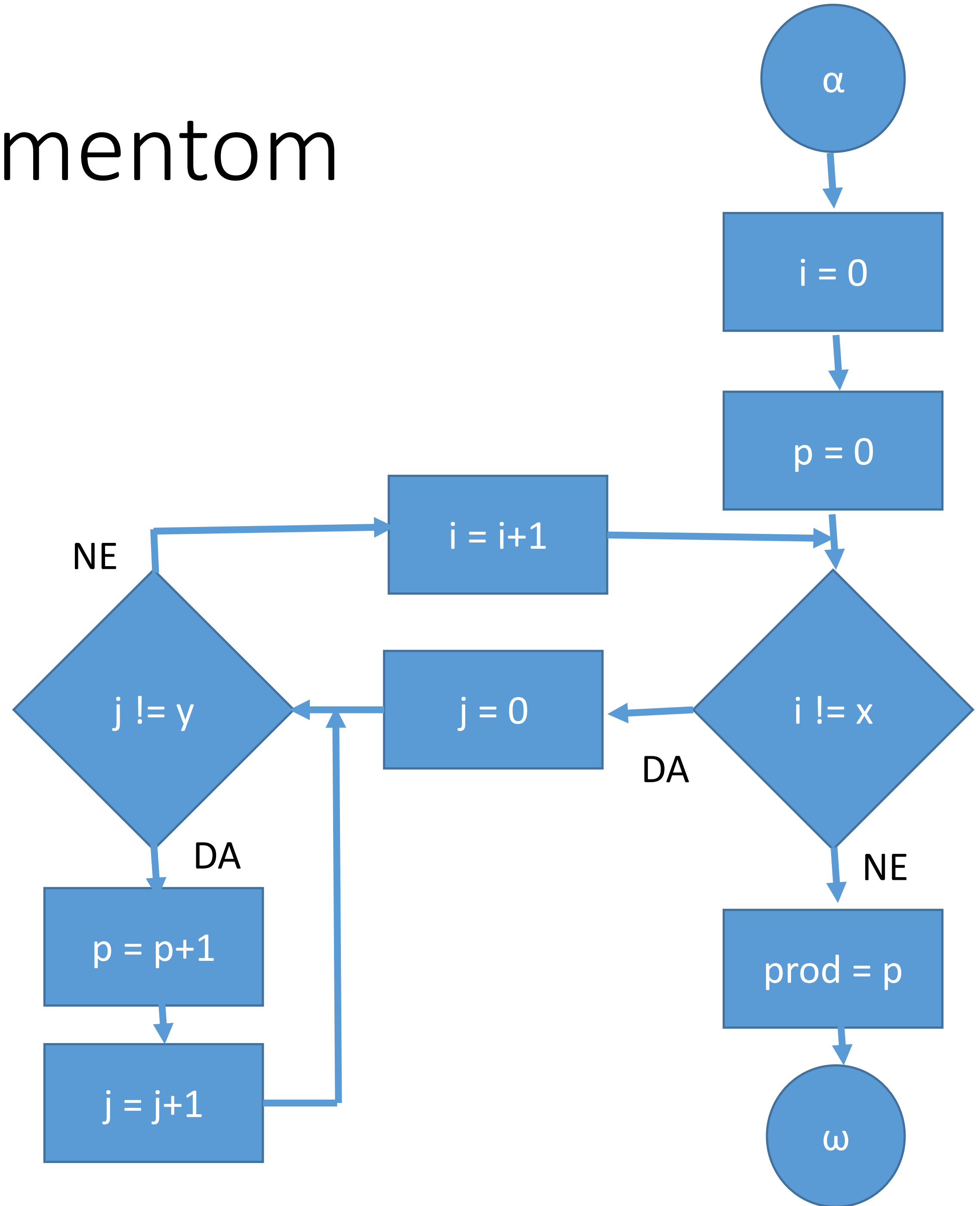
x,y >= 0
i = 0
p = 0
Torej velja: p = i*y → x-i ∊ N
i != x → **i < x** in p = i*y
j = 0 → p = i*y + j
j != y → j < y in p = i*y + j
p = i*y + j + 1
p = i*y + j
j = y in p = i*y + j → p = i*y+y = (i+1)*y
Torej po stavku i = i + 1 velja: x-i ∊ N
i = x → p = x*y → x-i ∊ N
prod = p → prod = x*y
PROGRAM JE PARCIALNO PRAVILEN



Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

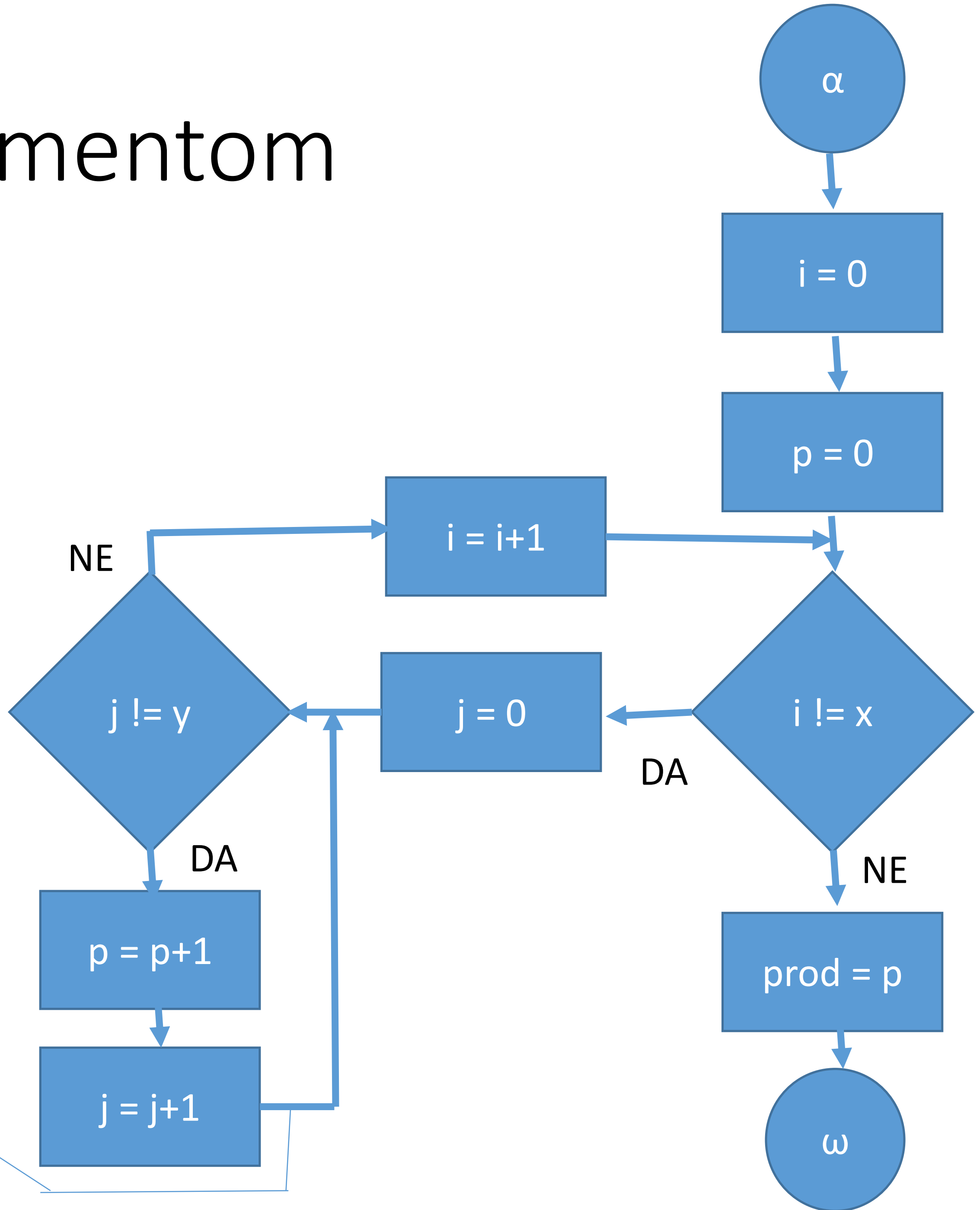
1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: *l1 = y-j*
3) **Zančna invarianta 1:** $y - j \in N$
4) Zančna spremenljivka 2: *l2 = x-i*
5) **Zančna invarianta 2:** $x - i \in N$
6) Vrednosti l1 in l2 se zmanjšujeta



Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

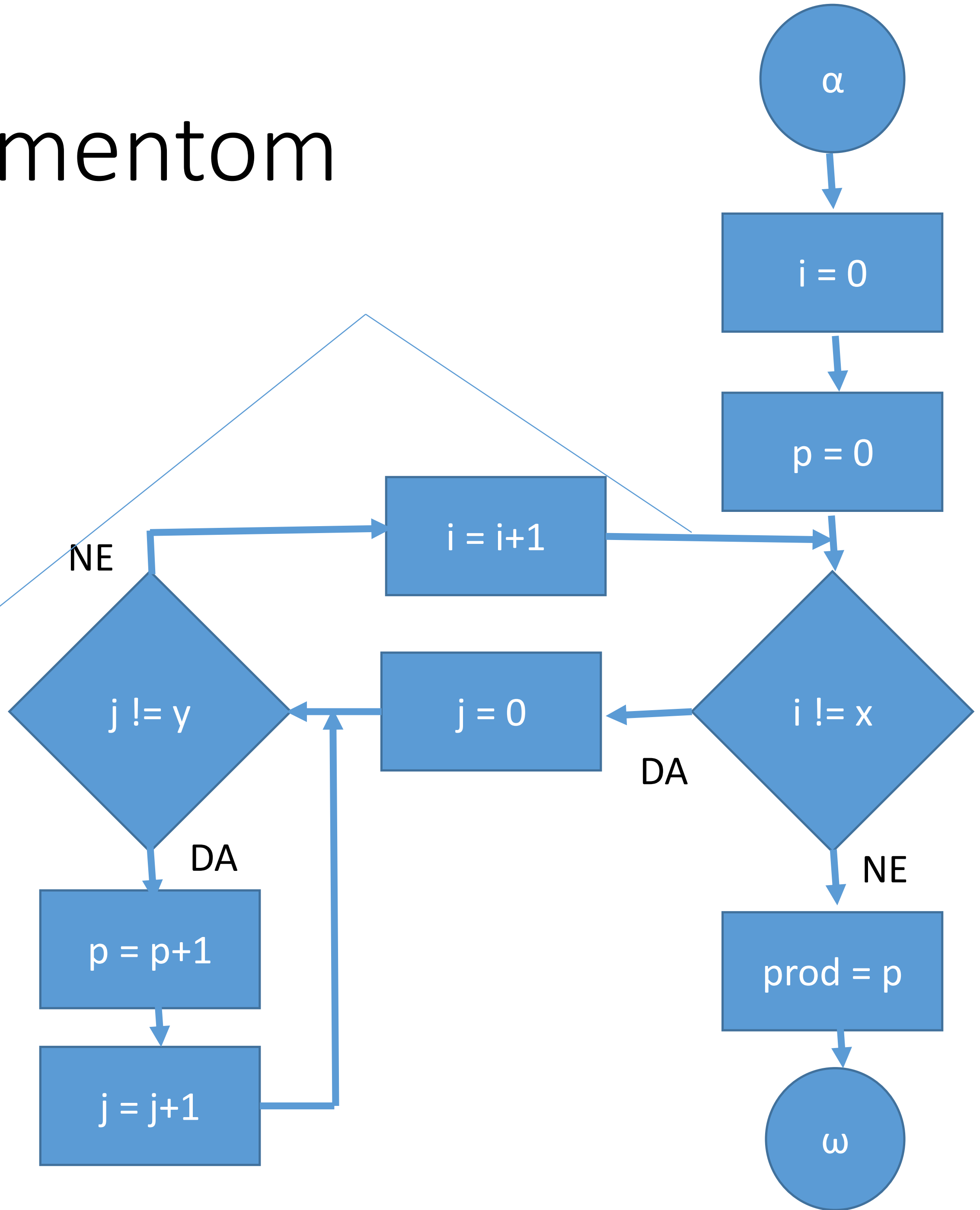
1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: $l1 = y-j$
3) Zančna invarianta 1: $y - j \in N$
4) Zančna spremenljivka 2: $l2 = x-i$
5) Zančna invarianta 2: $x - i \in N$
6) Vrednosti *l1* in *l2* se zmanjšujeta
   - $ll1 = y - j - 1 < l1$



Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: $l1 = y\text{-}j$
3) Zančna invarianta 1: $y - j \in N$
4) Zančna spremenljivka 2: $l2 = x\text{-}i$
5) Zančna invarianta 2: $x - i \in N$
6) Vrednosti *l1* in *l2* se zmanjšujeta
   - $ll1 = y - j - 1 < l1$
   - $ll2 = x - i - 1 < l2$



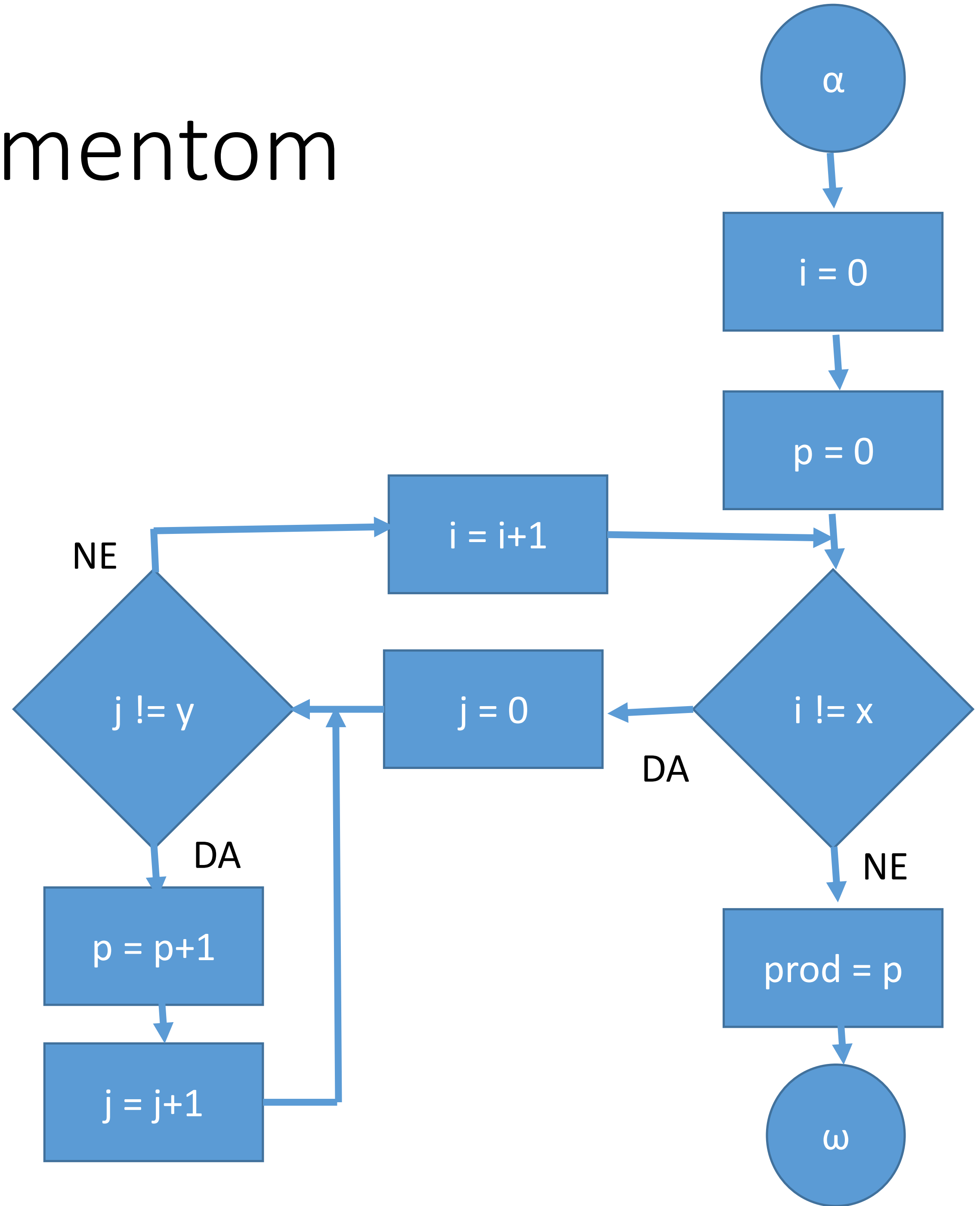
Primer: Množenje z inkrementom

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka 1: $l1 = y-j$
3) Zančna invarianta 1: $y - j \in N$
4) Zančna spremenljivka 2: $l2 = x-i$
5) Zančna invarianta 2: $x - i \in N$
6) Vrednosti *l1* in *l2* se zmanjšujeta
   - $ll1 = y - j - 1 < l1$
   - $ll2 = x - i - 1 < l2$

PROGRAM JE TOTALNO PRAVILEN:

- Je parcialno pravilen
- Se vedno ustavi




OH, COME ON! THE STICK IS RIGHT THERE!
GPS
MARK PARISI
offthemark.com
MarkParisi@aol.com 1-3
©2017 Mark Parisi Dist. by Andrews McMeel Synd.

Primer: Iskanje maksimalnega števila

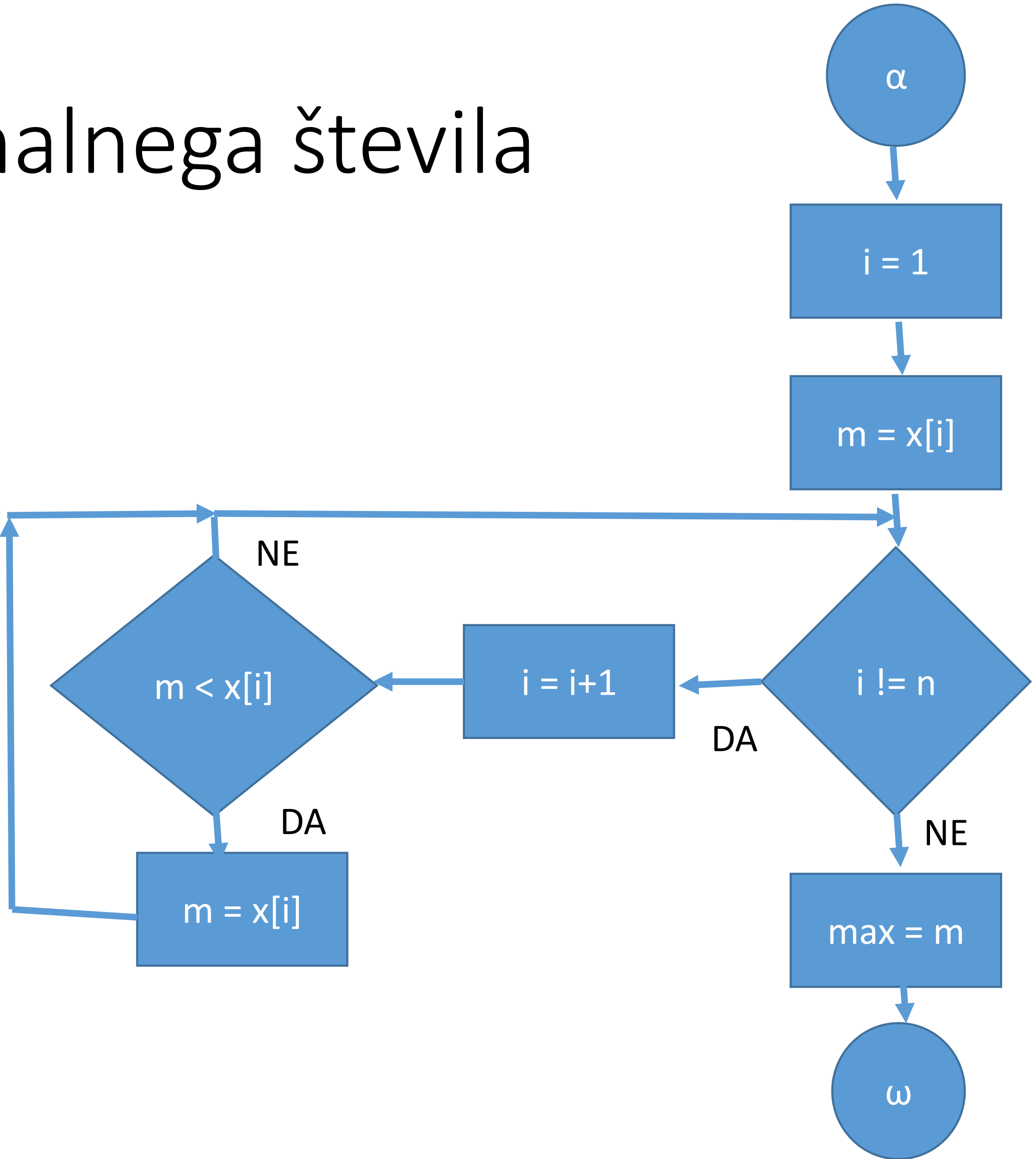
- Začetni pogoj: $\phi(n, x_1, \ldots, x_n) = (n \in \mathrm{N}) \wedge (x_i \in \mathrm{R},\ i = 1..n)$
- Zaključni pogoj: $\psi(max, n, x_1, \ldots, x_n) = \left(max = \max_{i=1}^{n} x_i \right)$

Primer: Iskanje maksimalnega števila

```
static public double max(double x[], int n) {
   int i ;
   double m ;
   // fi(x,n) = (n>0) & (x[i] pripada R, i=1..n)
   i=1 ;
   m=x[i] ;
   while (i != n) {
      i++ ;
      if (x[i] > m)
         m = x[i] ;
   } // while
   return m ;
   // psi(x, n, max) = (max = max x[i] )
} // max
```

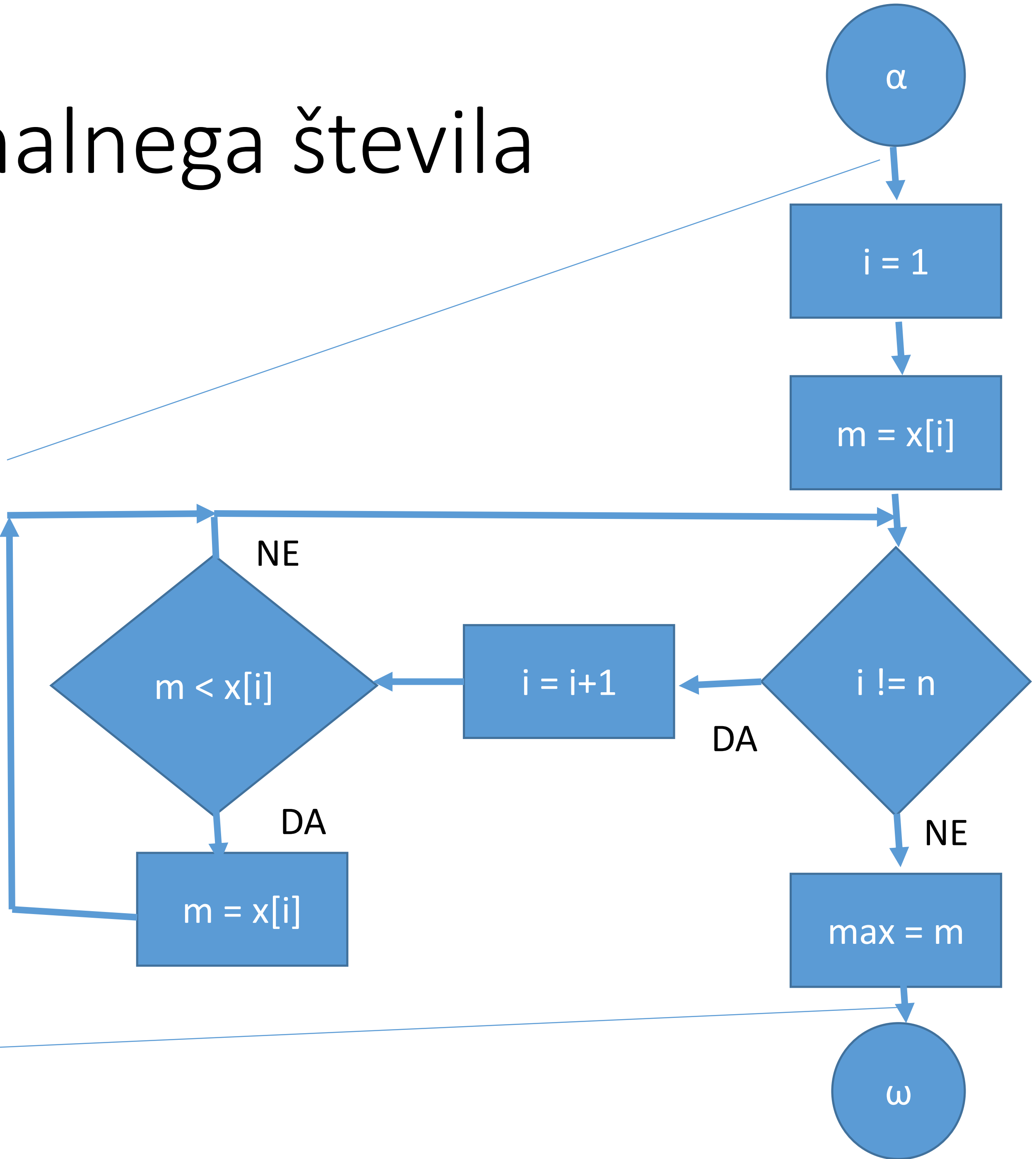
Primer: Iskanje maksimalnega števila

```
static public double max(double x[], int n) {
   int i ;
   double m ;
   // fi(x,n) = (n>0) & (x[i] pripada R, i=1..n)
   i=1 ;
   m=x[i] ;
   while (i != n) {
      i++ ;
      if (x[i] > m)
         m = x[i] ;
   } // while
   return m ;
   // psi(x, n, max) = (max = max x[i] )
} // max
```



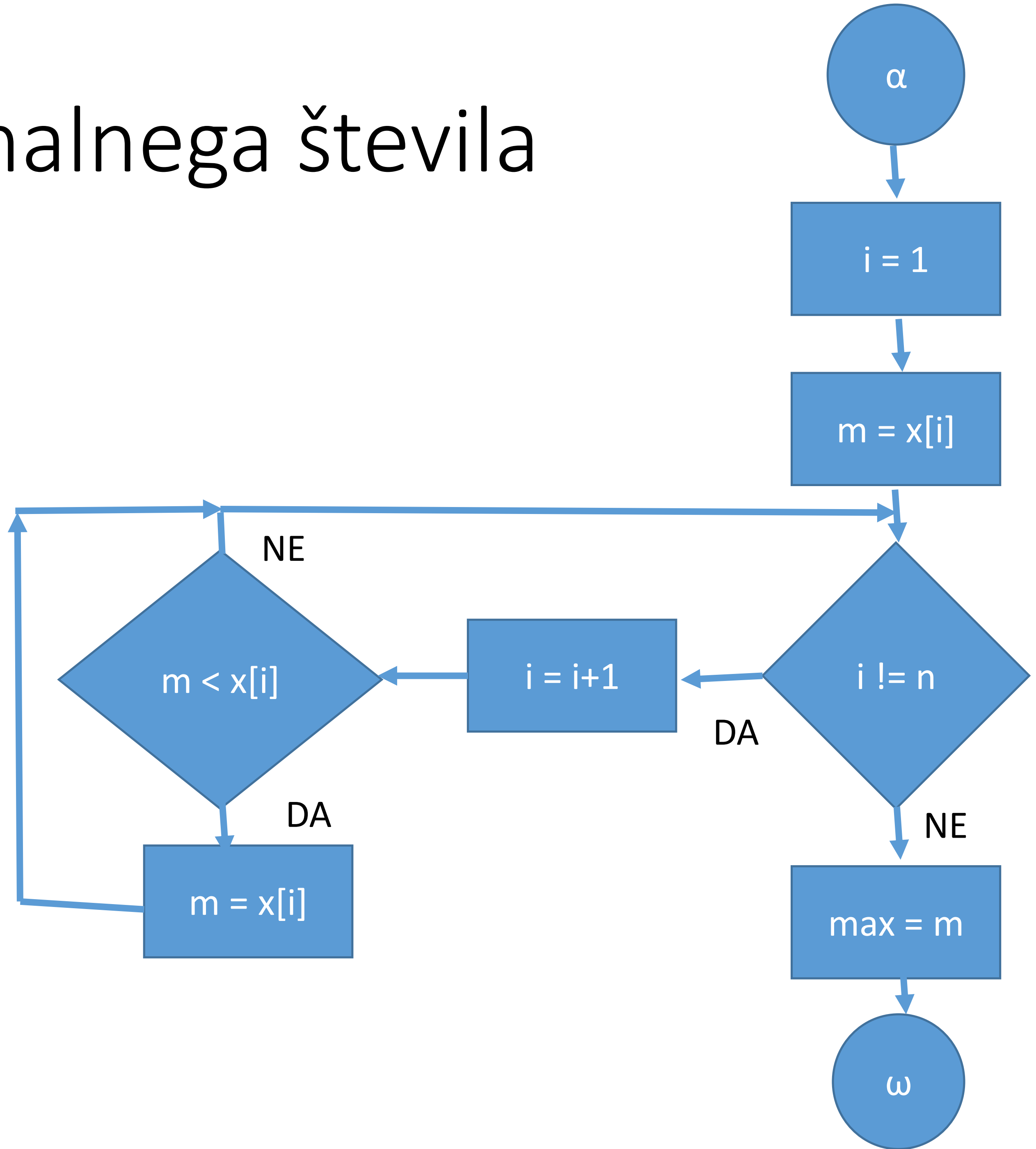
Primer: Iskanje maksimalnega števila

```
static public double max(double x[], int n) {
   int i ;
   double m ;
   // fi(x,n) = (n>0) & (x[i] pripada R, i=1..n)
   i=1 ;
   m=x[i] ;
   while (i != n) {
      i++ ;
      if (x[i] > m)
         m = x[i] ;
   } // while
   return m ;
   // psi(x, n, max) = (max = max x[i] )
} // max
```



Primer: Iskanje maksimalnega števila

Kaj je primerna zančna invarianta?

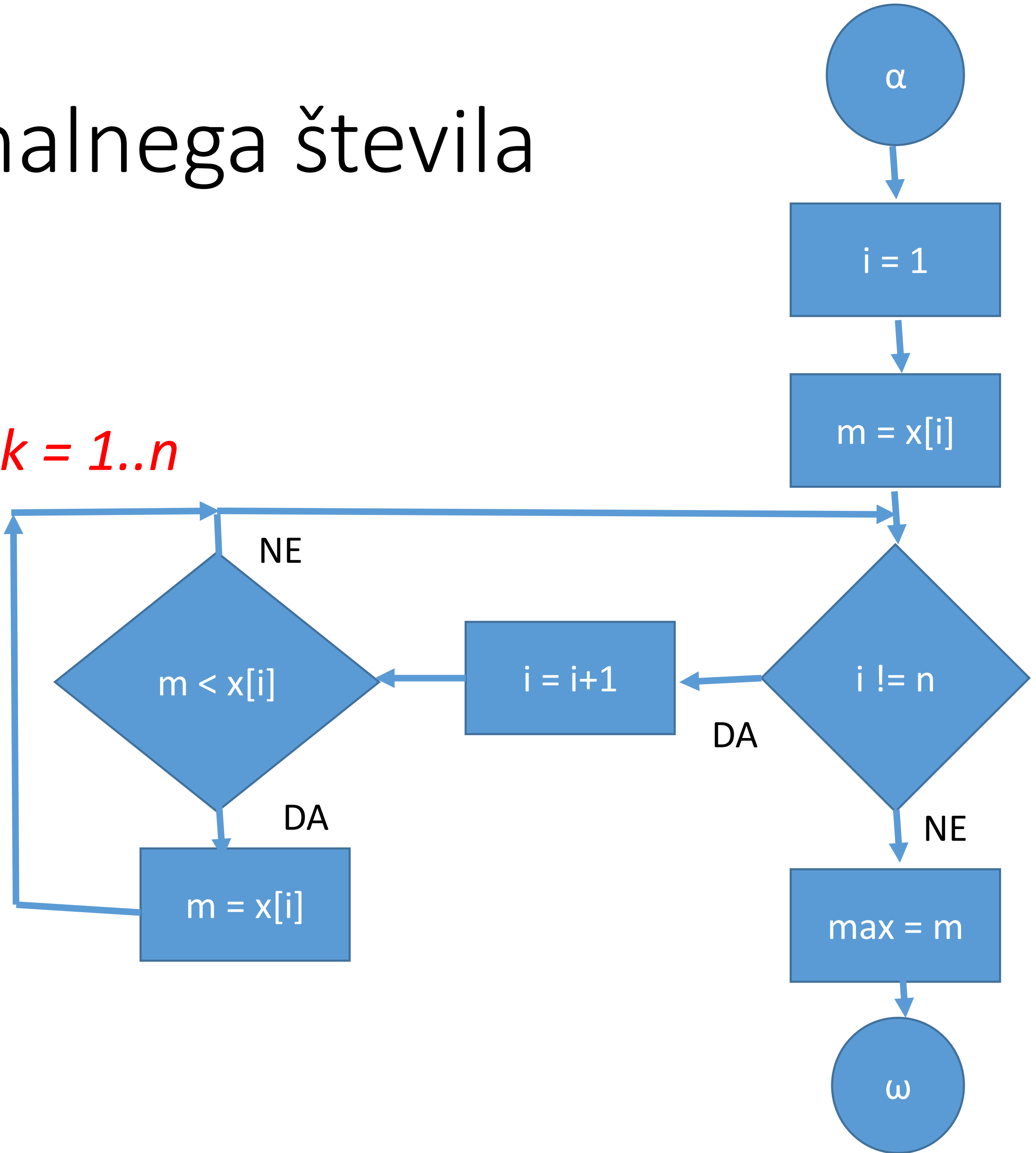


Primer: Iskanje maksimalnega števila

Kaj je primerna zančna invarianta?

Dokazati moramo, da je *max* = **max** *x[k], k = 1..n*

Torej pred stavkom *max = m* mora veljati *m* = **max** *x[k], k=1..n*



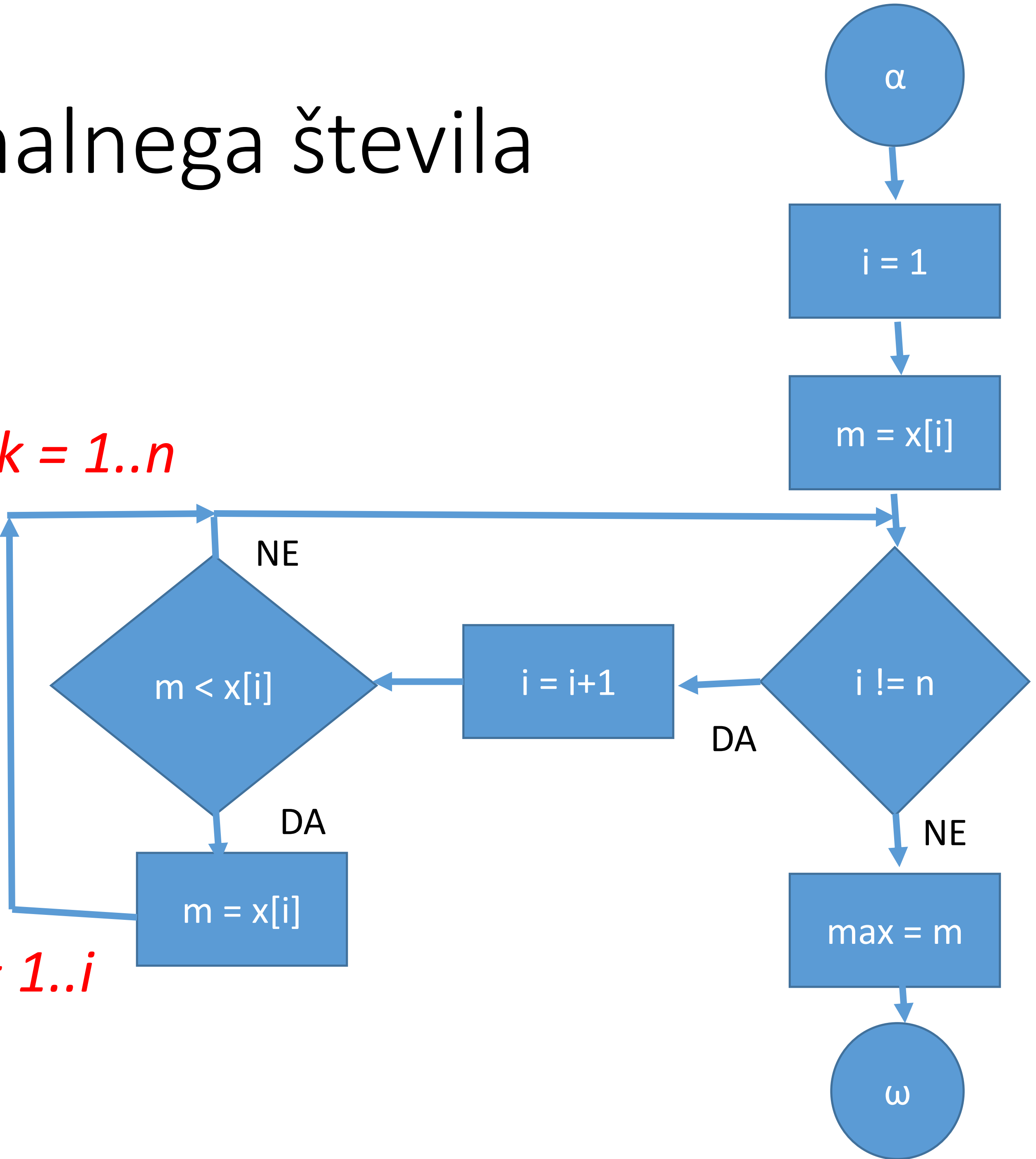
Primer: Iskanje maksimalnega števila

Kaj je primerna zančna invarianta?

Dokazati moramo, da je $max = \mathbf{max}\ x[k],\ k = 1..n$

Torej pred stavkom *max = m* mora veljati $m = \mathbf{max}\ x[k],\ k=1..n$

Ker v tej točki ne velja več $i\ != n$, mora torej veljati $i = n$. Torej je primerna zančna invarianta lahko $m = \mathbf{max}\ x[k],\ k = 1..i$

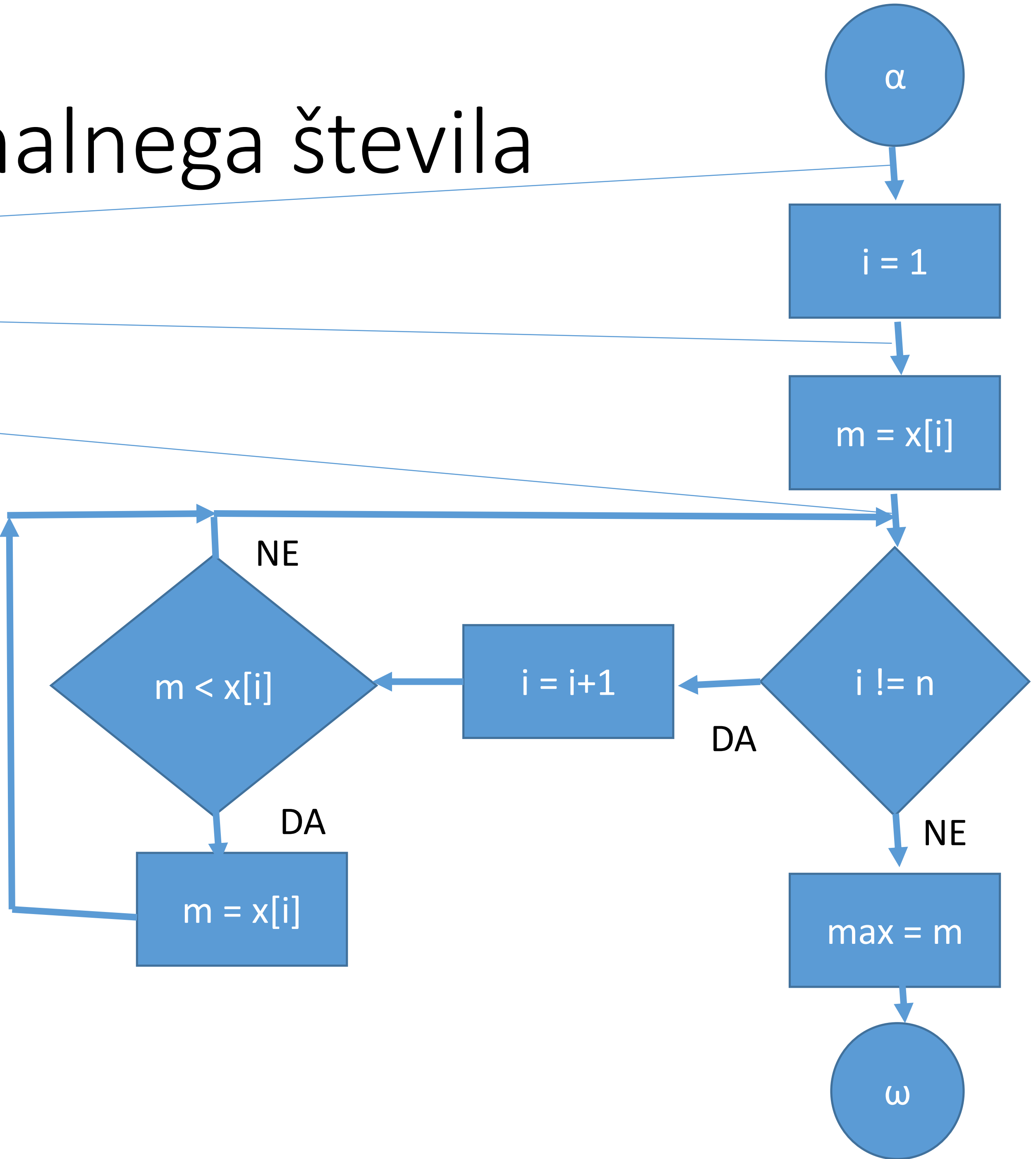


Primer: Iskanje maksimalnega števila

n > 0, x[i] ϵ R, i = 1..n

i = 1

m = x[i]



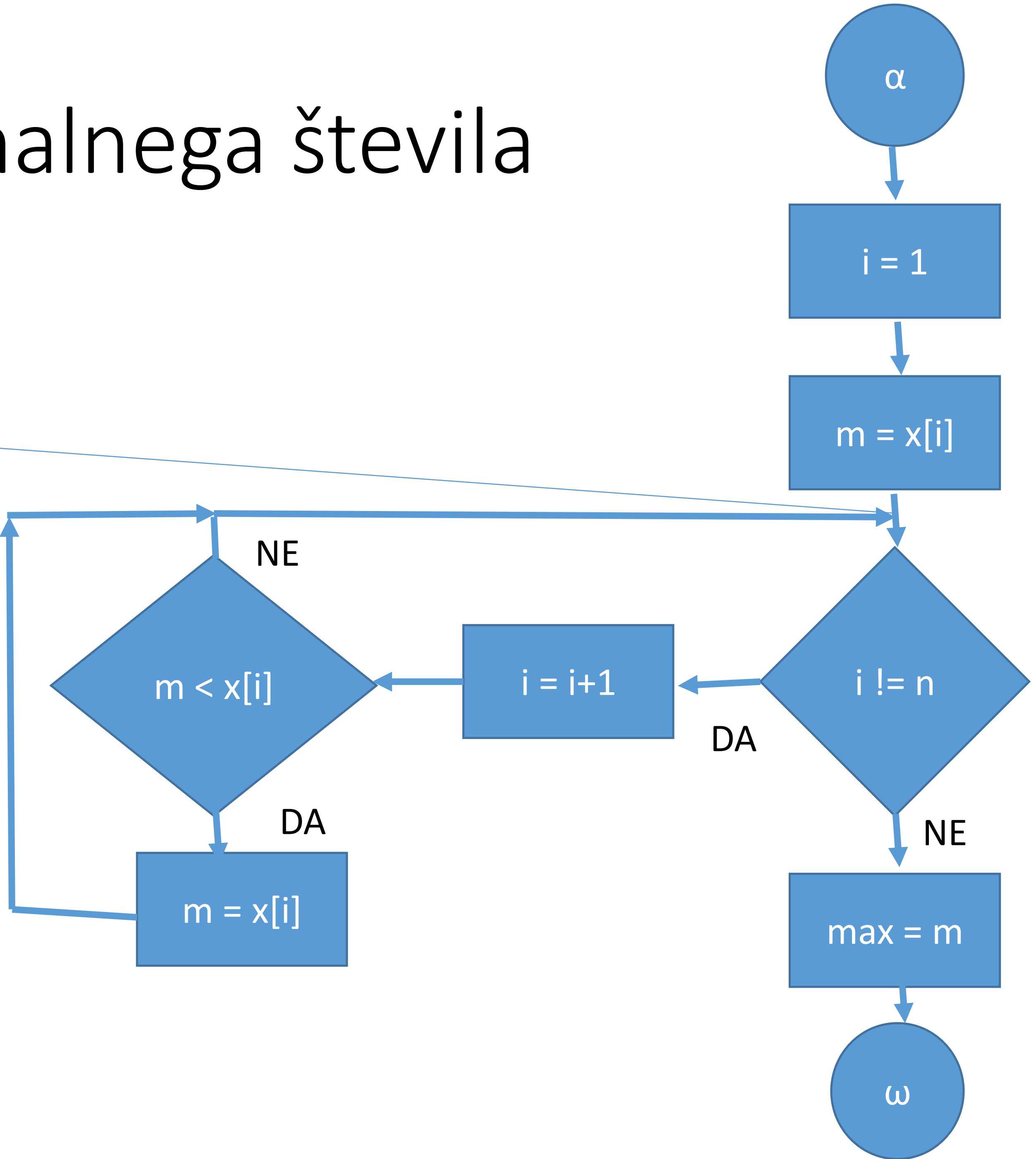
Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

i = 1

m = x[i]

Torej velja: *m =* ***max*** *x[k], k = 1..i*



Primer: Iskanje maksimalnega števila

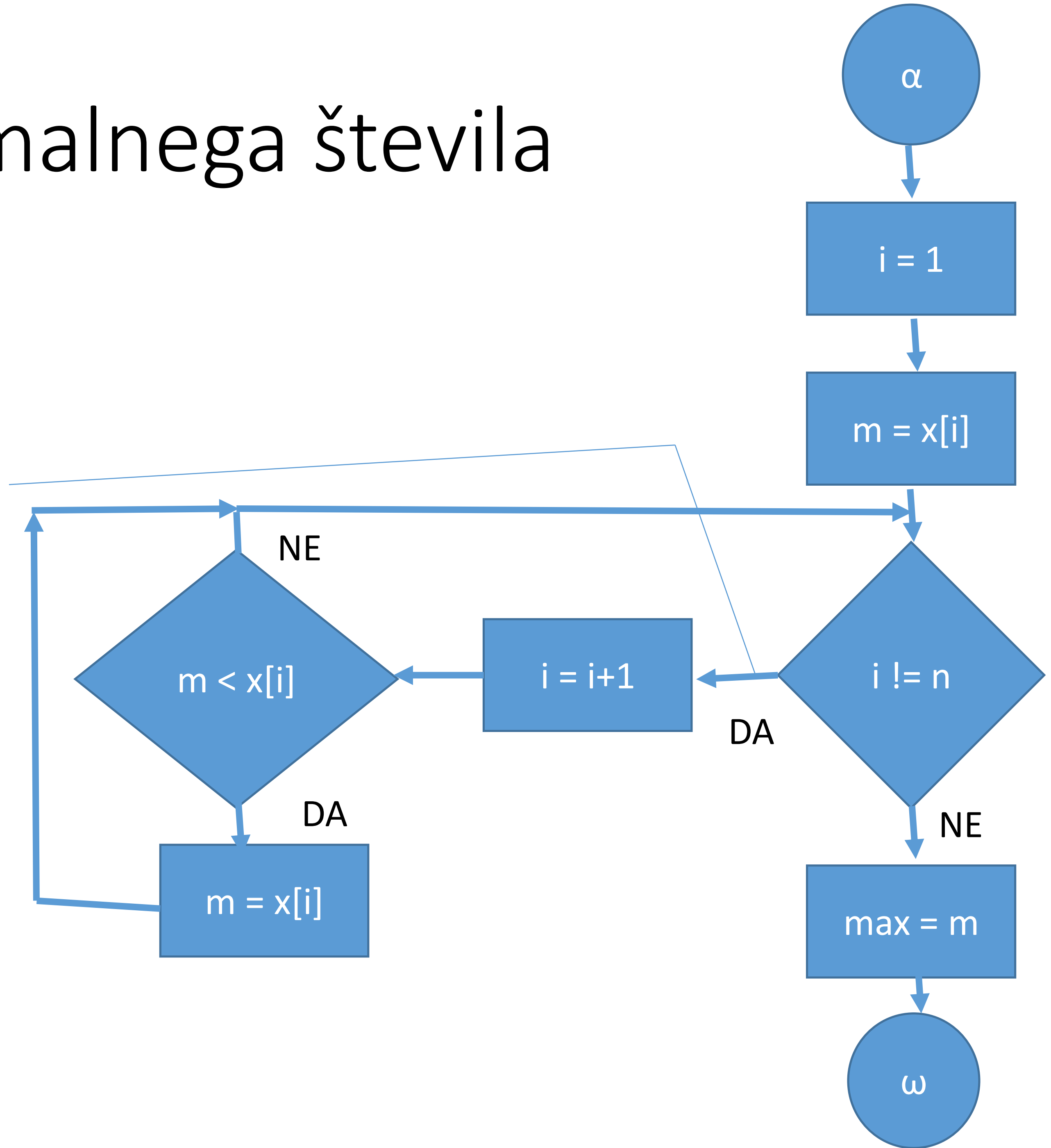
$n > 0$, $x[i] \in R$, $i = 1..n$

$i = 1$

$m = x[i]$

Torej velja: *m =* **max** *x[k], k = 1..i*

$i \mathrel{!=} n$ → $i < n$ in *m =* **max** *x[k], k = 1..i*



Primer: Iskanje maksimalnega števila

n > 0, x[i] ϵ R, i = 1..n

i = 1

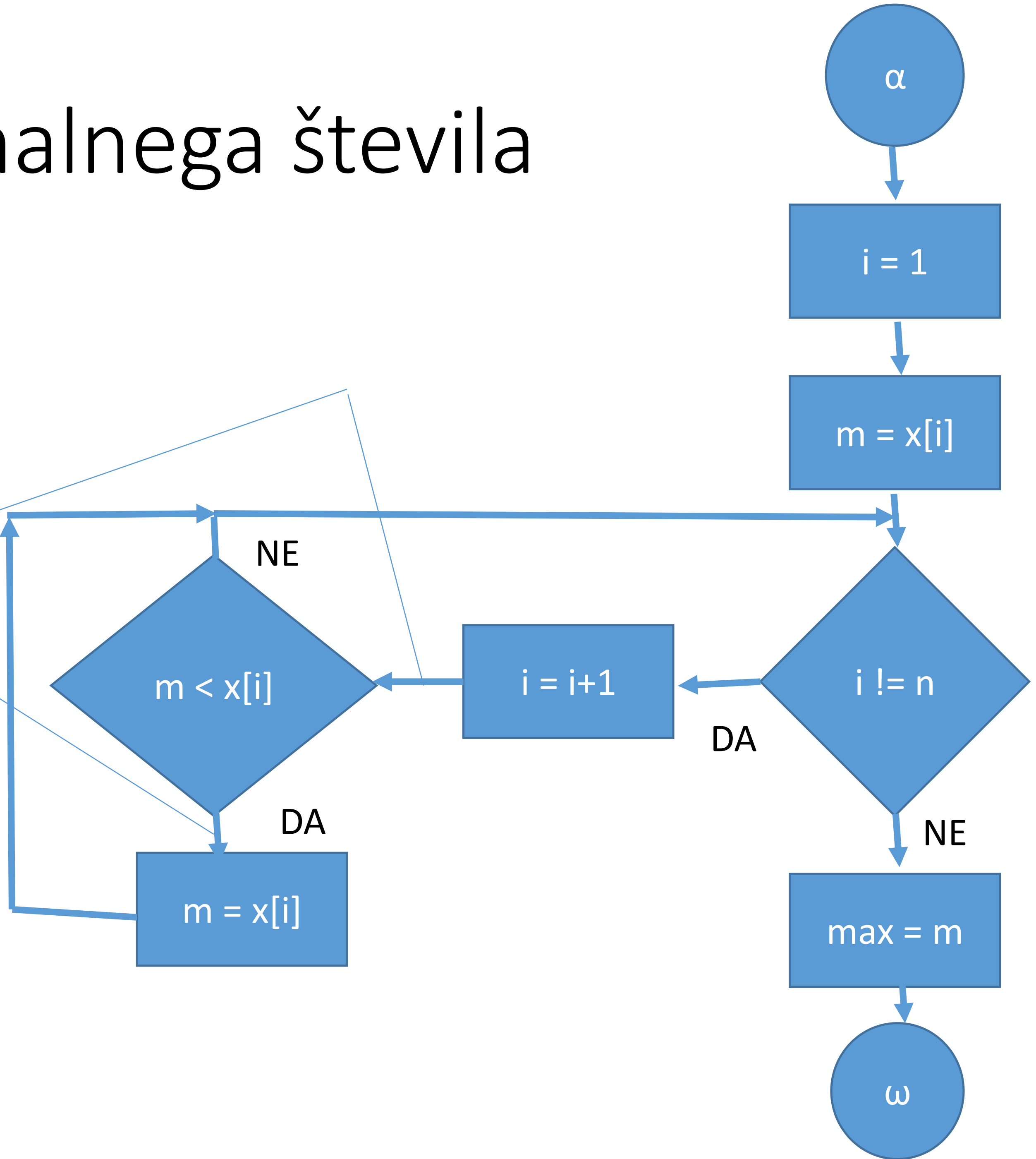
m = x[i]

Torej velja: *m* = **max** *x[k], k = 1..i*

i != n → i < n in *m* = **max** *x[k], k = 1..i*

i <= n in *m* = **max** *x[k], k = 1..i-1*

m < x[i] in *m* = **max** *x[k], k = 1..i-1*



Primer: Iskanje maksimalnega števila

n > 0, x[i] ∊ R, i = 1..n

i = 1

m = x[i]

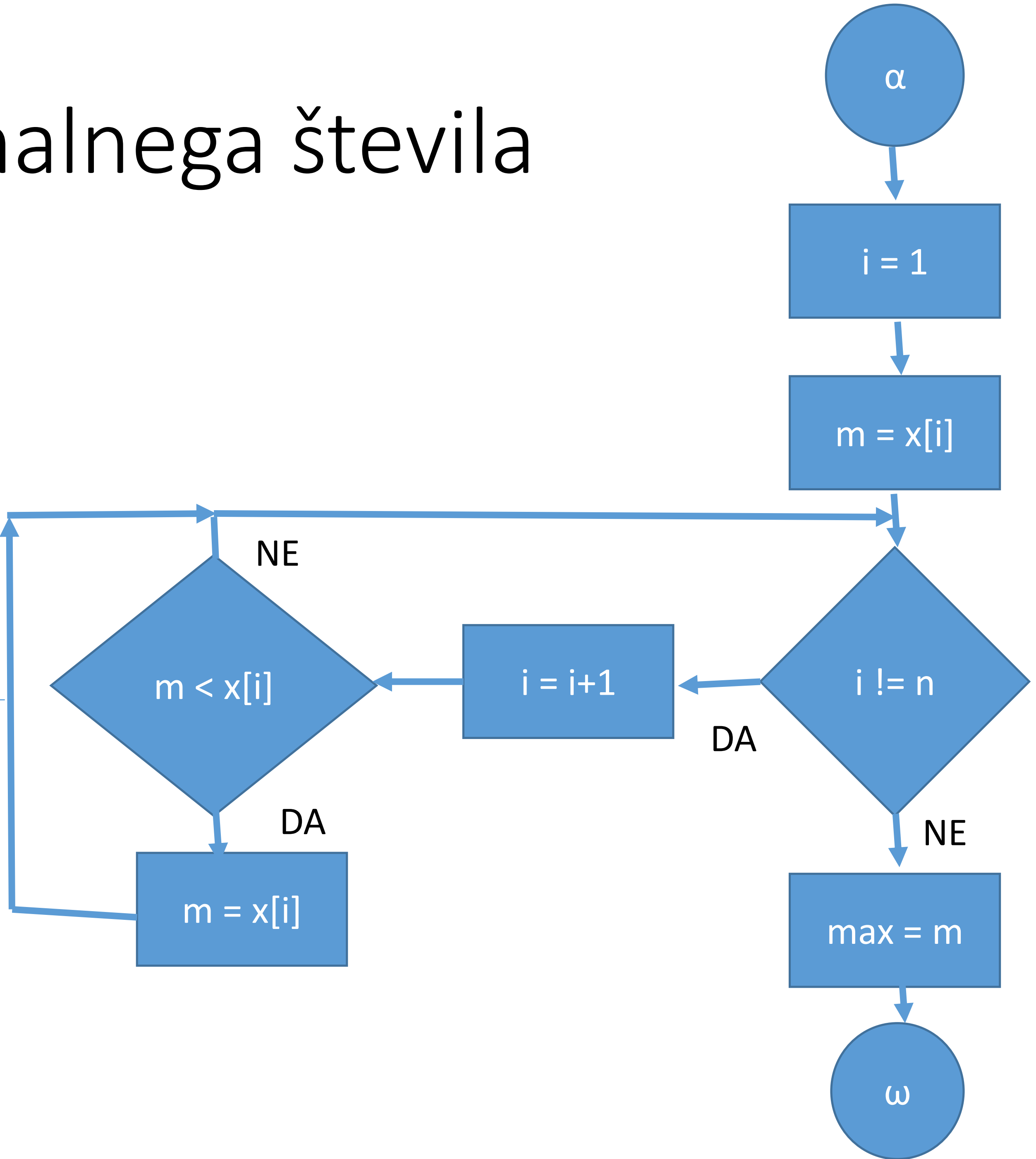
Torej velja: *m* = **max** *x[k], k = 1..i*

i != n → i < n in *m* = **max** *x[k], k = 1..i*

i <= n in *m* = **max** *x[k], k = 1..i-1*

m < x[i] in *m* = **max** *x[k], k = 1..i-1*

m = **max** *x[k], k = 1..i*



Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

i = 1

m = x[i]

Torej velja: *m =* **max** *x[k], k = 1..i*

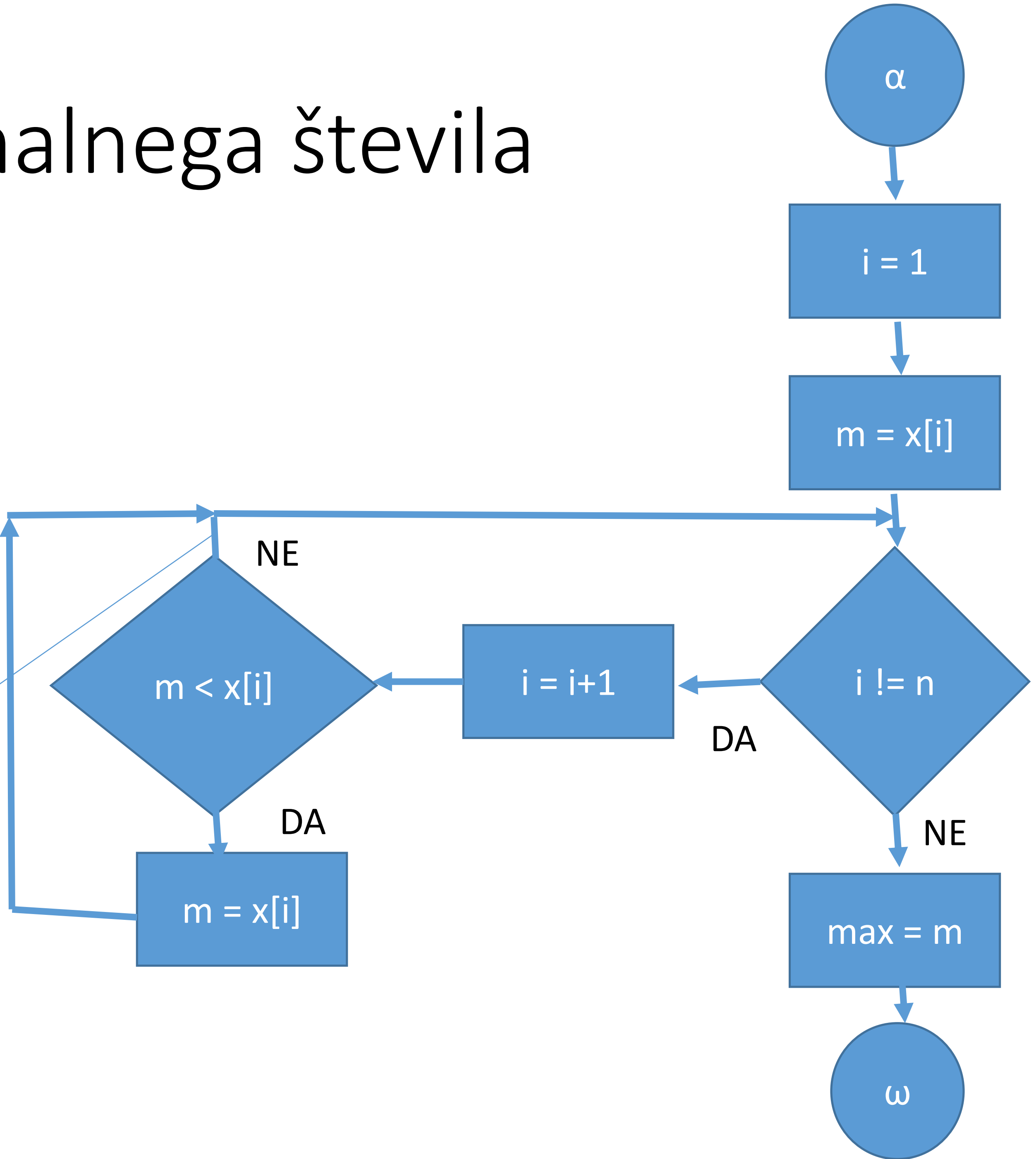
i != n → i < n in *m =* **max** *x[k], k = 1..i*

i <= n in *m =* **max** *x[k], k = 1..i-1*

m < x[i] in *m =* **max** *x[k], k = 1..i-1*

m = **max** *x[k], k = 1..i*

m >= x[i] in *m =* **max** *x[k], k = 1..i-1*



Primer: Iskanje maksimalnega števila

n > 0, x[i] ϵ R, i = 1..n

i = 1

m = x[i]

Torej velja: *m* = **max** *x[k], k = 1..i*

i != n → i < n in *m* = **max** *x[k], k = 1..i*

i <= n in *m* = **max** *x[k], k = 1..i-1*

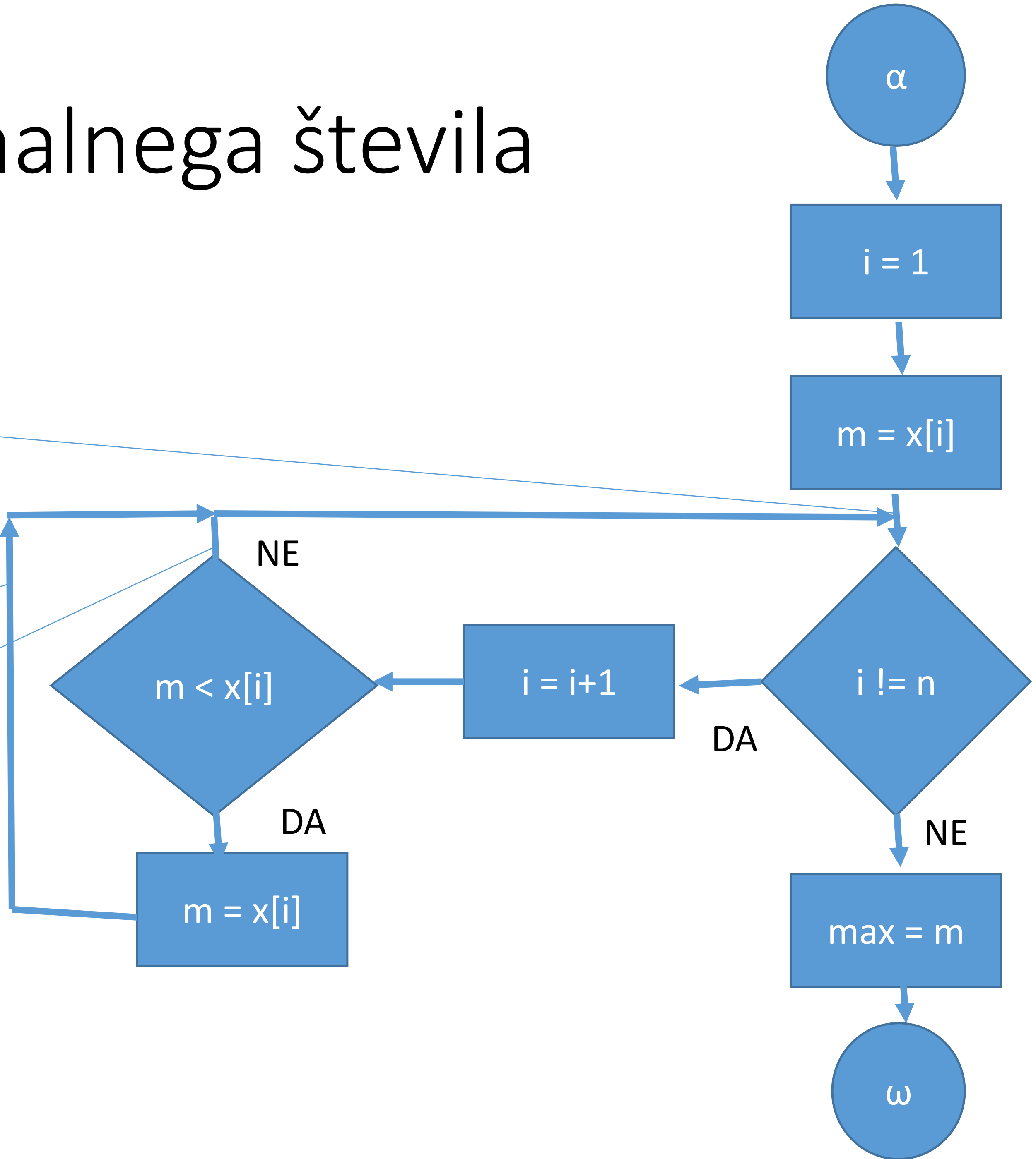
m < x[i] in *m* = **max** *x[k], k = 1..i-1*

m = **max** *x[k], k = 1..i*

m >= x[i] in *m* = **max** *x[k], k = 1..i-1* →

m = **max** *x[k], k = 1..i*

ZANČNA INVARIANTA



Primer: Iskanje maksimalnega števila

n > 0, x[i] ϵ R, i = 1..n

i = 1

m = x[i]

Torej velja: *m* = **max** *x[k], k = 1..i*

i != n → i < n in *m* = **max** *x[k], k = 1..i*

i <= n in *m* = **max** *x[k], k = 1..i-1*

m < x[i] in *m* = **max** *x[k], k = 1..i-1*

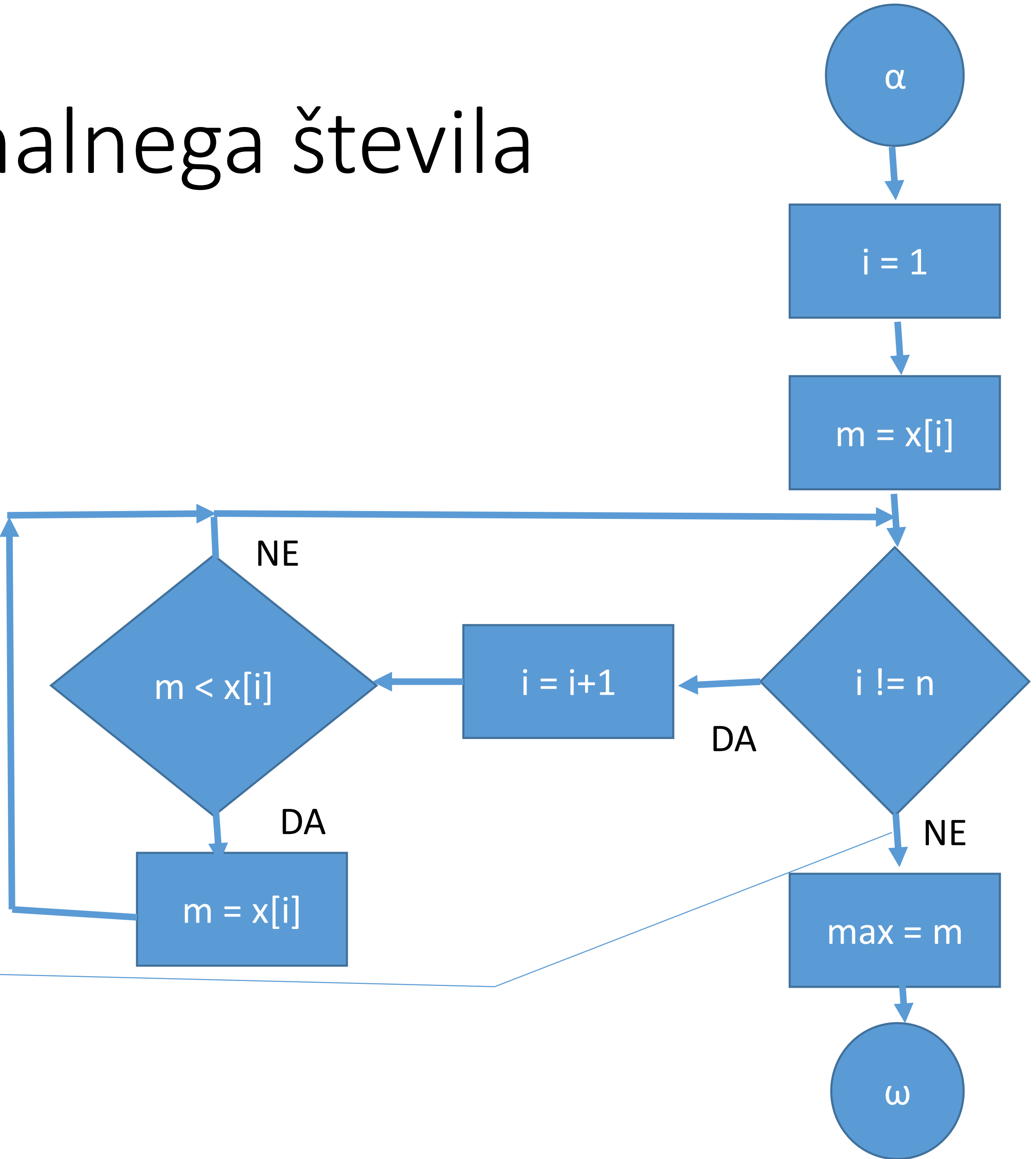
m = **max** *x[k], k = 1..i*

m >= x[i] in *m* = **max** *x[k], k = 1..i-1* →

m = **max** *x[k], k = 1..i*

i = n in *m* = **max** *x[k], k = 1..i* →

m = **max** *x[k], k = 1..n*



Primer: Iskanje maksimalnega števila

$n > 0$, $x[i] \in R$, $i = 1..n$

$i = 1$

$m = x[i]$

Torej velja: $m = \mathbf{max}\ x[k],\ k = 1..i$

$i \mathrel{!=} n \rightarrow i < n$ in $m = \mathbf{max}\ x[k],\ k = 1..i$

$i \le n$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1$

$m < x[i]$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1$

$m = \mathbf{max}\ x[k],\ k = 1..i$

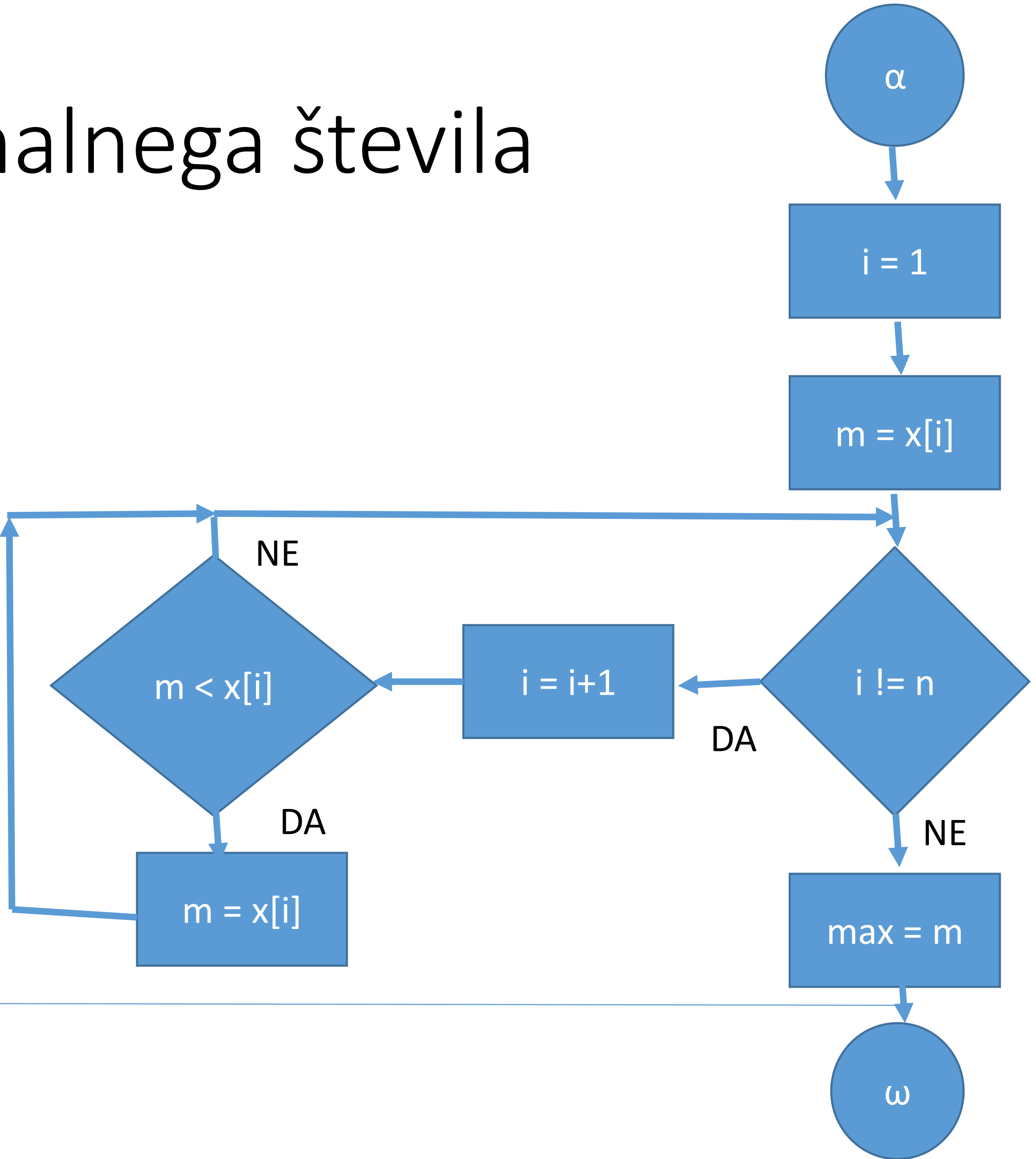
$m \ge x[i]$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1 \rightarrow$

$m = \mathbf{max}\ x[k],\ k = 1..i$

$i = n$ in $m = \mathbf{max}\ x[k],\ k = 1..i \rightarrow$

$m = \mathbf{max}\ x[k],\ k = 1..n$

$max = m \rightarrow max = \mathbf{max}\ x[k],\ k = 1..n$



Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

i = 1

m = x[i]

Torej velja: $m = \mathbf{max}\ x[k], k = 1..i$

i != n → i < n in $m = \mathbf{max}\ x[k], k = 1..i$

i <= n in $m = \mathbf{max}\ x[k], k = 1..i\text{-}1$

m < x[i] in $m = \mathbf{max}\ x[k], k = 1..i\text{-}1$

$m = \mathbf{max}\ x[k], k = 1..i$

m >= x[i] in $m = \mathbf{max}\ x[k], k = 1..i\text{-}1$ →

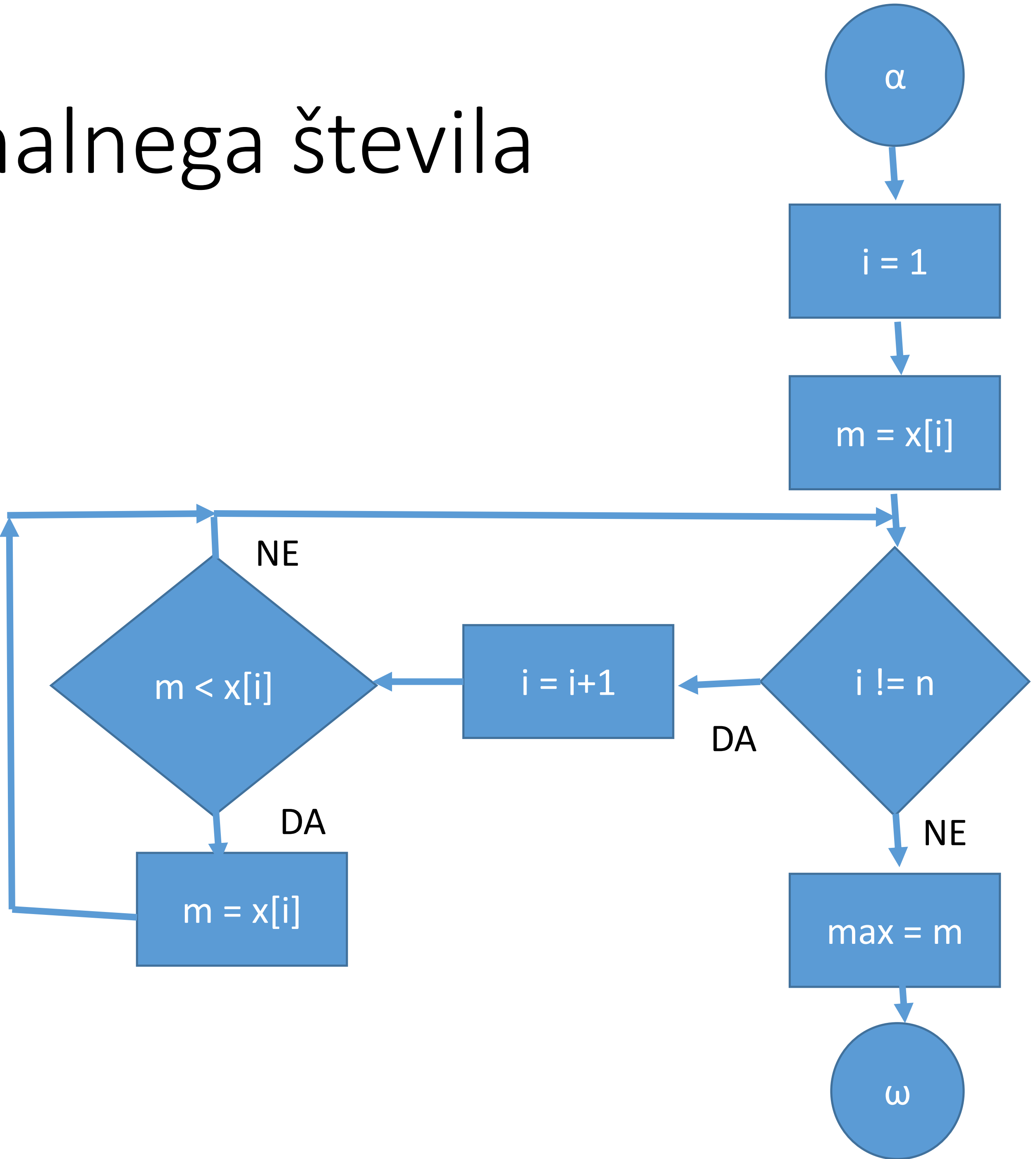
$m = \mathbf{max}\ x[k], k = 1..i$

i = n in $m = \mathbf{max}\ x[k], k = 1..i$ →

$m = \mathbf{max}\ x[k], k = 1..n$

max = m → $max = \mathbf{max}\ x[k], k = 1..n$

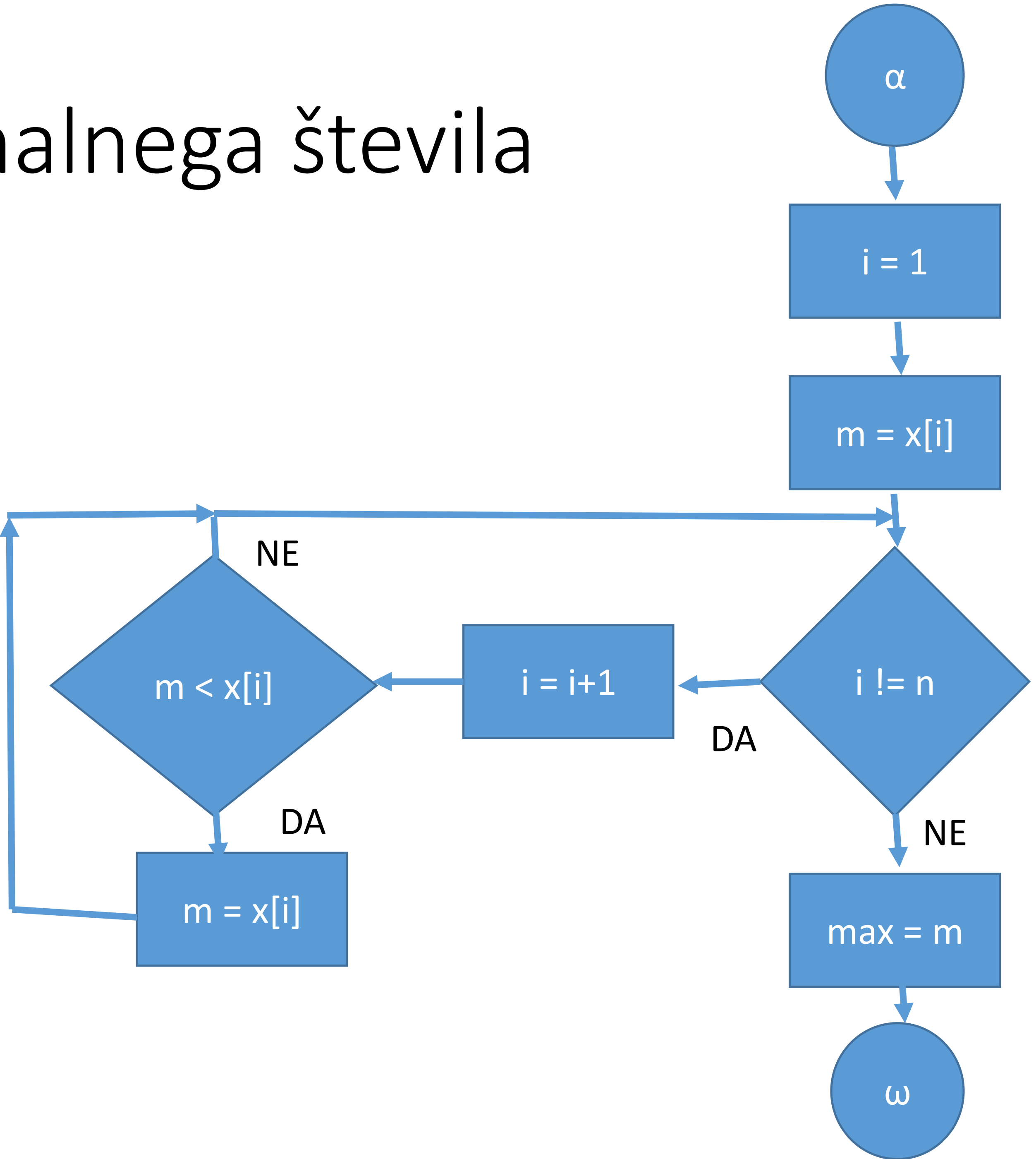
PROGRAM JE PARCIALNO PRAVILEN



Primer: Iskanje maksimalnega števila

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: *$l = n-i$*
3) **Zančna invarianta: *$n - i \in N$***



Primer: Iskanje maksimalnega števila

$n > 0,\ x[i] \in R,\ i = 1..n$

$i = 1$

$m = x[i]$

Torej velja: $m = \mathbf{max}\ x[k],\ k = 1..i$

$i \mathrel{!=} n \rightarrow i < n$ in $m = \mathbf{max}\ x[k],\ k = 1..i$

$i <= n$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1$

$m < x[i]$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1$

$m = \mathbf{max}\ x[k],\ k = 1..i$

$m >= x[i]$ in $m = \mathbf{max}\ x[k],\ k = 1..i-1 \rightarrow$

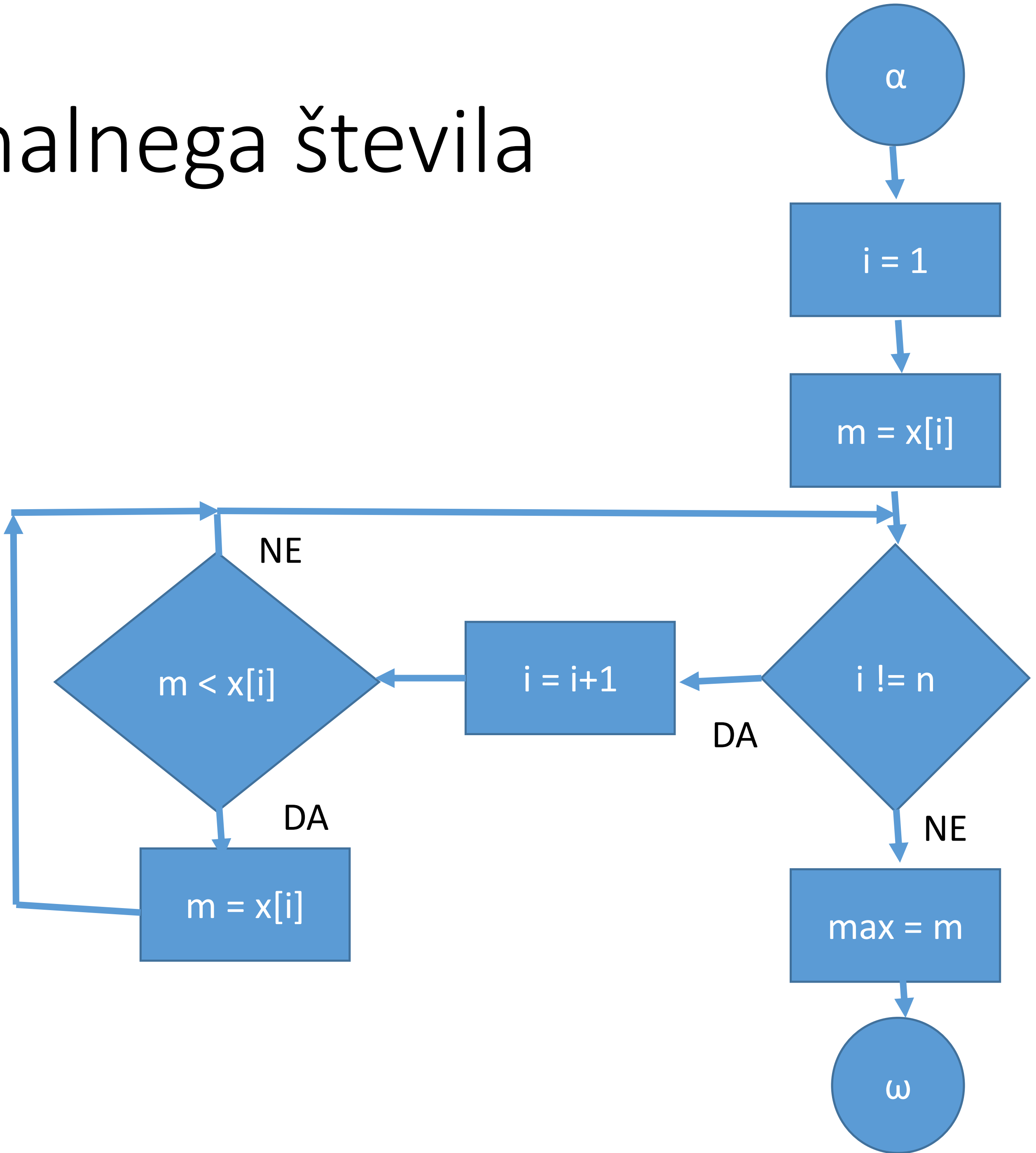
$m = \mathbf{max}\ x[k],\ k = 1..i$

$i = n$ in $m = \mathbf{max}\ x[k],\ k = 1..i \rightarrow$

$m = \mathbf{max}\ x[k],\ k = 1..n$

$max = m \rightarrow max = \mathbf{max}\ x[k],\ k = 1..n$

PROGRAM JE PARCIALNO PRAVILEN



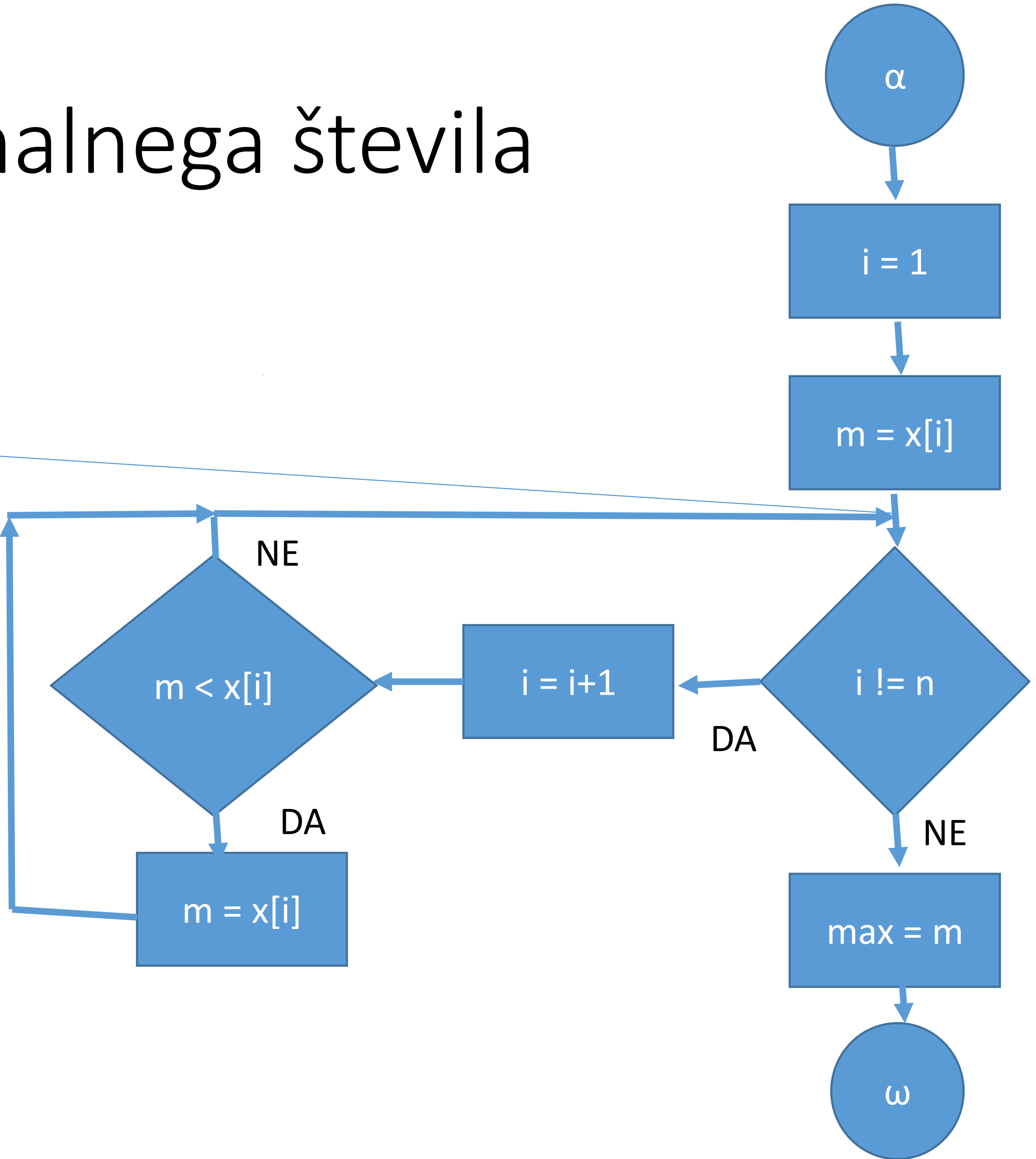
Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

i = 1

m = x[i]

Torej velja: ***$n - i \in N$***



Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

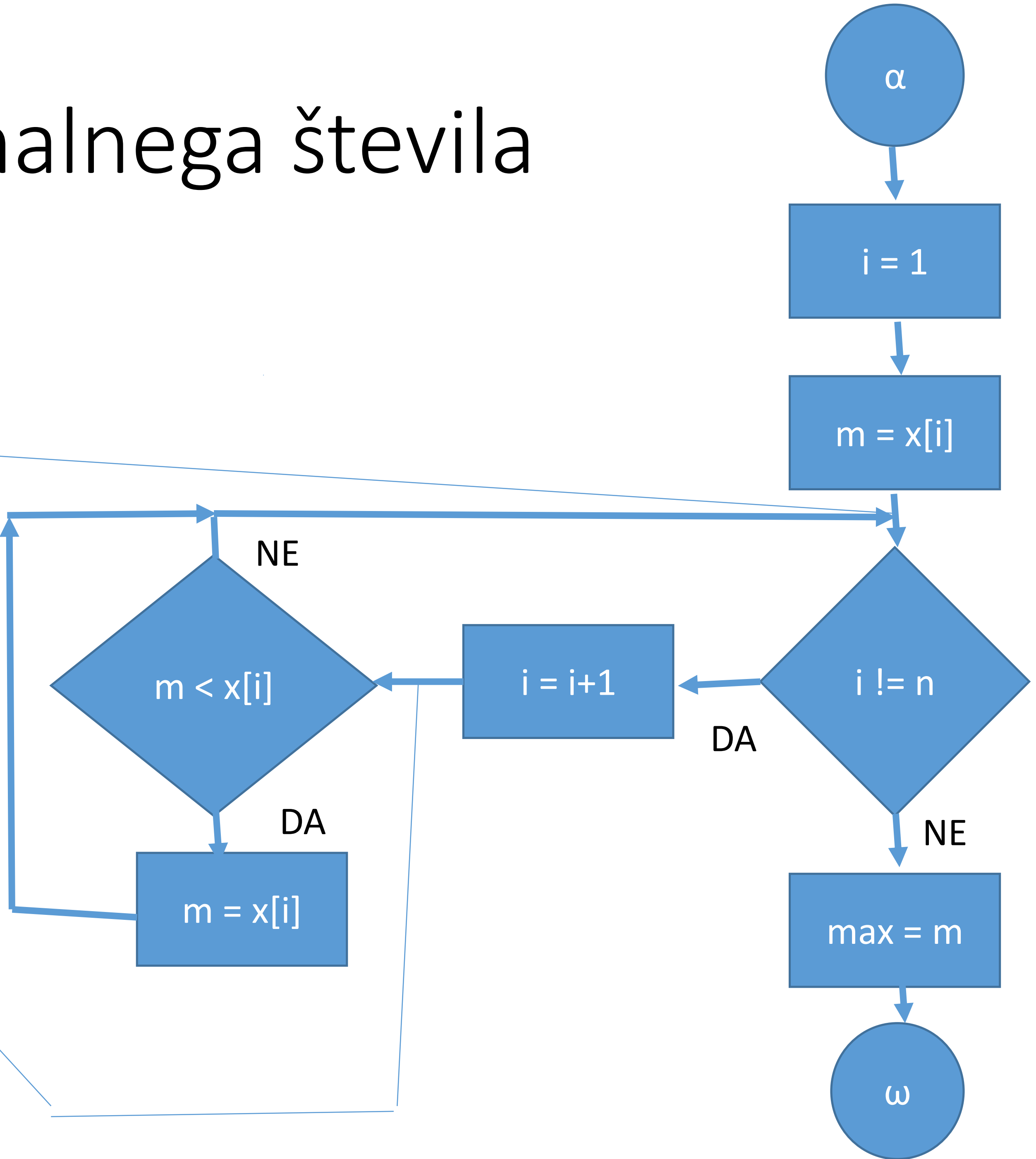
i = 1

m = x[i]

Torej velja: ***n – i ϵ N***

i != n → i < n in *m* = **max** *x[k], k = 1..i*

i <= n → ***n – i ϵ N***



Primer: Iskanje maksimalnega števila

$n > 0, x[i] \in R, i = 1..n$

$i = 1$

$m = x[i]$

Torej velja: $\boldsymbol{n - i \in N}$

$i \mathrel{!}= n \rightarrow i < n$ in $m = \mathbf{max}\ x[k], k = 1..i$

$i <= n \rightarrow \boldsymbol{n - i \in N}$

$m < x[i]$ in $m = \mathbf{max}\ x[k], k = 1..i\text{-}1$

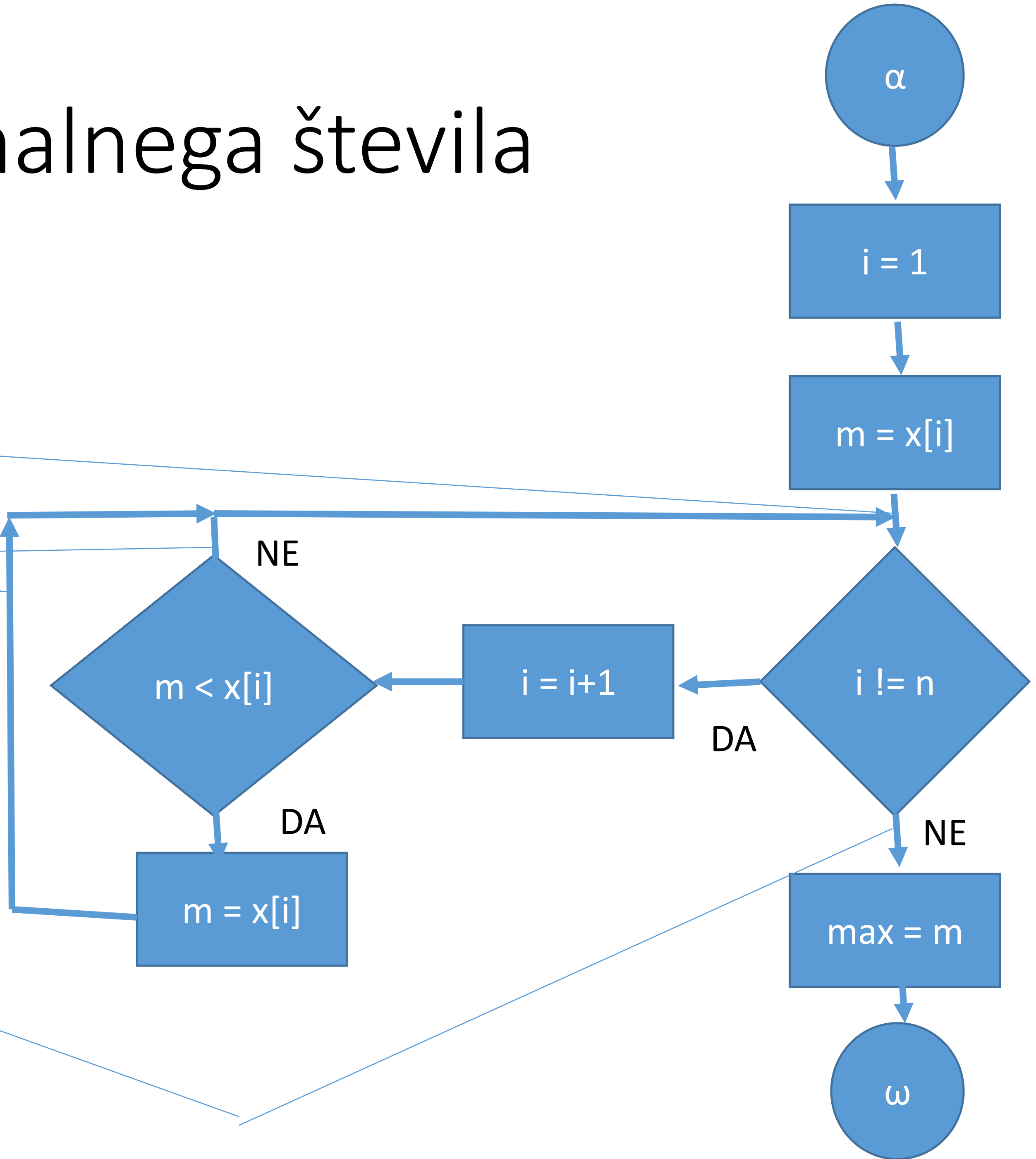
$m = \mathbf{max}\ x[k], k = 1..i$

$m >= x[i]$ in $m = \mathbf{max}\ x[k], k = 1..i\text{-}1 \rightarrow$

$m = \mathbf{max}\ x[k], k = 1..i$

$i = n \rightarrow \boldsymbol{n - i \in N}$

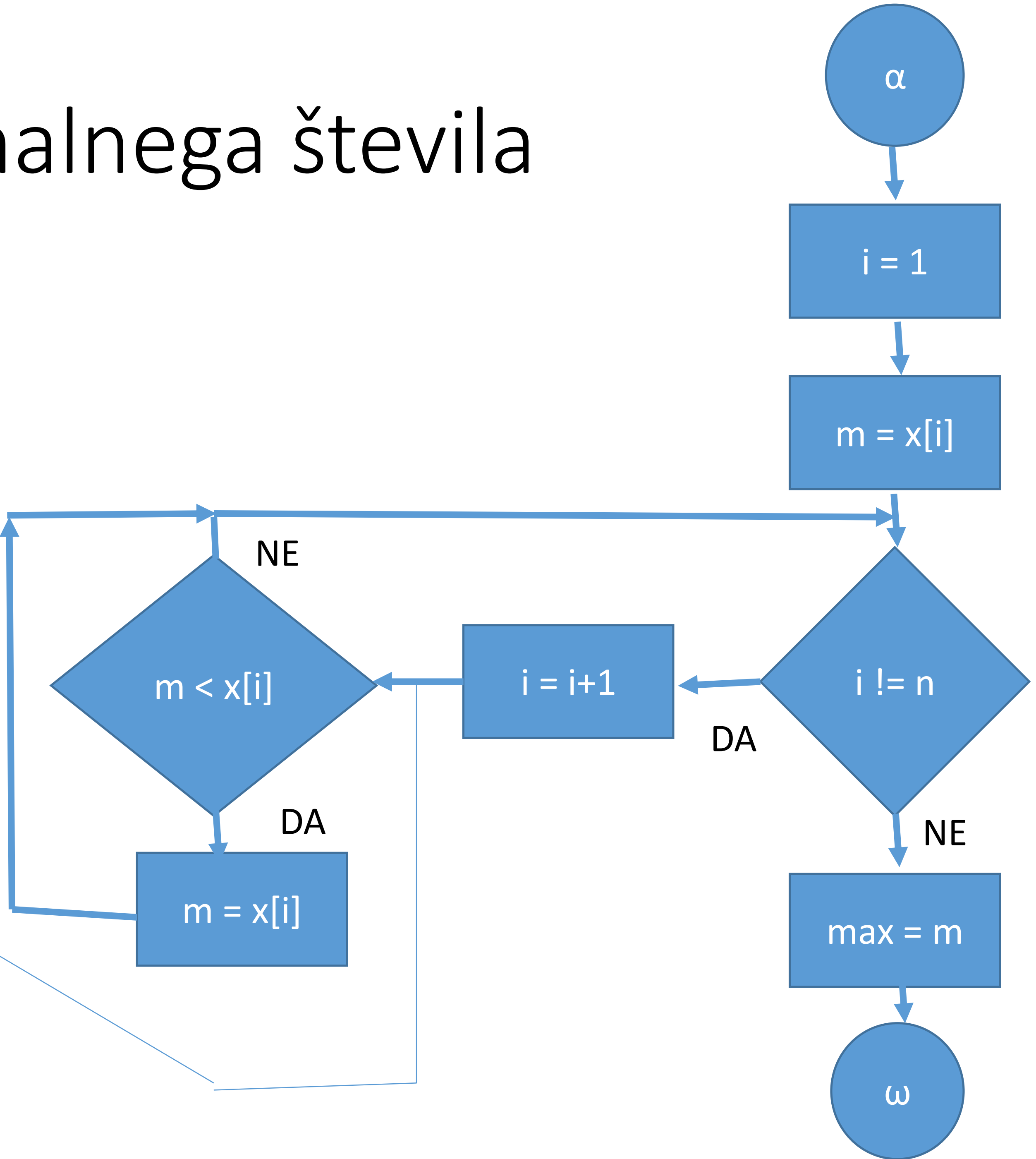
ZANČNA INVARIANTA



Primer: Iskanje maksimalnega števila

Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: $l = n - i$
3) **Zančna invarianta:** $\boldsymbol{n - i \in N}$
4) Vrednost l se zmanjšuje
   $ll = n - i - 1 < l = n - i$



Primer: Iskanje maksimalnega števila

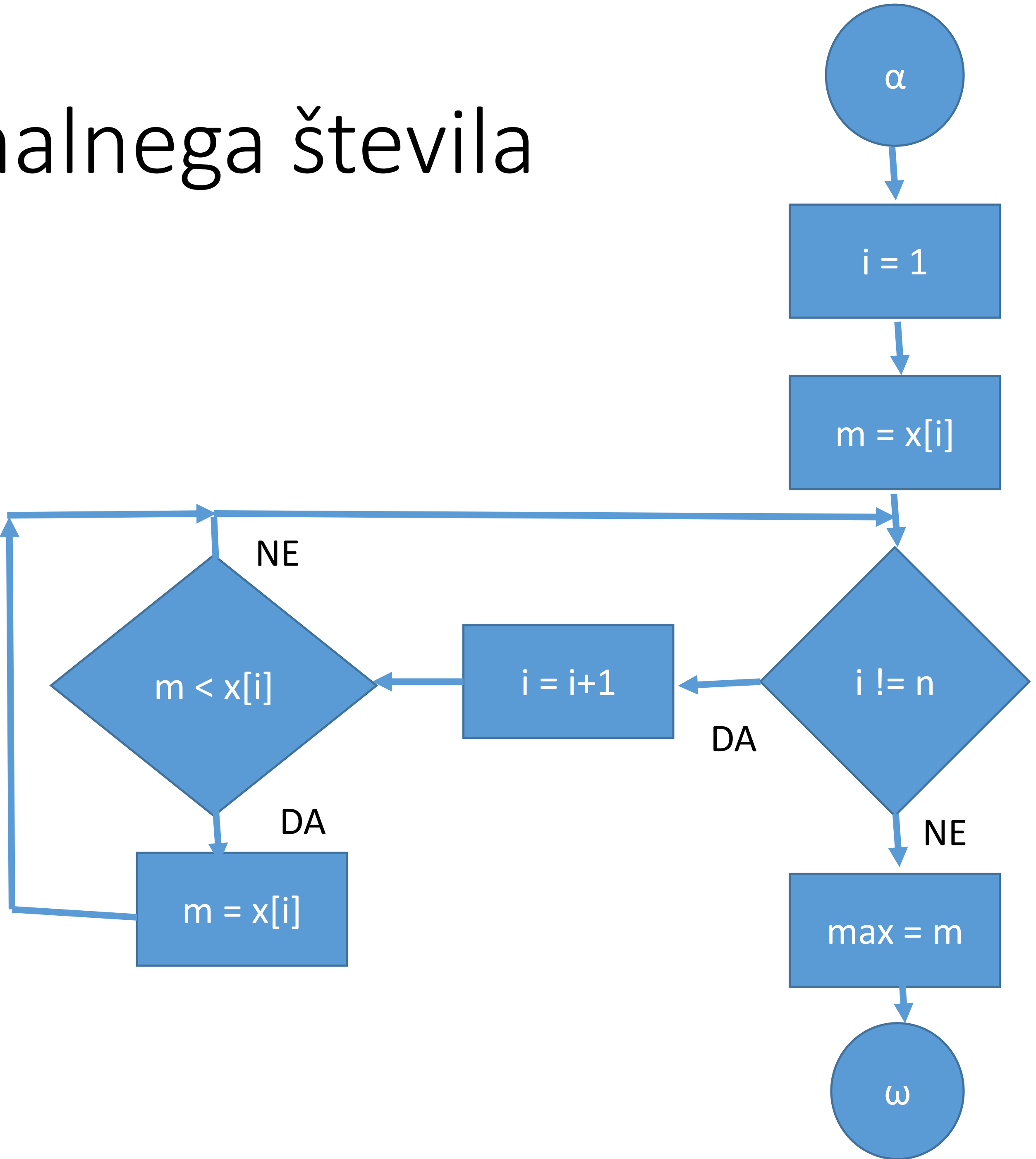
Treba je dokazati še ustavljivost programa.

1) Dobro utemeljena množica: *N (z 0)*
2) Zančna spremenljivka: $l = n - i$
3) **Zančna invarianta:** $n - i \in N$
4) Vrednost *l* se zmanjšuje

   $ll = n - i - 1 < l = n - 1$

PROGRAM JE TOTALNO PRAVILEN:

- Je parcialno pravilen
- Se vedno ustavi



Bogek dragi.. Daj neki znak,
Jel bu ova godina išla
kak na bolje?