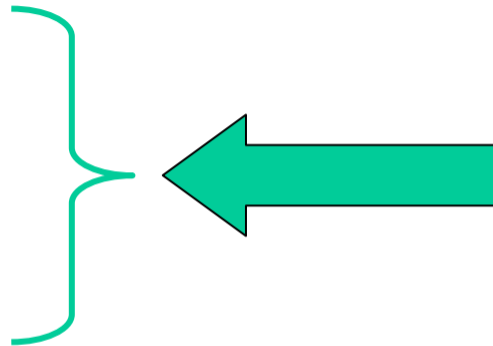


– **Struttura di Programma**

Sequenza

Selezione

Iterazione



Istruzioni per il trasferimento del controllo:

Istruzione break:

L'istruzione **break** provoca l'uscita immediata dal **ciclo** (o da un'istruzione **switch**) in cui è racchiusa

Istruzione continue:

L'istruzione **continue** provoca l'inizio della successiva iterazione del **ciclo** in cui è racchiusa (non si applica a **switch**).

```
for (i = val1; i <= val2; i++)
{
    ... ;
    if ... continue;
    else ...
}
```

Esempio continue:

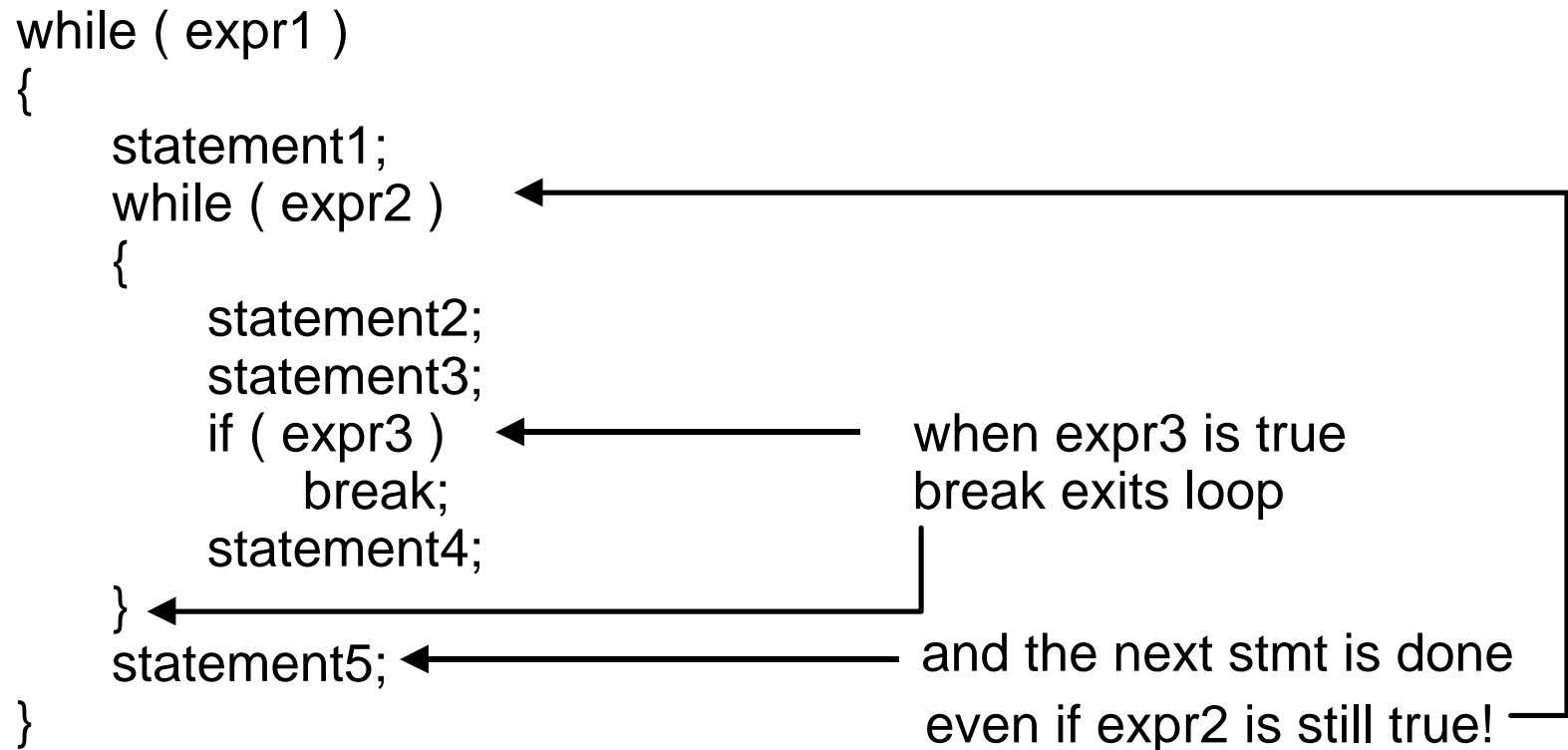
```
for (i = N; i > 0; i--)
{
    /* Salta alla prossima ripetizione
    se N è multiplo di i */
    if (N % i) continue;
    /* Esci dal ciclo se i vale 55 */
    if (i == 55) break;
    ...
}
```

break and continue

- break statement
 - used in loops (e.g., for(), while(), and do-while() statements) **and** in the switch() statement
 - causes immediate **exit from innermost loop** (exits switch in switch statement)
- continue statement
 - **used in loops only** (e.g., for(), while(), and do-while()), **NOT in the switch() statement; Why?**
 - causes immediate **exit from BODY** of innermost loop AND loop condition is evaluated for loop continuation

C Program Control Statements

break



C Program Control Statements

break

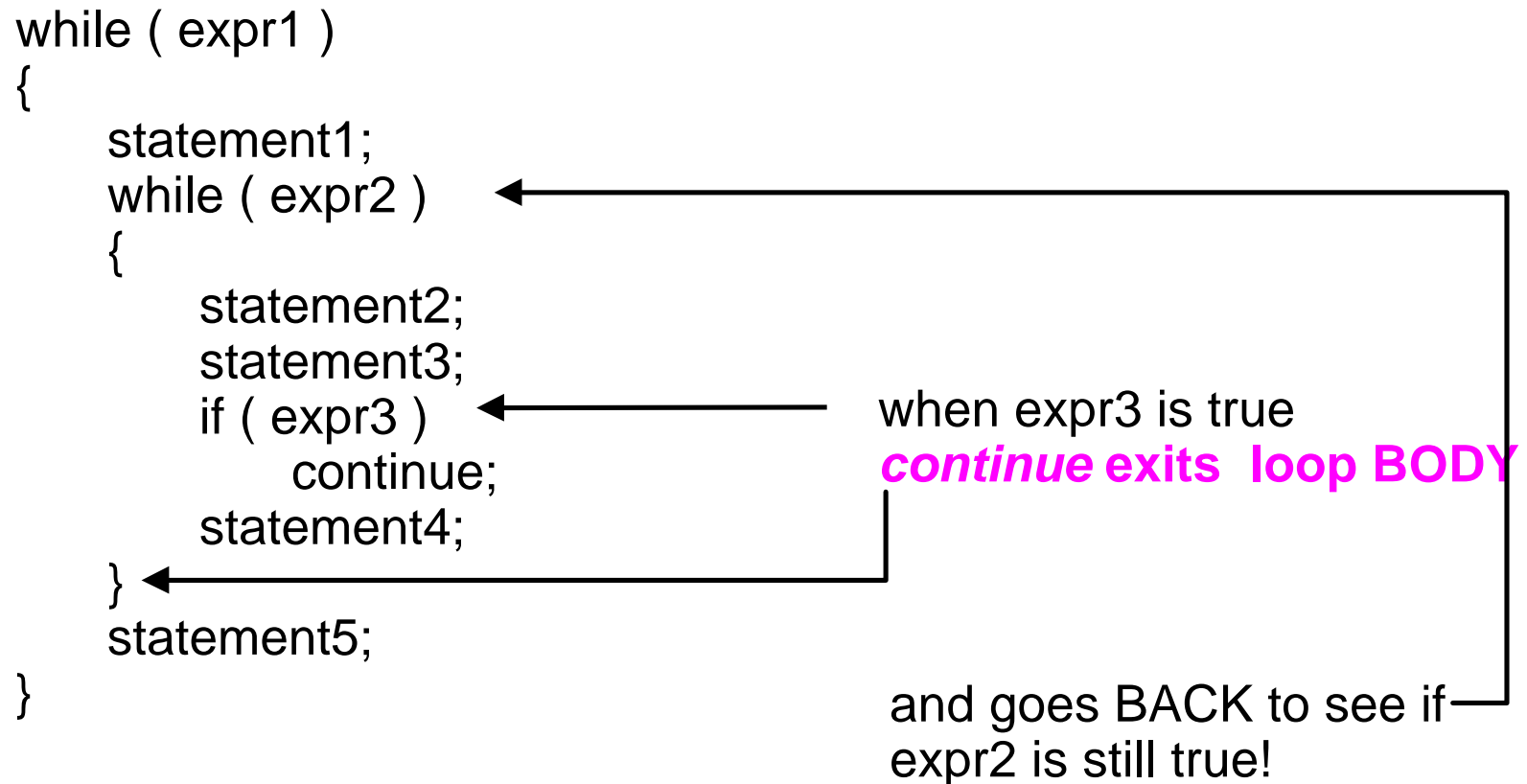
```
do
{
    statement1;
    do
    {
        statement2;
        statement3;
        if ( expr3 ) ←
            break;
        statement4;
    } while (expr2 );
    statement5; ←
} while (expr1 );
```

when expr3 is true
break exits loop

even if expr2 is still true
and the next stmt is done

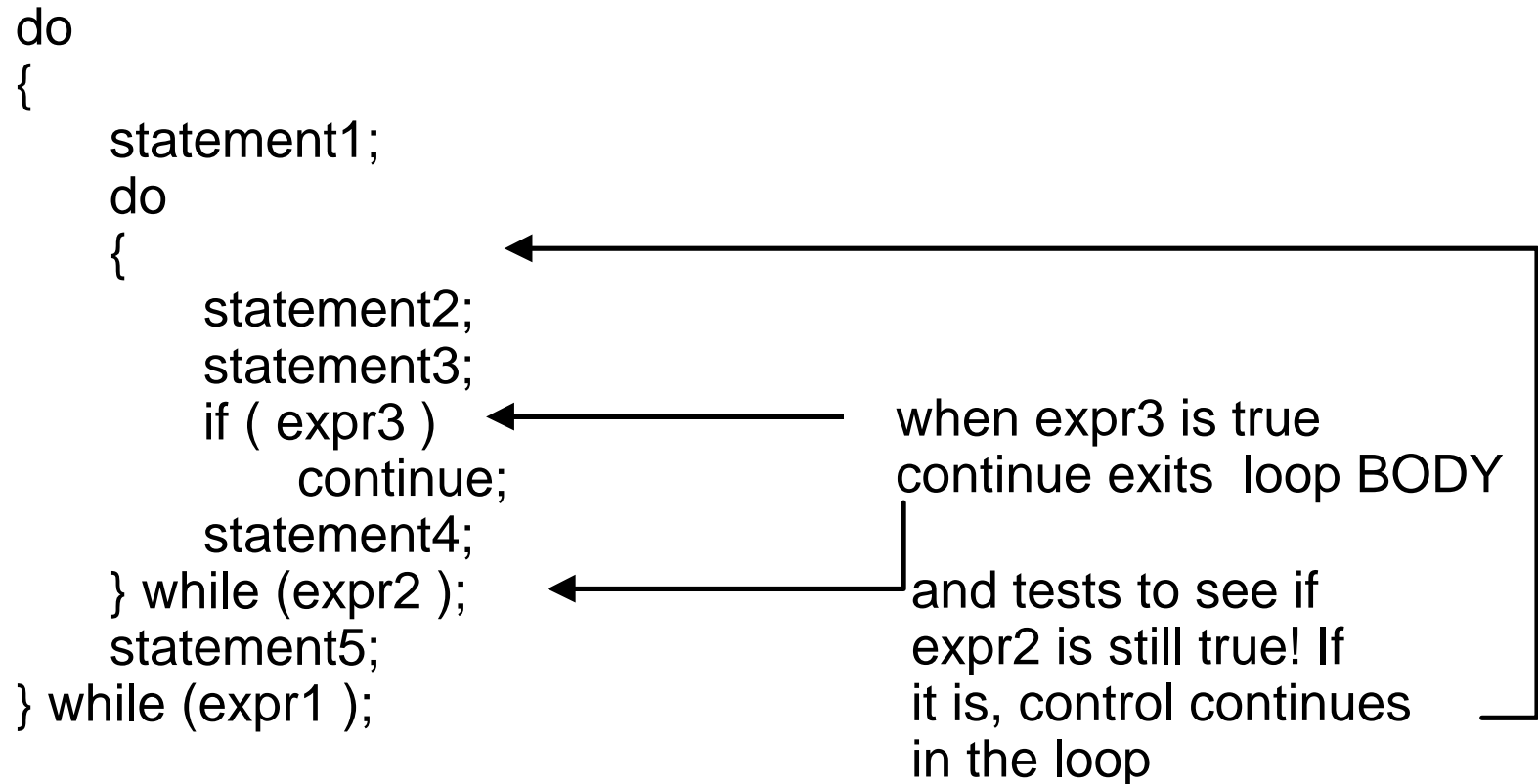
C Program Control Statements

continue



C Program Control Statements

continue



C Program Control Statements

continue

Does same thing in a for() statement:

```
for ( expr1 ; expr2 ; expr3 )  
{  
    statement1;  
    statement2;  
  
    if( expr4 )  
        continue; ← jumps to end of loop  
    statement3;  
} ←
```

which causes expr3 to be executed and expr2 to be evaluated for continuing the loop

C Program Control Statements

break and continue

What is printed?

2
4
5

Why is "bottom of loop"
skipped?

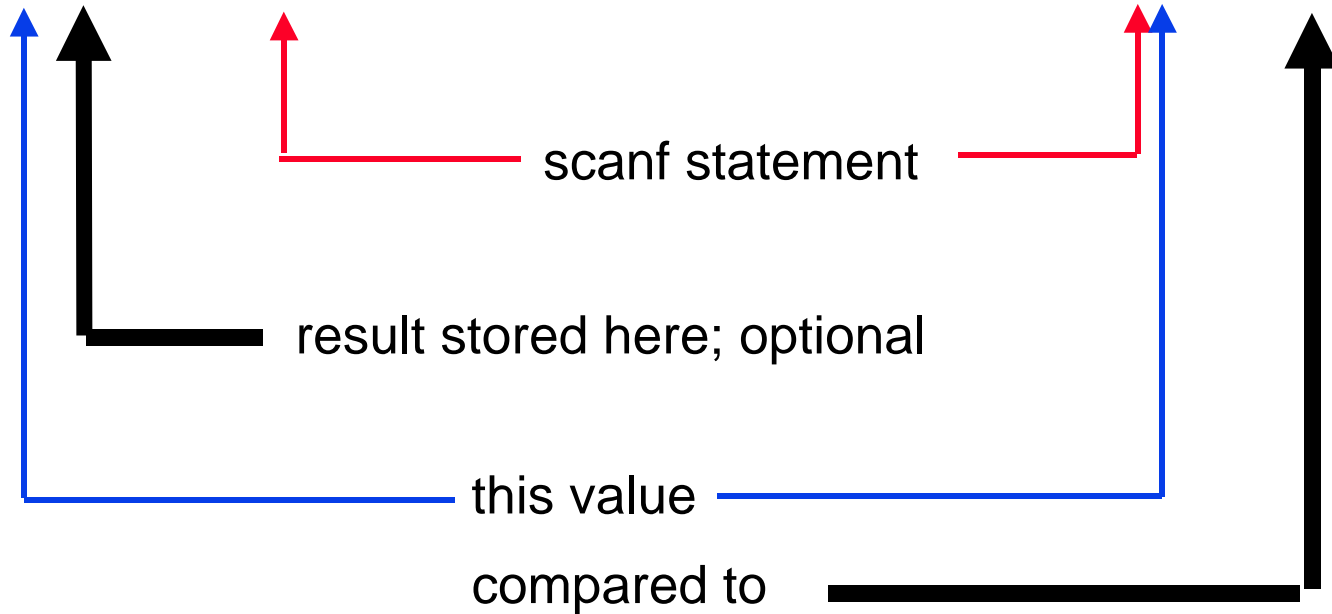
```
int ctr = 0;

while( ctr < 6 )
{
    if( ctr < 3 )
    {
        ctr += 2;
        printf( "%d\n", ctr );
        continue;
    }
    else
    {
        printf( "%d\n", ++ctr );
        break;
    }
    printf( "Bottom of loop\n" );
}
```

C Program Control Statements

Sample C Code scanf in while

```
while( ( vals = scanf( "%lf%c%lf", &op1, &action, &op2) ) == 3 )
```



ESEMPIO1 - CALCOLO DEL FATTORIALE

```
#include <stdio.h>

main ()
{
    int n,i,fattoriale;

    fattoriale=1;

    printf("Inserisci il numero positivo \n");
    scanf ("%d",&n);

    if(n!=0)
    {
        for(i=1;i<=n;i=i+1)
            fattoriale=fattoriale*i;
    }
    printf("il fattoriale di %d e' %d\n", n,
           fattoriale);
}
```

ESEMPIO2 - CALCOLO DEI DIVISORI DI UN INTERO

```
#include <stdio.h>

main ()
{
    int k,j;

    printf("Inserisci il numero \n");
    scanf("%d",&k);

    if(k!=0)
    {
        for(j=1;j<=k;j=j+1)
            if (k%j==0)
                printf("%d e' divisore di %d \n", j,k);
    }
    else
        printf("Non c'e' alcun divisore di %d\n",k);
}
```

ESEMPIO3 -

CALCOLO PRODOTTO MATRICE PER VETTORE (mat*vett=vris)

```

.....
int      vett[M], mat[N][M], vris[N];
int      i,j;
.....

    for(i=0;i<=N-1;i++) per ogni elem del vett risult
    {
        vris[i]=0;
        for (j=0; j<=M-1;j++)
            vris[i]=vris[i]+mat[i][j]*vet[j];
    }

```

ESEMPIO4 -

CALCOLO PRODOTTO MATRICE PER MATRICE (mat1*mat2=ris)

```

.....
int      mat1[N][M], mat2[M][K], ris[N][K];
int      i,j,k;
.....

for(i=0;i<=N-1;i++) per ogni riga
{
    for (k=0;k<=K-1,k++) per ogni colonna
    {
        ris[i][k]=0;
        for (j=0; j<=M-1;j++)
            ris[i][k]=ris[i][k]+mat1[i][j]*mat2[j][k];
    }
}

```

Espressioni (complementi)

Operatori sui bit:

<<	shift a sinistra	$k \ll 4$ shift a sinistra di 4 bit equivale a $k * 16$
>>	shift a destra	$k \gg 4$ shift a destra di 4 bit equivale a $k / 16$
&	and bit a bit	
	or inclusivo bit a bit	
^	or esclusivo bit a bit	
~	complemento a 1	

Operatore condizionale:

- ?: condizione ? parteVera : parteFalse
- la **parteVera** viene valutata solo se la condizione è verificata (valore diverso da 0)
 - la **parteFalse** viene valutata solo se la condizione **non** è verificata (valore uguale a zero)

```
x = (y != 0 ? 1/y : INFINITY);  
k = a < b ? a : b; /* assegna il minore */
```

Precedenza ed Associatività degli Operatori

La **precedenza degli operatori** è stabilita dalla sintassi delle espressioni.

$$a + b * c$$

equivale **sempre** a

$$a + (b * c)$$

L'**associatività** è ancora stabilita dalla sintassi delle espressioni.

$$a + b + c$$

equivale **sempre** a

$$(a + b) + c$$