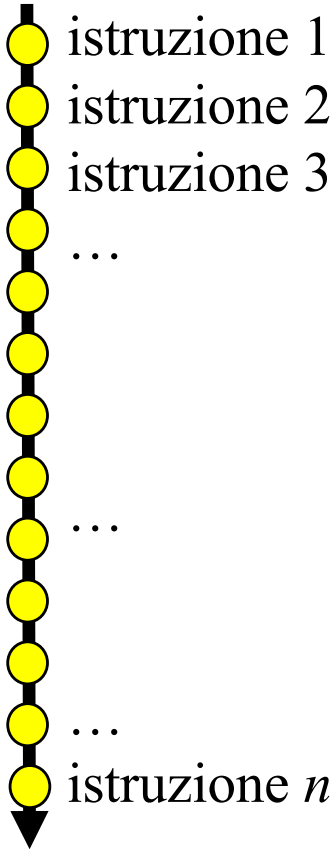
Capitolo 6

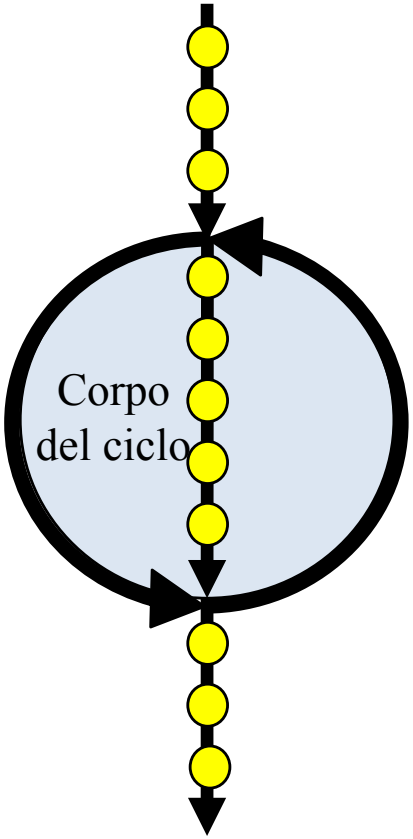
# Istruzioni di iterazione

# Istruzioni di iterazione

- Le istruzioni di iterazione servono per creare "cicli"
- Un ***ciclo*** permette di eseguire ripetutamente istruzioni (il ***corpo del ciclo***)
- In C ogni ciclo ha una ***espressione di controllo***
- Ogni volta che il corpo del ciclo viene eseguito (una ***iterazione*** del ciclo), l'espressione di controllo viene valutata
  - Se l'espressione di controllo è vera (ha un valore diverso da 0) il ciclo continua la sua esecuzione

# Cicli


istruzione 1
istruzione 2
istruzione 3
…
…
…
istruzione *n*


Corpo
del ciclo

# Istruzioni per le iterazioni

- In C esistono 3 istruzioni per le iterazioni:

- L'istruzione **`while`** viene usata per i cicli in cui l'espressione di controllo viene valutata ***prima*** dell'esecuzione del corpo del ciclo
- L'istruzione **`do`** viene usata per i cicli in cui l' espressione di controllo viene valutata ***dopo*** l'esecuzione del corpo del ciclo
- L'istruzione **`for`** è pensata per cicli in cui ad ogni iterazione si incrementa o decrementa una variabile.

# L'istruzione `while`

- Un ciclo `while` ha la forma seguente

  `while ( `*expression*` )` *statement*

- *expression* è l'espressione di controllo mentre *statement* è il corpo del ciclo

# L'istruzione `while`

- Esempi di `while`

```
while (i < n)   /* controlling expression */
  i = i * 2;    /* loop body */
```

- Quando un `while` viene eseguito, si valuta l'espressione di controllo
- Se il suo valore è diverso da 0 (cioè, se è *vera*) il corpo del ciclo viene eseguito
- Quindi l'espressione viene valutata nuovamente e l'esecuzione continua fino a quando l'espressione di controllo assume valore 0 (cioè, diventa *falsa*)

# L'istruzione `while`

- Il seguente `while` calcola la più piccola potenza di 2 che è maggiore o uguale a `n`:

```
i = 1;
while (i < n)
  i = i * 2;
```

- Dettagli dell'esecuzione per `n`=10:

| | |
|---|---|
| `i = 1;` | `i` ora vale 1. |
| `i < n?` | Si; continua ed esegui la 1ª iterazione |
| `i = i * 2;` | `i` ora vale 2. |
| `i < n?` | Si; continua ed esegui la 2ª iterazione |
| `i = i * 2;` | `i` ora vale 4. |
| `i < n?` | Si; continua ed esegui la 3ª iterazione |
| `i = i * 2;` | `i` ora vale 8. |
| `i < n?` | Si; continua ed esegui la 4ª iterazione |
| `i = i * 2;` | `i` ora vale 16. |
| `i < n?` | No; esci dal ciclo |

# L'istruzione `while`

- Il corpo del ciclo è una singola istruzione
  - questo dettaglio è meramente tecnico
- È possibile creare un corpo con quante istruzioni si vuole semplicemente usando un'istruzione composta:

```
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

- È prassi comune usare le parentesi graffe anche se il corpo contiene una sola istruzione

```
while (i < n) {
  i = i * 2;
}
```

# L'istruzione `while`

- Il seguente pezzo di programma stampa dei messaggi di "conto alla rovescia":

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

- L'ultimo messaggio che viene stampato è

`T minus 1 and counting.`

# L'istruzione `while`

- Osservazioni
  - L'espressione di controllo è falsa quando il ciclo termina. Quindi se il ciclo è controllato da `i > 0` quando il ciclo termina, `i` deve essere minore o uguale a 0
  - Il corpo di un `while` potrebbe non essere eseguito se l'espressione di controllo è falsa già al primo controllo
  - Un `while` può essere scritto in molti modi. Ad esempio, una versione più concisa del "conto alla rovescia" è

```
while (i > 0)
 printf("T minus %d and counting\n", i--);
```

# Cicli infiniti

- Un `while` non terminerà mai se l'espressione di controllo non diventa mai 0
- In alcuni casi si scrivono volutamente dei ***cicli infiniti*** usando una costante diversa da zero come espressione di controllo:

  ```
  while (1) …
  ```

- Un'istruzione `while` di questo tipo eseguirà il corpo del ciclo all'infinito, a meno che nel corpo non venga eseguita un'istruzione che trasferisce il controllo all'esterno del ciclo (`break`, `goto`, `return`) oppure che chiama una funzione che "non restituisce il controllo"

# Programma: stampa quadrati

- Il programma `square.c` usa un `while` per stampare una tabella con dei quadrati
- L'utente specifica il numero di righe della tabella:

```
This program prints a table of squares.
Enter number of entries in table: 5
          1         1
          2         4
          3         9
          4        16
          5        25
```

# square.c

```
/* Prints a table of squares using a while statement */

#include <stdio.h>

int main(void)
{
  int i, n;

  printf("This program prints a table of squares.\n");
  printf("Enter number of entries in table: ");
  scanf("%d", &n);

  i = 1;
  while (i <= n) {
    printf("%10d%10d\n", i, i * i);
    i++;
  }

  return 0;
}
```

# Programma: somma di numeri

- Il programma `sum.c` somma una serie di numeri forniti dall'utente:

```
This program sums a series of integers.
Enter integers (0 to terminate): 8 23 71 5 0
The sum is: 107
```

- Il programma usa un ciclo che sfrutta la `scanf` per leggere un numero e somma il numero letto al totale (parziale)

# sum.c

```
/* Sums a series of numbers */

#include <stdio.h>

int main(void)
{
  int n, sum = 0;

  printf("This program sums a series of integers.\n");
  printf("Enter integers (0 to terminate): ");

  scanf("%d", &n);
  while (n != 0) {
    sum += n;
    scanf("%d", &n);
  }
  printf("The sum is: %d\n", sum);

  return 0;
}
```

# L'istruzione do

- La forma generale dell'istruzione `do` è:

  `do` *statement* `while ( expression ) ;`
- Nel ciclo `do` prima si esegue l'istruzione *statement* (il corpo del ciclo) e poi si valuta l'espressione di controllo
- Se il valore dell'espressione di controllo non è zero (cioè, se è *vera*) allora si ripete l'esecuzione del corpo e si valuta nuovamente l'espressione di controllo
- Il ciclo termina quando l'espressione di controllo risulta *falsa*

# L'istruzione do

- L'esempio del conto alla rovescia può essere riscritto usando un ciclo `do`:

```
i = 10;
do {
  printf("T minus %d and counting\n", i);
  --i;
} while (i > 0);
```

- Il ciclo `do` è molto simile al ciclo `while`
  - L'unica differenza è che si esegue prima il corpo e poi si valuta l'espressione di controllo (per la prossima esecuzione del corpo)
- Nel `do` il corpo viene eseguito **almeno una** volta

# L'istruzione `do`

- È buona prassi usare sempre le parentesi graffe per il corpo di un ciclo `do`:

```
do
  printf("T minus %d and counting\n", i--);
while (i > 0);
```

- Un lettore poco attento potrebbe pensare che la parola `while` sia l'inizio di un ciclo `while`

# Programma: numero di cifre in un intero

- Il programma `numdigits.c` calcola il numero di cifre di un intero fornito dall'utente:

```
Enter a nonnegative integer: 60
The number has 2 digit(s).
```

- Il programma dividerà il numero letto dall'input ripetutamente per 10 fino a che il numero diventa 0
  - Il numero di divisioni eseguite è pari al numero di cifre
- Scrivere un tale ciclo con un `do` è preferibile (e non con un `while` ) in quanto qualunque numero intero —anche 0—ha almeno una cifra.

# numdigits.c

```
/* Calculates the number of digits in an integer */

#include <stdio.h>

int main(void)
{
  int digits = 0, n;

  printf("Enter a nonnegative integer: ");
  scanf("%d", &n);

  do {
    n /= 10;
    digits++;
  } while (n > 0);

  printf("The number has %d digit(s).\n", digits);

  return 0;
}
```

# L'istruzione `for`

- L'istruzione `for` è perfetta per cicli che usano una variabile che "conta" il numero di iterazioni
  - Tuttavia è abbastanza versatile da poter essere utilizzato in molte altre situazioni
- La forma generale di un'istruzione `for` è:

  `for ( `*expr1*` ; `*expr2*` ; `*expr3*` )` *statement*

  dove *expr1*, *expr2*, e *expr3* sono espressioni

- Esempio:

```
for (i = 10; i > 0; i--)
  printf("T minus %d and counting\n", i);
```

# L'istruzione `for`

- Il ciclo `for` è simile al ciclo `while`
- Tranne che in rari casi, un ciclo `for` può essere rimpiazzato da un equivalente ciclo `while`:

```
expr1;
while ( expr2 ) {
   statement
   expr3;
}
```

- L'espressione *expr1* viene eseguita una volta soltanto, prima dell'esecuzione del ciclo
  - viene detta "inizializzazione" del ciclo

# L'istruzione `for`

- L'espressione *expr2* è l'espressione di controllo: il corpo del ciclo viene eseguito a patto che l'espressione di controllo sia *vera*
- L'espressione *expr3* è un'operazione che viene eseguita alla fine di *ogni* iterazione del ciclo
- Il `for` dell'esempio precedente può essere quindi scritto come:

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

# L'istruzione `for`

- Guardare all'equivalenza del ciclo `while` con il `for` è utile per capire alcune sottigliezze
- Ad esempio, cosa succede se al posto di `i--` usiamo `--i`?

```
for (i = 10; i > 0; --i)
  printf("T minus %d and counting\n", i);
```

- Analizzando la versione "`while`" si capisce chiaramente che non c'è differenza:

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  --i;
}
```

# L'istruzione `for`

- Poiché la prima e la terza espressione in un ciclo `for` sono di fatto usate come istruzioni, il loro valore è irrilevante
  - Le espressioni sono rilevanti solo per i loro effetti collaterali
- Di conseguenza, solitamente queste due espressioni sono assegnamenti oppure incrementi/decrementi.

# Modi d'uso tipici per il ciclo `for`

- Il ciclo `for` è solitamente la scelta migliore per cicli che "contano" le iterazioni (incrementando o decrementando una variabile)
- Un `for` di `n` iterazioni spesso ha una delle seguenti forme:

***Si conta da* 0 *a* n–1:** `for (i = 0; i < n; i++) …`

***Si conta da* 1 *a* n:** `for (i = 1; i <= n; i++) …`

***Si conta da* n–1 *a* 0:** `for (i = n - 1; i >= 0; i--) …`

***Si conta da* n *a* 1:** `for (i = n; i > 0; i--) …`

# Modi d'uso tipici per il ciclo `for`

- Errori comuni nell'uso del `for`:

- Usare < invece di > (o viceversa) nell'espressione di controllo. Usando un incremento per il conteggio si deve usare < o <=. Usando un decremento si deve usare > o >=
- Sbagliare il conteggio di un'unità:
  - Usare < invece di <= (o viceversa)
  - Usare > invece di >= (o viceversa)
- Usare == nelle espressioni di controllo invece di <, <=, >, or >=

# Omettere espressioni nel `for`

- In C è permesso omettere alcune (o anche tutte) le espressioni di un ciclo `for`
- Se la *prima* espressione è omessa non viene eseguita nessuna inizializzazione prima dell'esecuzione del ciclo

```
i = 10;
for (; i > 0; --i)
  printf("T minus %d and counting\n", i);
```

- Se la *terza* espressione è omessa, allora dovrà essere il corpo del ciclo a far in modo che l'espressione di controllo diventi falsa prima o poi

```
for (i = 10; i > 0;)
  printf("T minus %d and counting\n", i--);
```

# Omettere espressioni nel `for`

- Quando sia la *prima* che la *terza* espressione vengono omesse, il ciclo che ne deriva non è niente altro che un `while` camuffato da `for` :

```
for (; i > 0;)
  printf("T minus %d and counting\n", i--);
```

è equivalente a

```
while (i > 0)
  printf("T minus %d and counting\n", i--);
```

- In questo case la versione `while` è preferibile in quanto più chiara

# Omettere espressioni nel `for`

- Se la *seconda* espressione viene omessa, il suo valore di default è vero, quindi il `for` diventa un ciclo infinito (a meno che il ciclo non venga fermato in qualche altro modo)
- Per esempio, alcuni programmatori usano la seguente istruzione per creare un ciclo infinito:

```
for (;;) …
```

Che è l'analogo di

```
while (1) …
```

# L'istruzione `for` in C99

- Nel C99, la prima espressione (l'inizializzazione) di un ciclo `for` può essere una dichiarazione
- Questa caratteristica permette di dichiarare una variabile da usare come contatore del ciclo

```
for (int i = 0; i < n; i++)
  …
```

- In questo modo non dobbiamo preoccuparci di dichiarare `i` prima di usare il ciclo

# L'istruzione `for` in C99

- Una variabile dichiarata da un ciclo `for` è accessibile (visibile) solo all'interno del ciclo e quindi non può essere usata fuori dal ciclo:

```
for (int i = 0; i < n; i++) {
  …
  printf("%d", i);
  /* legal; i is visible inside loop */
  …
}
printf("%d", i);    /*** WRONG ***/
```

# L'istruzione `for` in C99

- Se il programma ha necessità di accedere al valore della variabile di conteggio alla fine del ciclo, allora è necessario dichiarare la variabile al di fuori del `for`

- È possibile dichiarare più di una variabile, a patto che abbiano tutte lo stesso tipo:

```
for (int i = 0, j = 0; i < n; i++)
  …
```

# L'operatore "virgola"

- In alcuni casi, si ha bisogno di usare più istruzioni nell'espressione di inizializzazione o in quella eseguita alla fine di ogni ciclo
- L'istruzione `for` prevede questo caso e permette l'inserimento di espressioni composte usando come separatore la ***virgola***
  - Vale solo per la prima e la terza espressione
- Quindi, la prima o la terza espressione possono assumere la forma

  *exprA* , *exprB*

  dove *exprA* e *exprB* sono due espressioni

# L'operatore "virgola"

- Un'espressione con virgola viene valutata in più passi:
  - Primo, *exprA* viene valutata ed il suo valore viene ignorato
  - Secondo, *exprB* viene valutata; il suo valore è il valore dell'intera espressione con virgola
- La valutazione di *exprA* dovrebbe avere effetti collaterali, altrimenti non ha senso scriverla
- Quando l'espressione con virgola `++i, i + j` viene valutata, la variabile `i` viene incrementata e dopo viene valutata l'espressione `i + j`
  - Ad esempio se `i` e `j` valgono rispettivamente 1 e 5, il valore dell'espressione totale sarà 7 mentre il valore di `i` sarà 2.

# L'operatore "virgola"

- L'operatore "virgola" è associativo a sinistra, pertanto

  ```
  i = 1, j = 2, k = i + j
  ```

  sarà interpretata come

  ```
  ((i = 1), (j = 2)), (k = (i + j))
  ```

- Poichè l'operando sinistro viene valutato prima dell'operando destro, gli assegnamenti `i = 1`, `j = 2`, e `k = i + j` saranno eseguiti da sinistra a destra

# L'operatore "virgola"

- L'operatore "virgola" permette di "incollare" insieme due espressioni per formarne una sola
  - Il valore dell'espressione risultante è uguale al valore della seconda espressione
- Alcune definizioni di macro sfruttano l'operatore virgola (Capitolo 14)
- Il ciclo `for` è l'unico altro caso in cui l'operatore virgola è utile:
- Esempio:

```
for (sum = 0, i = 1; i <= N; i++)
  sum += i;
```

# Programma: stampa quadrati (rivisitato)

- Il programma `square.c` (visto prima) può essere scritto con un `for` invece che con il `while`

# square2.c

```
/* Prints a table of squares using a for statement */

#include <stdio.h>

int main(void)
{
  int i, n;

  printf("This program prints a table of squares.\n");
  printf("Enter number of entries in table: ");
  scanf("%d", &n);

  for (i = 1; i <= n; i++)
    printf("%10d%10d\n", i, i * i);

  return 0;
}
```

# Programma: stampa quadrati (rivisitato)

- Non ci sono restrizioni sulle 3 espressioni che controllano un ciclo `for`
- Sebbene queste 3 espressioni normalmente siano usate per l'inizializzazione, il controllo e l'aggiornamento della variabile di conteggio, le si può usare in qualsiasi altro modo
  - Il programma `square3.c` è equivalente a `square2.c`, ma il `for` contiene un'inizializzazione che assegna un valore ad una variabile (`square`), un test che ne controlla un'altra (`i`), e un incremento che opera su una terza variabile (`odd`)
- La flessibilità del `for` può essere a volte utile, ma in questo caso la versione originale del programma è più chiara

# square3.c

```
/* Prints a table of squares using an odd method */

#include <stdio.h>

int main(void)
{
  int i, n, odd, square;

  printf("This program prints a table of squares.\n");
  printf("Enter number of entries in table: ");
  scanf("%d", &n);

  i = 1;
  odd = 3;
  for (square = 1; i <= n; odd += 2) {
    printf("%10d%10d\n", i, square);
    ++i;
    square += odd;
  }

  return 0;
}
```

# Uscire da un ciclo

- Di norma si esce da un ciclo quando la condizione di controllo è falsa prima dell'esecuzione del corpo (nel `while` e nel `for`) oppure dopo l'esecuzione del corpo (nel `do`)

- L'istruzione `break` permette di uscire da un ciclo da un qualsiasi punto del corpo

# L'istruzione `break`

- Abbiamo già visto l'uso di `break` per uscire dall'istruzione `switch`. In modo analogo l'istruzione permette di uscire da un ciclo `while`, `do` o `for`

- Ecco un ciclo che controlla se un numero `n` è primo e si ferma usando `break` appena trova un divisore del numero :

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;
```

# L'istruzione **`break`**

- Quando il ciclo termina, un `if` ci permetterà di capire se il ciclo è stato eseguito completamente, cioè non abbiamo trovato divisori (`n` è primo), oppure se è terminato prima (`n` non è primo)

```
if (d < n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

# L'istruzione `break`

- L'istruzione `break` è molto utile nei casi in cui abbiamo bisogno di uscire dal ciclo nel mezzo del corpo del ciclo (piuttosto che all'inizio o alla fine)
- Cicli che leggono un input e terminano quando viene letto un particolare valore sono spesso di questo tipo:

```
for (;;) {
  printf("Enter a number (enter 0 to stop): ");
  scanf("%d", &n);
  if (n == 0)
    break;
  printf("%d cubed is %d\n", n, n * n * n);
}
```

# L'istruzione `break`

- L'istruzione `break` trasferisce il controllo all'istruzione che segue la fine del blocco del `while`, `do`, `for`, o `switch` nel quale il `break` si trova
- Quando queste istruzioni sono nidificate (una all'interno di un'altra) il `break` fa uscire da un solo livello della nidificazione, quello più interno
- Esempio:

```
while (…) {
   switch (…) {
      …
      break;
      …
   }
}
```

- Il `break` fa uscire dallo `switch` ma non dal `while`

# L'istruzione `continue`

- L'istruzione `continue` è simile al `break`:
  - `break` trasferisce il controllo all'istruzione subito dopo la fine del ciclo
  - `continue` trasferisce il controllo alla fine del corpo del ciclo
- Con `break` usciamo fuori dal ciclo; con `continue`, rimaniamo all'interno del ciclo
  - Verrà valutata l'espressione di controllo
- L'istruzione `continue`: non può essere usata in uno `switch` ma solo nei cicli (`while`, `do`, e `for`)

# L'istruzione `continue`

- Un ciclo che usa l'istruzione `continue`:

```
n = 0;
product = 1;
while (n < 10) {
  scanf("%d", &i);
  if (i == 0)
    continue;
  product *= i;
  n++;
  /* continue jumps to here */
}
```

# L'istruzione `continue`

- Lo stesso ciclo senza l'uso di `continue`:

```
n = 0;
product = 1;
while (n < 10) {
  scanf("%d", &i);
  if (i != 0) {
    product *= i;
    n++;
  }
}
```

# L'istruzione `goto`

- L'istruzione `goto` permette di "saltare" (cioè di passare il controllo dell'esecuzione) ad una qualsiasi istruzione del programma, a patto che l'istruzione di destinazione del salto abbia un'***etichetta***
- Un'etichetta è semplicemente un identificatore messo prima dell'istruzione:

  *identifier* **:** *statement*
- L'istruzione `goto` è;

  `goto` *identifier* **;**
- L'esecuzione di `goto` *L*`;` trasferisce il controllo all'istruzione che ha come etichetta *L* (che deve essere nella stessa funzione del `goto`)

# L'istruzione `goto`

- Se il C non avesse avuto l'istruzione `break` si sarebbe potuto usare un `goto` per uscire da un ciclo:

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    goto done;
done:
if (d < n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

# L'istruzione `goto`

- Il `goto` è necessario molto raramente, ed è buona norma non utilizzarlo
- Le istruzioni `break`, `continue` e `return`—che essenzialmente sono dei `goto` con delle restrizioni—e la funzione `exit`, sono sufficienti per gestire la quasi totalità dei casi nei quali è necessario effettuare un "salto"

# L'istruzione goto

- Si consideri il problema di uscire da un ciclo all'interno di uno `switch`
- Il `break` non ha l'effetto desiderato: fa uscire dallo `switch`, ma non dal ciclo `while`
- Un `goto` ottiene l'effetto desiderato:

```
while (…) {
  switch (…) {
    …
    goto loop_done;    /* break won't work here */
    …
  }
}
loop_done: …
```

- Allo stesso modo il `goto` può essere utile per uscire da cicli nidificati

# Programma: controllo del bilancio

- Molti programmi (semplici) offrono all'utente una lista di comandi possibili
- Quando l'utente seleziona un particolare comando, il programma esegue l'azione corrispondente e poi chiede di inserire un nuovo comando
- Questo processo viene ripetuto fino all'esecuzione di un comando di uscita ("exit" o "quit")
- Il cuore del programma è quindi un ciclo del tipo:

```
for (;;) {
   prompt user to enter command;
   read command;
   execute command;
}
```

# Programma: controllo del bilancio

- La scelta di uno dei vari comandi potrà essere fatta grazie ad uno `switch` (o anche un `if` a cascata) :

```
for (;;) {
   prompt user to enter command;
   read command;
   switch (command) {
      case command_1: perform operation_1; break;
      case command_2: perform operation_2; break;
        .
        .
        .
      case command_n: perform operation_n; break;
      default: print error message; break;
   }
}
```

# Programma: controllo del bilancio

- Il programma `checking.c` permette di eseguire il controllo di un bilancio di un conto corrente usando l'approccio descritto prima

- L'utente può azzerare il bilancio, inserire delle somme di accredito e delle somme di addebito, chiedere il valore attuale del bilancio.

- Il comando "exit" permette l'uscita

# Programma: controllo del bilancio

```
*** ACME checkbook-balancing program ***
Commands: 0=clear, 1=credit, 2=debit, 3=balance, 4=exit

Enter command: 1
Enter amount of credit: 1042.56
Enter command: 2
Enter amount of debit: 133.79
Enter command: 1
Enter amount of credit: 1754.32
Enter command: 2
Enter amount of debit: 1400
Enter command: 2
Enter amount of debit: 68
Enter command: 2
Enter amount of debit: 50
Enter command: 3
Current balance: $1145.09
Enter command: 4
```

# checking.c

```
/* Balances a checkbook */

#include <stdio.h>

int main(void)
{
  int cmd;
  float balance = 0.0f, credit, debit;

  printf("*** ACME checkbook-balancing program ***\n");
  printf("Commands: 0=clear, 1=credit, 2=debit, ");
  printf("3=balance, 4=exit\n\n");
  for (;;) {
    printf("Enter command: ");
    scanf("%d", &cmd);
    switch (cmd) {
      case 0:
        balance = 0.0f;
        break;
```

```
        case 1:
          printf("Enter amount of credit: ");
          scanf("%f", &credit);
          balance += credit;
          break;
        case 2:
          printf("Enter amount of debit: ");
          scanf("%f", &debit);
          balance -= debit;
          break;
        case 3:
          printf("Current balance: $%.2f\n", balance);
          break;
        case 4:
          return 0;
        default:
          printf("Commands: 0=clear, 1=credit, 2=debit, ");
          printf("3=balance, 4=exit\n\n");
          break;
      }
    }
}
```

# L'istruzione "nulla"

- Un'istruzione può essere "vuota" o "nulla"—non ci sono simboli a parte il punto e virgola che termina le istruzioni

- Ad esempio questa linea contiene 3 istruzioni
- ```
  i = 0; ; j = 1;
  ```

- L'istruzioni nulla è utile principalmente per una cosa: scrivere cicli il cui corpo è vuoto
  ```

# L'istruzione "nulla"

- Riconsideriamo il programma per controllare se un numero è primo:

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;
```

- Se spostiamo la condizione `n % d == 0` all'interno delle espressioni di controllo il corpo del ciclo diventa nullo:

```
for (d = 2; d < n && n % d != 0; d++)
  /* empty loop body */ ;
```

- Per facilitare la lettura solitamente un'istruzione nulla viene scritta da sola su una linea

# Errori comuni con l'istruzione "nulla"

- Mettere per errore un punto e virgola dopo le parentesi di un `if`, `while`, o `for` crea un'istruzione nulla
- Esempio 1:

```
if (d == 0);                                  /*** WRONG ***/
  printf("Error: Division by zero\n");
```

La `printf` non è nel corpo dell'`if` quindi verrà eseguita indipendentemente dal valore di `d`

- Esempio 2:

```
i = 10;
while (i > 0);                                /*** WRONG ***/
{
  printf("T minus %d and counting\n", i);
  --i;
}
```

L'istruzione nulla crea un ciclo infinito

# Errori comuni con l'istruzione "nulla"

- Esempio 3:

```
i = 11;
while (--i > 0);                    /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```

La `printf` viene eseguita una solo volta (dopo il ciclo):

```
T minus 0 and counting
```

- Esempio 4:

```
for (i = 10; i > 0; i--);           /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```

Stessa situazione e stesso output dell'esempio 3.

… arrivederci alla prossima lezione