Esempio di esecuzione di un programma

Consideriamo il seguente programma.

```
int x = 2;

int g(int a, int b){
  return a + x++;
}


int y = 5;

int f(int a, int x){
  int y = 8;
  x = g(y-x,--a);
  a = g(x,a);
  return a;
}

int a = 0;

int main(){
  int a;
  a = f(5,x++) + 2;
  return 0;
}

int b = 20;
```


Analisi statica del programma

Analizziamo le proprietà statiche del programma, ossia quelle che dipendono dalla sua struttura (e non dalla esecuzione).

Il programma è composto da 7 definizioni: 4 definizioni di variabili globali (`x`, `y`, `a` e `b`) e 3 definizioni di funzioni (`g()`, `f()` e `main()`).

Gli **ambienti** (=insieme dei nomi visibili) del programma sono:

- **Funzione** `g()`

  All'interno di `g()` sono visibili le variabili `a` e `b` locali a `g()` e la variabile globale `x`. Non sono visibili le variabili globali `y`, `a` e `b` in quanto sono dichiarate dopo `g()`. Quindi, se in `g()` si scrive l'istruzione

  ```
  y = 10;
  ```

  si ha errore in compilazione.

- **Funzione** `f()`

  All'interno di `f()` sono visibili le variabili `a`, `x` e `y` locali a `f()`. Non sono visibili le variabili globali `x` e `y` in quanto sono nascoste dalle variabili locali con lo stesso nome. Non sono visibili le variabili globali `a` e `b` in quanto definite dopo (inoltre, la `a` locale a `f()` nasconderebbe l'eventuale definizione di una variabile `a` globale).

- **Funzione** `main()`

  All'interno di `main()` è visibile la variabile locale `a` (che nasconde la variabile globale `a`) e le variabili globali `x` e `y`. Non è visibile la variabile globale `b`.

Nota

Gli ambienti di un programma dipendono esclusivamente dal modo in cui il programma è scritto, non dalla esecuzione, quindi possono essere determinati già in fase di compilazione. Ci sono invece proprietà che possono essere determinate solo in fase di esecuzione.

Ad esempio, il compilatore è in grado di stabilire che il nome `x` in

```
x = g(y-x,--a);
```

si riferisce alla variabile `x` locale a `f()`. Tuttavia, l'indirizzo *assoluto* in memoria di `x` può essere determinato solo in fase di esecuzione e dipende dall'indirizzo in cui è allocato il record di attivazione di `f()` (si tenga presente che una stessa funzione può essere chiamata più di una volta e, in genere, ogni volta il suo record di attivazione sarà in locazioni diverse).

Il compilatore si limita a produrre un indirizzo *logico* (o *relativo*) che determina la posizione di `x` all'interno del record di attivazione di `f()`.

Esecuzione passo-passo

1. All'inizio dell'esecuzione in memoria sono allocate solamente le variabili globali nell'area dati globali, e qui rimangono fino al termine del programma. La configurazione della memoria può essere schematizzata così:

   DATI GLOBALI

x	2
y	5
a	0
b	20

   I nomi `x`, `y`, `a` e `b` denotano *locazioni di memoria*, il cui indirizzo è noto solo in fase di esecuzione.

2. Inizia l'esecuzione della funzione `main()`. Viene allocato in cima allo stack dell'area dati (*push*) il record di attivazione di `main()` che contiene la variabile `a`, il cui valore iniziale non è definito. L'immagine della memoria è:

main()

a	?

DATI GLOBALI

x	2
y	5
a	0
b	20

3. Viene eseguita l'istruzione

```
a = f(5,x++) + 2;
```

che richiede l'esecuzione della chiamata

```
f(5,x++)
```

Passaggio dei parametri

La funzione `main()`, prima di interrompersi, calcola i due valori (*parametri attuali* o *argomenti*) da passare alla funzione `f()`.

Il primo argomento è 5.

Il secondo argomento è il valore di `x++`.
La valutazione di `x++` avviene come in Java: il valore è 2 (attuale valore di `x`) e `x` viene incrementato di 1.

4. Inizia ora l'esecuzione di `f()`. Viene allocato sullo stack (*push*) il record di attivazione di `f()` che contiene le variabili locali `a`, `x` e `y`. Le variabili `a` e `x` (*parametri formali* di `f()`) hanno i valori iniziali passati da `main()`, mentre `y` vale 8 (nel record di attivazione sono contenute altre informazioni non rappresentate nella figura; ad esempio, l'istruzione da eseguire al termine di `f()`).

f()

a	5
x	2
y	8

main()

a	?

DATI GLOBALI

x	3
y	5
a	0
b	20

L'attivazione di una funzione determina un cambio di contesto (*context switch*) in quando cambia l'insieme dei nomi visibili.

5. Inizia l'esecuzione di `f()`. L'istruzione

```
x = g(y-x,--a);
```

provoca la chiamata

```
g(y-x, --a)
```

Vengono calcolati i valori da passare a `g()`.

Il valore dell'espressione `y-x` è 6 (si tenga presente che `x` e `y` si riferiscono alle variabili `x` e `y` locali a `f()`).

Per valutare l'espressione `--a`, occorre prima decrementare il valore di `a` (locale a `f()`); il valore passato come secondo argomento è 4.

6. L'esecuzione di `f()` è sospesa e inizia l'esecuzione di `g()`.

Viene allocato sullo stack il record di attivazione di `g()` in cui i valori iniziali di `a` e `b` sono quelli passati da `f()`.

La memoria è:

g()

a	6
b	4

f()

a	4
x	2
y	8

main()

a	?

DATI GLOBALI

x	3
y	5
a	0
b	20

7. Viene eseguita l'istruzione

```
return a + x++;
```

Occorre calcolare il valore di

```
a + x++
```

Il valore di `a` è 6; il valore di `x++` (dove `x` si riferisce alla variabile `x` globale) è 3 e `x` viene incrementato di 1.

Il valore restituito da `g()` è 9 e `g()` termina l'esecuzione. Prima di togliere dallo stack il record di attivazione di `g()`, viene letto (nello stack) l'indirizzo delle prossima istruzione da eseguire, che è l'istruzione del punto 5 lasciata sospesa.

8. Riprende l'esecuzione di `f()` dall'istruzione

```
x = g(y-x,--a);
```

Viene assegnato a `x` il valore 9 restituito da `g()`.

L'immagine della memoria è:

f()

a	4
x	9
y	8

main()

a	?

DATI GLOBALI

x	4
y	5
a	0
b	20

9. L'esecuzione prosegue con

```
a = g(x,a);
```

Viene effettuata la chiamata `g(x,a)` e allocato di nuovo il record di attivazione di `g()`. I valori passati sono 9 e 4.

g()

a	9
b	4

f()

a	4
x	9
y	8

main()

a	?

DATI GLOBALI

x	4
y	5
a	0
b	20

10. Viene eseguita `g()`. Con l'istruzione

```
return a + x++;
```

viene restituito 13 e incrementato `x` (globale) di uno. Il record di attivazione di `g()` è tolto dallo stack.

11. Riprende l'esecuzione di `f()` dall'istruzione

```
a = g(x,a);
```

in cui ad `a` viene assegnato il valore 13 restituito dalla chiamata a `g()`.

f()

a	13
x	9
y	8

main()

a	?

DATI GLOBALI

x	5
y	5
a	0
b	20

12. Con l'istruzione

```
return a;
```

`f()` restituisce il valore 13 e termina l'escuzione. Il record di attivazione di `f()` è tolto dallo stack.

13. Riprende l'esecuzione di `main()` dal punto in cui era rimasta sospesa (punto 3), ossia da

```
a = f(5,x++) + 2;
```

Ad `a` (locale a `main()`) viene assegnato il valore restituito da `f()` sommato a 2, ossia 15.

main()

a	15

DATI GLOBALI

x	5
y	5
a	0
b	20

14. Termina l'esecuzione di `main()` e viene deallocato il suo record di attivazione.

15. Terminato `main()`, viene liberata anche l'area dati globali e il programma termina.

Esercizi

1. Scrivere il programma precedente aggiungendo delle istruzioni di stampa per visualizzare il contenuto della memoria.

Ad esempio, nella funzione `g()` si può scrivere l'istruzione

```
printf("Funzione g(): a = %d, b = %d, x = %d\n",a,b,x);
```

che produce in output le linee

```
Funzione g(): a = 6, b = 4, x = 3
Funzione g(): a = 9, b = 4, x = 4
```

corrispondenti alle due chiamate di `g()`. Si noti che questi sono gli unici valori che possono essere stampati da `g()`.

2. Eseguire passo-passo il seguente programma

```
int a = 5;
int b = 7;

int f(int x, int y, int z){
  int b = 50;
  b = z-a;
  return b + x;
}

int c = 10;

int g(int x, int y, int z){
  int w;
  x = f(x+y, c++, z-y);
  w = f(++a, x, x);
  return w;
}

int main(){
  int c = 20;
  c = g(c+2, c-2, 30);
  return 0;
}

d = 100;
```

Controllare il risultato inserendo delle istruzioni di stampa.

3. Eseguire passo-passo il seguente programma

```
int a = 10;
int b = 20;

int g(int b){
  return ++a + b;
}

int f(int a, int b){
  int c;
  c = g(++a);
  c = c + g(b++);
  return c;
}

int main(){
    int c = 5;
    b = f(c++,--a);
    return 0;
}
```

Controllare il risultato inserendo delle istruzioni di stampa.