Esempio 1: semplificazione di frazioni

Una frazione positiva $\frac{n}{m}$, con $n \geq 0$ e $m > 0$, è in forma semplificata se e solo se $MCD(n, m) = 1$. Semplificare una frazione $\frac{n}{m}$, con $n \geq 0$ e $m > 0$, significa trovare l'*unica* frazione in forma semplificata equivalente; ad esempio, la forma semplificata di $\frac{20}{12}$ è $\frac{5}{3}$, forma semplificata di $\frac{0}{100}$ è $\frac{0}{1}$.

Scrivere una funzione `simplifyPos()` per semplificare una frazione positiva.

Svolgimento

La funzione deve ricevere in input due interi x e y (numeratore e denominatore della frazione da semplificare) e restituire due interi n e m (numeratore e denominatore della frazione semplificata equivalente a quella data).

Poiché una funzione può restituire al massimo un valore, per comunicare i risultati al chiamante occorre *simulare* una chiamata per indirizzo.

Occorre definire due parametri formali `p` e `q` che devono contenere gli *indirizzi* di due locazioni di memoria di tipo `int` (locazioni in cui sono posti i risultati).

Il tipo di `p` e `q` è `int*` (puntatore a `int`).

Il prototipo della funzione è:

```
void simplifyPos(int x, int y, int* p, int* q)
```

Al termine dell'esecuzione della funzione, numeratore e denominatore della frazione semplificata si trovano rispettivamente nelle locazioni di memoria `*p` e `*q`.

Pertanto è opportuno commentare la funzione come segue:

```
/* Dati  x>=0 e y>0, la frazione (*p)/(*q) e' la forma semplificata
   di x/y    */
```

Non è invece corretto il commento

```
/* Dati  x>=0 e y>0, la frazione p/q e' la forma semplificata
   di x/y    */
```

in quanto `p` non è il *valore* del numeratore, ma l'*indirizzo* della locazione di memoria contenente il numeratore (stesso discorso per `q`).

La forma semplificata $\frac{n}{m}$ della frazione $\frac{x}{y}$, con $x \geq 0$ e $y > 0$, è data da:

$$n = \frac{x}{MCD(x, y)} \qquad m = \frac{y}{MCD(x, y)}$$

Supponendo di aver definito una funzione `mcd()` per il calcolo del MCD di due interi positivi, il codice della funzione è:

```
void simplifyPos(int x, int y, int* p, int* q){
  int d;
  d = mcd(x,y);
  *p = x / d;
  *q = y / d;
}
```

Esempio di chiamata

```
int main(){
 int a = 20, b = 6;
 simplifyPos(a, b, &a, &b);
 ...
}
```

Dopo la chiamata alla funzione `simplifyPos()`, la variabile `a` vale 10, `b` vale 3.

Infatti:

(1). Quando la funzione `main()` inizia l'esecuzione, viene allocato sullo stack il suo record di attivazione; le variabili locali `a` e `b` sono inizializzate rispettivamente a 20 e 6.

main()

a	20
b	6

(2). Viene eseguita la chiamata

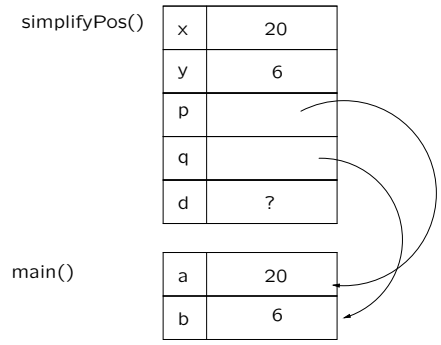
`simplifyPos(a, b, &a, &b)`

Viene allocato il record di attivazione di `simplifyPos()`; i parametri passati sono 20 (*valore* della variabile `a`), 6 (*valore* della variabile `b`), l'*indirizzo* di `a` e l'*indirizzo* di `b`.

Questo significa che, quando `simplifyPos()` inizia la computazione, `*p` è "sinonimo" di `a` (`*p` e `a` sono due nomi per la stessa locazione di memoria), `*q` è "sinonimo" di `b`.

Di conseguenza, `simplifyPos()` può accedere a locazioni di memoria del record di attivazione di `main()` (pur non potendo nominare direttamente le variabili `a` e `b` locali a `main()`).

L'immagine della memoria è:



(3). Inizia l'esecuzione di `simplifyPos()`. Con l'istruzione

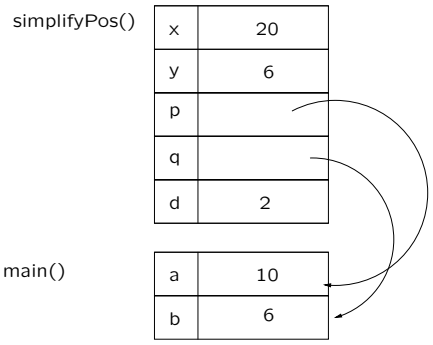
```
d = mcd(x,y);
```

a `d` viene assegnato il valore restituito dalla chiamata alla funzione `mcd()` con argomenti 20 e 6; quando l'assegnamento è completato, `d` vale 2.

(4). Viene ora eseguito l'assegnamento

```
*p = x / d;
```

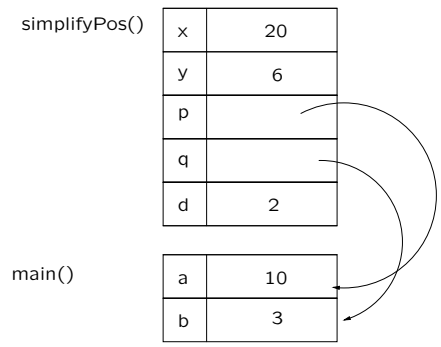
Poiché l'espressione `x/d` vale 10, si ha:



(5). Analogamente, dopo l'istruzione

```
*q = y / d;
```

`*q` (ossia, `b`) vale 3.



(6). La funzione termina, il record di attivazione di `simplifyPos()` è tolto dallo stack e le variabili `a` e `b` risultano modificate nel modo richiesto.

Si noti che il valore iniziale delle variabili `a` e `b` locali a `main()` è perso.

Osservazione

Supponiamo di scrivere

```
simplifyPos(a, b, a, &b);  /* chiamata errata */
```

in cui come terzo argomento viene passato `a` anziché `&a`. Il programma viene compilato ugualmente, segnalando (dipende dal compilatore) il warning

```
warning: passing arg 3 of 'simplifyPos' makes pointer
 from integer without a cast
```

Questo significa che il compilatore introduce implicitamente un'operazione di "cast a `int*`" del terzo parametro.

È come se si fosse scritto

```
simplifyPos(a, b, (int*)a, &b);
```

In esecuzione, la funzione `simplifyPos()` tenta di acceder alla locazione di memoria di indirizzo 20 (valore di `p`).

Poiché in genere un programma non può accedere a tali locazioni, si ha in esecuzione errore di *segmentation fault*.

Per evitare errori di questo tipo (in genere molto frequenti), controllare attentamente i warning segnalati dal compilatore (usare opzione `-Wall` per forzare una maggiore segnalazione di warning).

Esempio 2: swap

Scrivere una funzione `swap1()` che permetta scambiare i valori di due variabili di tipo `int*`.

Svolgimento

Come nell'esempio della funzione `swap()` che scambia due interi, occorre simulare una chiamata per indirizzo.

- Definiamo due parametri `a` e `b` che contengono gli *indirizzi* delle locazioni di memoria (di tipo `int*`) i cui valori vanno scambiati.

- I parametri `a` e `b` devono avere tipo `int**` (puntatore a puntatore a `int`).

```
/* Scambia  *a e *b (valori di tipo int*)  */

void swap1(int **a, int **b){
  int *temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```


Esempio di chiamata

```
int g(){
  int a=5,  b=2, *s = &a,  *t = &b;
  swap1(&s, &t);
}
```

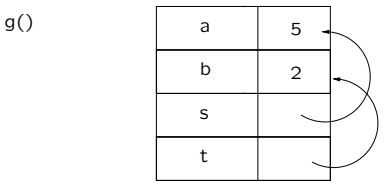
(1). Quando `g()` viene chiamata, il suo record di attivazione è posto sullo stack.

Le variabili locali a `g()` sono:

- `a` e `b` di tipo `int`;

- `s` e `t` di tipo `int*`.

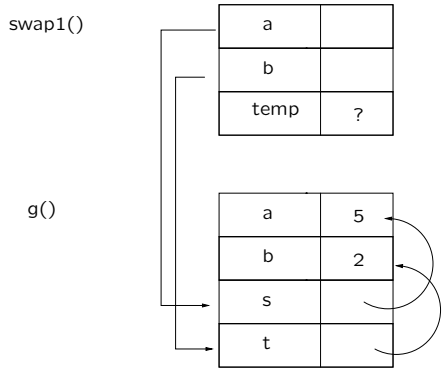
I valori iniziali sono quelli mostrati nella figura.



(2). Inizia l'esecuzione di `g()` con la chiamata

```
swap1(&s, &t);
```

Viene posto sullo stack il record di attivazione di `swap1()` e vengono passati come parametri gli indirizzi delle variabili `s` e `t` locali a `g()`.



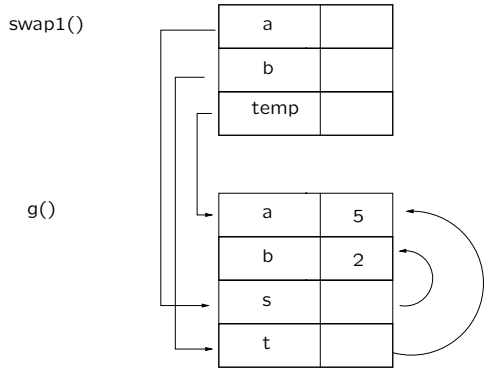
(3). Inizia l'esecuzione di `swap1()`. L'espressione `*a` è sinonimo di `s`, `*b` di `t`.

Ad esempio

```
temp = *a;
```

pone in `temp` il *valore* di `s`, ossia l'indirizzo di `a` locale a `g()`.

Al termine dell'esecuzione di `swap1()` si ha:



I valori di `s` e `t` locali a `g()` risultano scambiati.

Esercizi

1. Data la funzione `main()`

```
int main(){
  int a=1, b=2, c=3, *p=&a, *q=&b, *r=&c;
  swap1(&p,&q);
  swap(p,&c);
  swap1(&r,&q);
  swap(r,p);
  return 0;
}
```

dire qual è il valore delle variabili locali a `main()` dopo ogni chiamata (le chiamate sono corrette).

2. Si consideri il programma

```
int main(){
  int a=5, b=2;
  swap(&a, &b);
  return 0;
}
```

Inserire nel programma delle istruzioni per stampare gli indirizzi e i valori di tutte le variabili Per stampare l'indirizzo di `a` si può usare l'istruzione

```
printf("&a = %u\n", (unsigned int) &a);
```

dove viene effettuato un cast di `&a` al tipo `unsigned int` (intero senza segno), che permette di rappresentare in modo corretto numeri positivi molto grandi.