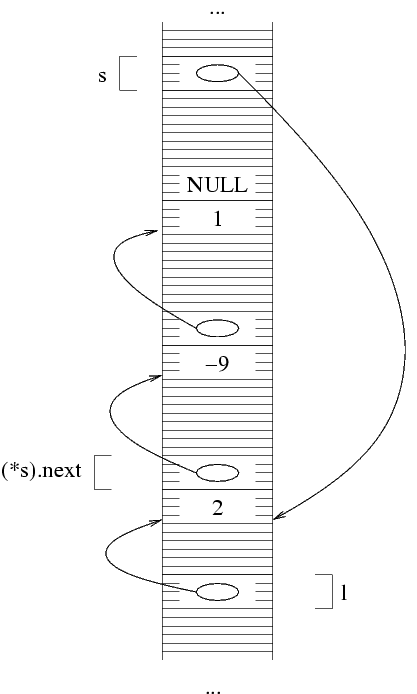
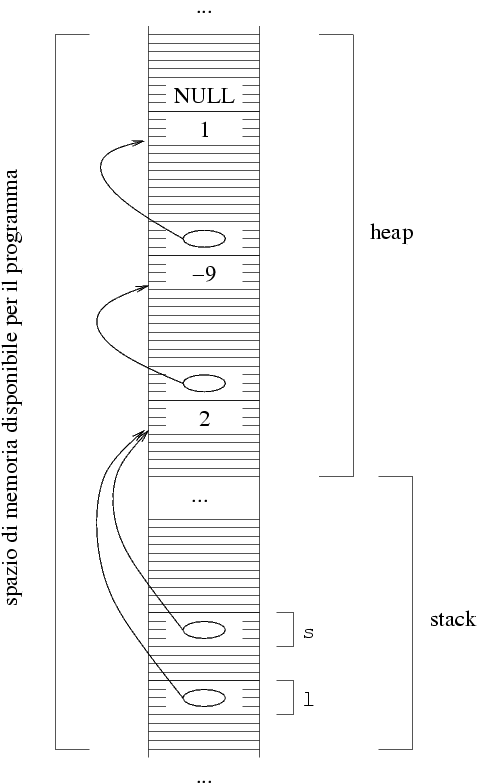
**Stack, heap e record di attivazione**

Finora abbiamo disegnato la memoria senza fare distinzioni fra le variabili e le zone di memoria allocate con malloc. Un esempio di figura fatta in questo modo è la seguente:



In realtà, le variabili del programma stanno in una zona di memoria detta *stack*, mentre le zone di memoria allocate con malloc stanno in una diversa zona, detta *heap*.



La struttura dello heap non ci interessa. Ci basta sapere che questa è la zona che contiene tutte le zone di memoria allocate con malloc. È invece importante capire in che modo è fatto lo stack. La struttura è la seguente: per ogni funzione, c'è una zona di memoria in cui si trovano le variabili locali della funzione. In particolare, la zona di memoria per la funzione main (ossia quella che contiene le variabili del programma principale) si trova nel punto più basso dello stack. Ogni volta che si chiama una funzione, viene creata una zona di memoria immediatamente sopra, e questa zona contiene le variabili locali della funzione chiamata (incluse le variabili che rappresentano i parametri). La zona di memoria che viene creata per una singola funzione viene detta *record di attivazione*. Consideriamo il seguente esempio:

void Esempio(int x) {

int y;

}

void Altra(float m) {

}

int main() {

float f=-12.3;

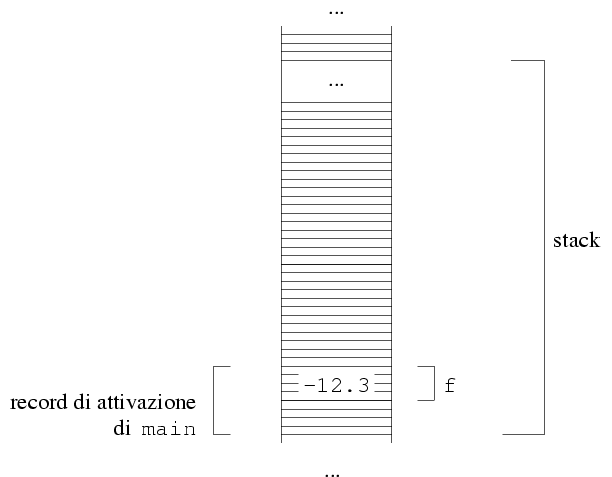
Esempio(90);

Altra(f);

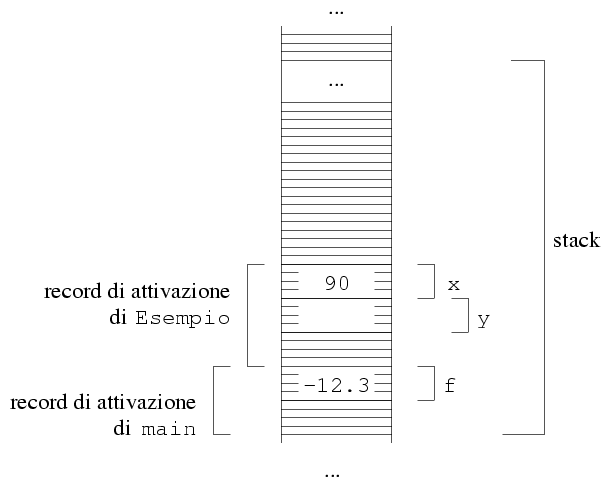
return 0;

}

Quando viene lanciata la funzione main, ossia quando si esegue il programma, si crea il primo record di attivazione, che è una zona di memoria che contiene le variabili locali del main.



Quando si esegue anche la funzione Esempio, viene creato un altro record di attivazione, che viene messo immediatamente sopra. Questo record di attivazione contiene sia le variabili che rappresentano i parametri formali della funzione (in cui vengono immediatamente scritti i valori passati) che le variabili locali del programma.

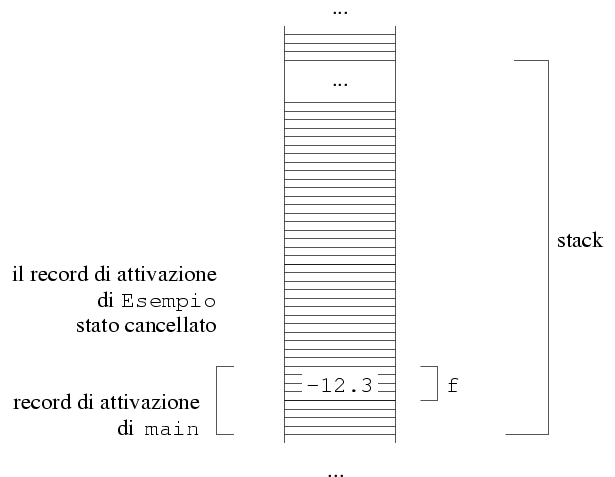


I record di attivazione contengono anche altre cose, ma questo per il momento non ci interessa.

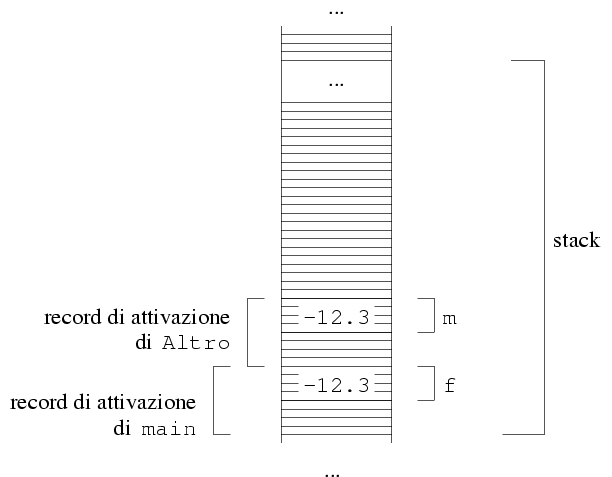
La regola precisa, per i record di attivazione, è:

ogni volta che viene chiamata una funzione, si crea per essa un nuovo record di attivazione, che viene messo immediatamente sopra a quelli già presenti nello stack.

Ogni volta che una funzione termina, il suo record di attivazione viene cancellato. Nel caso di sopra, quando la funzione Esempio, termina, il record di attivazione che era stato creato per essa viene rimosso dallo stack:



Quando si chiama Altra, viene creato un nuovo record di attivazione, che si trova nella posizione più bassa libera dello stack. Dal momento che il record di attivazione di Esempio è stato cancellato, il nuovo record viene messo subito sopra quello di main.



Quando anche questa funzione termina, il suo record di attivazione viene cancellato. Il motivo della cancellazione è che le variabili locali non servono più, per cui si può liberare la memoria, che può quindi venire riusata per creare dei record di attivazione di altre funzioni (nell'esempio di sopra, parte della memoria del record di attivazione di Esempio veniva riusato per creare il record di attivazione di Altro).

Vediamo ora cosa succede quando si fanno chiamate a funzioni ricorsive. La regola da usare è sempre la stessa, ossia:

* quando di chiama una funzione, si crea per essa un record di attivazione nel punto più basso libero dello stack;
* questo record di attivazione è una zona di memoria che contiene i suoi parametri formali e le sue variabili locali;
* quando la funzione termina, il record di attivazione viene cancellato dallo stack, liberando quindi la memoria per successivi record di attivazione.

Si tratta solo di applicare questa regola. Supponiamo quindi che la funzione Esempio contenga una chiamata ricorsiva:

void Esempio(int x) {

int y;

Esempio(x-1);

}

int main() {

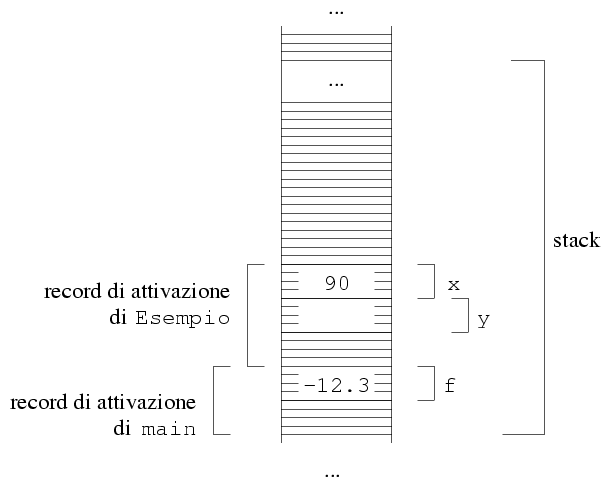
float f=-12.3;

Esempio(90);

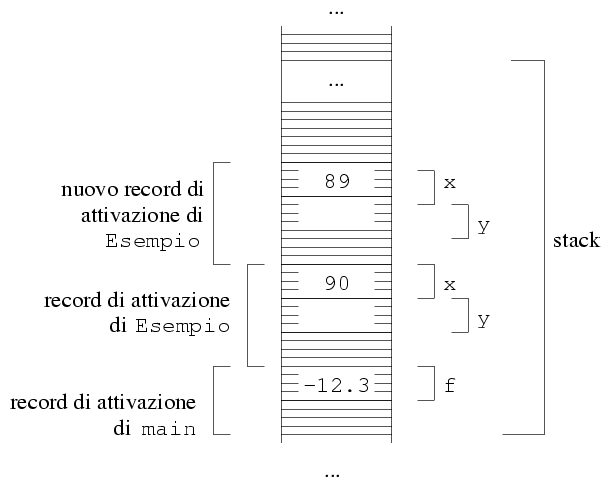
return 0;

}

Alla prima chiamata, si crea il record di attivazione per Esempio, per cui siamo nella solita situazione:

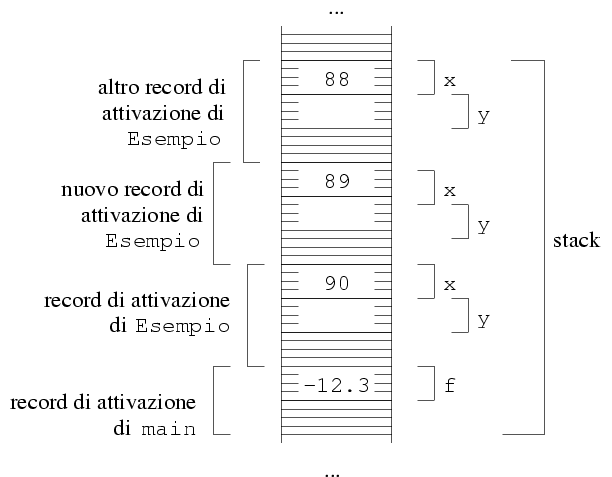


Viene fatta la chiamata ricorsiva. Come al solito, la regola va interpretata alla lettera: in particolare, la regola dice che si deve creare un nuovo record di attivazione. Dal momento che la chiamata ricorsiva avviene passando x-1, e che x vale 90, il nuovo record di attivazione contiene 89 nella variabile x.



È importante notare come le variabili di Esempio siano state duplicate. In particolare, ogni record di attivazione contiene una sua versione di queste variabili. Si noti anche che le diverse variabili possono anche contenere valori differenti.

A questo punto si fa la terza chiamata. Si passa x-1. Quando vale x? In memoria abbiamo due versioni della variabile x: una che contiene 90 e una che contiene 89. La regola, in questo caso, è che *in ogni esecuzione di una funzione, essa usa le variabili che si trovano nel record di attivazione che è stato creato per essa*. Dal momento che il record di attivazione che è stato creato per questa esecuzione di Esempio è quello più in alto dello stack, la x è che si usa è quella che contiene 89. Quindi x-1 vale 88, ed è questo il valore che viene passato nella chiamata ricorsiva.



Nella nuova variabile x viene messo il valore che è stato passato, ossia 88. La cosa va avanti cosí: ogni volta che si chiama una funzione, si crea per essa un nuovo record di attivazione che viene messo subito sopra i precedenti. Ogni volta che si attiva una funzione, questa ``vede'' le variabili che stanno scritte nel suo record di attivazioni. Per esempio, la prima attivazione di Esempio vede la x che contiene 90, la seconda vede la x che contiene 89, ecc.

In questo esempio, la funzione Esempio continua a richiamare se stessa fino a riempire lo stack con i suoi record di attivazione. Questo è un punto essenziale, quando si creano funzioni ricorsive: *prima o poi, la funzione deve smettere di richiamare se stessa*. Questo è un punto che è facile scordare.

Una prima verifica che occorre fare è quella di vedere se esiste una possibile esecuzione della funzione su cui non viene fatta la chiamata ricorsiva. In altre parole, devono esistere dei dati sui quali la funzione non chiama se stessa. Per esempio, la funzione di sopra non termina mai, perchè per qualsiasi valore del suo argomento x, viene comunque fatta una chiamata ricorsiva. Una alternativa possibile è la seguente:

void Esempio(int x) {

int y;

if(x==0)

return;

Esempio(x-1);

}

int main() {

float f=-12.3;

Esempio(90);

return 0;

}

Qui si vede chiaramente che c;è un ``punto di uscita'' dalla funzione, ossia un percorso che non contiene la chiamata ricorsiva: se x vale 0, allora si esce dalla funzione senza fare la chiamata alla funzione. Lo stack evolve in questo modo: si chiama Esempio con il valore 90, e questa richiama se stessa con il valore 89, che richiama ricorsivamente con il valore 88, ecc, fino a che si arriva a una chiamata con il valore 0. A questo punto, non si fanno chiamate ricorsive, ma si esce dalla funzione.

La regola, quando si esce dalla funzione, è semplice: si toglie il record di attivazione, e *si passa ad eseguire il resto della funzione chiamante*. In altre parole, ogni funzione viene attivata solo quando un'altra funzione contiene una chiamata ad essa (questo vale per tutte le funzioni, non solo quelle ricorsive). Alla fine della esecuzione, si deve eseguire quello che resta della funzione chiamante.

Nel nostro caso, la funzione con x=1 chiama la funzione con x=0: quando questa è finita, occorre finire di eseguire la funzione con x=1. Nel nostro caso, la chiamata ricorsiva è l'ultima, per cui la funzione con x=1 termina, e si torna alla funzione precedente, che è ancora la funzione Esempio, ma quella con x=2. È chiaro che quello che succede è che tutte le attivazioni della funzione, dato che sono già arrivate alla fine, terminano e i loro record di attivazione vengono tutti cancellati. Nella prossima pagine si vede un caso in cui questo non è vero.

Per concludere, va notato come quest'ultima funzione, anche se ha un punto di uscita, non necessariamente termina. Se per esempio si fa una chiamata Esempio(-12), la chiamata ricorsiva avviene con un valore di x pari a -13, poi -14, ecc, e non si arriva mai a 0. Questo è un altro punto importante sulla uscita della ricorsione: non solo ci deve essere un possibile punto di uscita, ma deve anche essere possibile dimostrare che, qualsiasi siano i parametri con cui la funzione viene chiamata, si arriva prima o poi a una attivazione in cui non si fa una chiamata ricorsiva. Nel nostro caso, una modifica che sicuramente termina è la seguente.

void Esempio(int x) {

int y;

if(x<=0)

return;

Esempio(x-1);

}

int main() {

float f=-12.3;

Esempio(90);

return 0;

}