9.5 Oggetti dinamici

I puntatori sono usati nella creazione e manipolazione di oggetti dinamici. Mentre gli oggetti statici vengono creati specificandoli in una definizione, gli oggetti dinamici sono creati durante l'esecuzione del programma. Il numero degli oggetti dinamici non è definito dal testo del programma, come per gli oggetti creati attraverso una definizione: essi vengono creati o distrutti durante l'esecuzione del programma, non durante la compilazione. Gli oggetti dinamici, inoltre, non hanno un nome esplicito, ma occorre fare riferimento a essi per mezzo di puntatori.

Il valore `NULL`, che può essere assegnato a qualsiasi tipo di puntatore, indica che nessun oggetto è puntato da quel puntatore. È un errore usare questo valore in riferimento a un oggetto dinamico.

Il puntatore `NULL` è un indirizzo di memoria che corrisponde al valore convenzionale di puntatore che non punta a nulla e la sua definizione può essere diversa da macchina a macchina. Per esempio:

```
#include <stdio.h>

main()
{
  char *p;
  ...
  p = NULL
  ...
  if (p != NULL {
  ...
  }
  else {
  ...
  }
}
```

Il valore di puntatore nullo `NULL` è una costante universale che si applica a qualsiasi tipo di puntatore (puntatore a `char`, a `int` ecc.). Generalmente la sua definizione è:

```
#define NULL 0
```

ed è contenuta in `<stdio.h>`.

Come detto, in C la memoria è allocata dinamicamente per mezzo delle funzioni di allocazione `malloc` e `calloc` che hanno le seguenti specifiche:

```
void *malloc(int num);  /* num: quantità di memoria da allocare */

void *calloc(int numele, int eledim);
                        /* numele: numero di elementi;  eledim:
                           quantità di memoria per ogni elemento */
```

Sia `malloc` sia `calloc` ritornano un puntatore a carattere che punta alla memoria allocata. Se l'allocazione di memoria non ha successo – o perché non c'è memoria sufficiente, o perché si sono passati dei parametri sbagliati – le funzioni ritornano il puntatore `NULL`.
Per stabilire la quantità di memoria da allocare è molto spesso utile usare l'operatore `sizeof`, che si applica nel modo seguente:

```
sizeof( espressione )
```

nel qual caso restituisce la quantità di memoria richiesta per memorizzare *`espressione`*, oppure

```
sizeof( T )
```

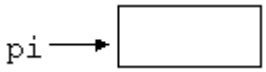
nel qual caso restituisce la quantità di memoria richiesta per valori di tipo *`T`*. Vediamo un esempio:

```
main()
{
  char a[10];
  int i;

  i = sizeof(a[10]);
  printf("L'array %s ha dimensione = %d\n", a, i);

  i = sizeof(int);
  printf("Gli interi hanno dimensione %d", i);
}
```

L'operatore `sizeof` ritorna un intero maggiore di zero corrispondente al numero di `char` che formano l'espressione o il tipo. Si ricordi che l'unità di misura di `sizeof` è il `char` e non il byte, come potrebbe venire naturale pensare.
Gli allocatori `malloc` e `calloc` ritornano un puntatore all'oggetto dinamico creato. In realtà, gli allocatori ritornano dei puntatori a `void`, cioè a tipo generico, e perciò devono essere esplicitamente convertiti in un tipo specifico. Il valore che ritorna dagli allocatori di memoria è molto importante perché è solo attraverso di esso che si può far riferimento agli oggetti dinamici.
Consideriamo per esempio l'istruzione:

```
pi = (int *) malloc (sizeof(int));
```

che alloca una quantità di memoria sufficiente per accogliere un intero. Questo intero, allocato dinamicamente e di cui non si conosce il nome, può essere raggiunto per mezzo del puntatore `pi`. L'indirizzo dell'intero è assegnato a `pi` dopo aver esplicitamente convertito il tipo `void *`, ritornato `malloc`, nel tipo `int *`, il tipo della variabile `pi`, mediante la semplice espressione (`int *`) detta *cast*. Graficamente l'oggetto dinamico puntato da `pi` potrebbe essere rappresentato come segue.



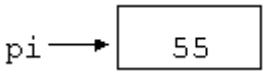
La scatola vuota simboleggia lo spazio riservato dall'intero. Per poter utilizzare le funzioni di allocazione è necessario includere la libreria `malloc.h` e/o `stdlib.h`; in implementazioni del C meno recenti la libreria da includere è stddef.h:

```
#include <malloc.h>
#include <stdlib.h>
```

Si accede a un oggetto dinamico tramite un puntatore e l'operatore di indirezione. Così, nell'esempio, si accede all'intero puntato da `pi` con il nome `*pi`. Per esempio:

```
*pi = 55;
```

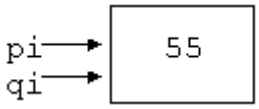
Graficamente l'effetto della precedente assegnazione può essere rappresentato come segue.



Lo stesso valore del puntatore può essere assegnato a più di una variabile puntatore. In tal modo si può far riferimento a un oggetto dinamico con più di un puntatore. Un oggetto cui si fa riferimento con due o più puntatori possiede degli *alias*. Per esempio, il risultato dell'assegnazione

```
qi = pi;
```

è di creare due puntatori allo stesso oggetto, cioè due alias. Graficamente l'effetto della precedente assegnazione può essere rappresentato come segue.



✓ **NOTA**
Un uso smodato degli alias può deteriorare la leggibilità di un programma. Il fatto di accedere lo stesso oggetto con puntatori diversi può rendere difficoltosa l'analisi locale del programma, cioè la lettura di una porzione di codice senza avere in testa il tutto.

Gli oggetti dinamici devono essere esplicitamente deallocati dalla memoria se si vuole recuperare dello spazio. La deallocazione esplicita dello spazio di memoria avviene con la funzione `free`, così specificata:

```
free( char * ptr )
```

Se non si effettua questa operazione lo spazio di memoria verrà perso, cioè non sarà possibile riutilizzarlo. Per esempio:

```
free( pi );
```

libera la memoria occupata dall'intero puntato da `pi`. Occorre prestare molta attenzione a evitare errori del tipo: "fare riferimento a un oggetto che è già stato deallocato". Alcuni linguaggi, come il Lisp, lo Snobol, il Perl e Java, hanno dei meccanismi automatici di recupero della memoria detti "spazzini" (*garbage collector*) . In C il programmatore deve raccogliere la "spazzatura" da solo ■.