12.3 Unioni

Le *unioni* (`union`) sono analoghe alle strutture: introducono nel programma una nuova definizione di tipo e sono costituite da un insieme di membri, che possono essere – in generale – di tipo e dimensione diversa. I membri di una unione, però – a differenza di una struttura –, coincidono, cioè hanno lo stesso indirizzo e vanno a occupare le medesime locazioni di memoria. Questo implica che l'occupazione di memoria di una unione coincide con quella del membro dell'unione di dimensione maggiore. La sintassi generale di una `union` è analoga a quella delle strutture:

```
union nome_unione {
        tipo_membro nome_membro1;
        tipo_membro nome_membro2;
        ...
        tipo_membro nome_membroN;
```

```
};
```

Anche le `union` sono nuovi tipi che vengono introdotti in un programma, e le variabili dichiarate di tipo `union` possono, in tempi diversi, assumere oggetti di tipo e dimensione differenti in accordo alle specifiche dei membri dell'unione. Nell'esempio

```
union fantasma {
        int i;
        long d;
        char c;
};
union fantasma a;
```

la dimensione della variabile `a` coincide con quella della unione `fantasma` e corrisponde allo spazio occupato da un `long` (in alcune implementazioni 4 byte). Si noti come in base al tipo del membro si abbia una diversa allocazione della variabile `a` (Figura 12.1).

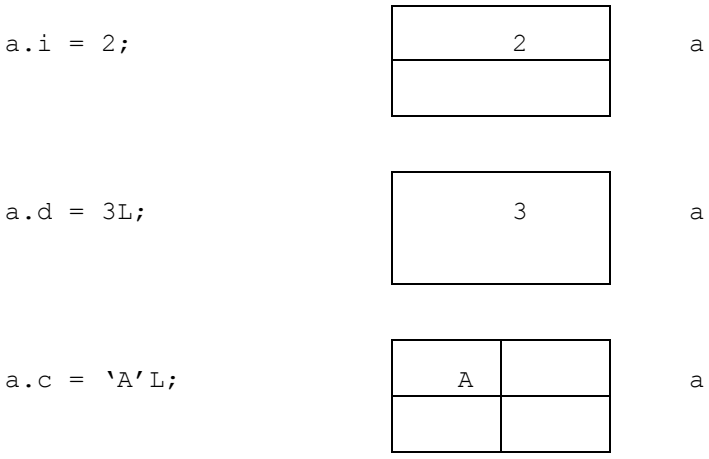

Figura 12.1 Rappresentazione di una unione

Le unioni vengono preferite alle strutture quando si hanno delle variabili che possono assumere più tipi e/o dimensioni diverse a seconda delle circostanze. Per esempio, supponiamo di voler rappresentare per mezzo di una struttura una tavola di simboli, cioè una tabella i cui elementi siano una coppia `<nome, valore>` e dove il valore associato a un nome può essere o una stringa o un numero intero. Una possibile struttura per la tavola dei simboli potrebbe essere:

```
struct tav_sim {
        char *nome;
        char tipo;
        char *val_str; /* usato se tipo ==  's'  */
        char val_int;  /* usato se tipo ==  'i'  */
    };
    void stampa_voce(struct tav_sim *p)
    {
        switch (p->tipo) {
        case  's':
            printf("%S", p->val_str);
            break;
        case  'i':
            printf("%d", p->val_int);
            break;
        default:
            printf("valore di tipo non corretto\n");
```

```
            break;
        }
    }
```

La funzione `stampa_voce` ha lo scopo di visualizzare la decodifica di una voce della tabella, voce che corrisponde a una variabile di tipo `tav_sim`, passata alla funzione `stampa_voce` tramite un puntatore `p`. Si osserva però che una voce della tavola dei simboli, a seconda dell'attributo tipo, ha una decodifica di tipo stringa o di tipo intero, ma mai entrambe. La struttura che si è proposta è allora ridondante, cioè comporta un'occupazione di memoria superiore a quella richiesta. È in situazioni come questa, allora, che si fa ricorso alle unioni. I due attributi `val_str` e `val_int` si trattano come membri di una unione:

```
struct tav_sim {
        char *nome;
        char tipo;
        union {
           char *val_str; /*usato se tipo ==  's'  */
           int  val_int; /* usato se tipo ==  'i'  */
        };
    }
```

Con questa soluzione i programmi che inseriscono, cancellano, ricercano o stampano una voce della tavola dei simboli rimangono invariati, e per mezzo del concetto di unione si ha che i due attributi `val_str` e `val_int` hanno lo stesso indirizzo, ovvero sono allocati nella medesima area di memoria.

Talvolta le unioni sono usate anche per effettuare conversioni di tipo. Questa pratica può essere fonte di ambiguità e quindi di errore. Il lettore si attenga a quanto detto a proposito delle conversioni implicite ed esplicite di tipo.