

**OBIETTIVO CORRETTEZZA: specificazione completa con precondizioni e postcondizioni**

- Le **precondizioni** e **postcondizioni** aggiunte al prototipo, consentono di fornire una specificazione precisa della funzione
- Le **precondizioni** stabiliscono le eventuali restrizioni di definizione della funzione sull'insieme dei dati in input (dato dal tipo dei parametri) il chiamante deve assicurarsi di chiamare la funzione solo se le precondizioni sono soddisfatte, infatti se una precondizione non è rispettata al momento della chiamata, la funzione può comportarsi in modo imprevedibile.
- Le **postcondizioni** specificano l'output della funzione.

Se il chiamante ha rispettato le precondizioni, la funzione deve rispettare le postcondizioni, cioè restituire i valori in conformità con le postcondizioni.

**OBIETTIVO CORRETTEZZA: specificazione completa con precondizioni e postcondizioni**

**Esempio:** sia  $L$  una collezione e  $x$  un elemento nell'universo di definizione di  $L$ , consideriamo la funzione

$$\text{cerca}(L,x) = \begin{cases} 1 & \text{se } \exists x x \in L \text{ (in genere abbreviato se } x \in L) \\ 0 & \text{altrimenti} \end{cases}$$

Qual'è il risultato di **cerca** se  $L = \emptyset$ ?

Dal punto di vista logico **cerca** $(\emptyset,x) = 0$ ,  
infatti  $\exists x x \in \emptyset$  è evidentemente falsa

**OBIETTIVO CORRETTEZZA: specificazione attraverso precondizioni e postcondizioni**

Supponendo di aver rappresentato in memoria la collezione come una lista, l'implementazione è:

“cerca” è una funzione totale

```
int cerca(Listptr L, Elem el)
```

```
/* verifica se l'elemento el occorre almeno una volta nella collezione
```

```
*prec: nessuna
```

```
*postc: restituisce 1 se l'elemento occorre nella lista, 0 altrimenti */
```

```
{if (!L) return 0;
```

```
if(L->elem == el) return 1;
```

```
return cerca(L->nextPtr, el); }
```

Qui si fa riferimento al seguente tipo

```
struct nodo {
```

```
int elem;
```

```
struct nodo * next;
```

```
};
```

```
typedef struct nodo Lista;
```

```
typedef Lista* ListaPtr;
```

**CONVENZIONE:**

Ometteremo il riferimento alle precondizioni se non ce ne sono

**OBIETTIVO CORRETTEZZA: specificazione attraverso precondizioni e postcondizioni**

**Ma in qualche caso è necessario distinguere il caso di ricerca con insuccesso dal caso dell'insieme vuoto.**

**La prima soluzione proposta esclude l'insieme vuoto come argomento della funzione:**

```
int cerca1(Listptr L, Elem el)
/* verifica se l'elemento el occorre almeno una volta nella collezione
*prec: L != NULL
*postc: restituisce 1 se l'elemento occorre nella lista, 0 altrimenti */

{if (L->elem == el) return 1;
if (L -> nextPtr) return cerca1(L->nextPtr, el);
/* Non ci devono essere chiamate su L==NULL */
else return 0; }
```

“cerca1” è una funzione parziale

**OBIETTIVO CORRETTEZZA: specificazione attraverso precondizioni e postcondizioni**

La seconda soluzione proposta distingue in output i due casi:

```
int cerca2(Listptr L, Elem el)
```

```
/* verifica se l'elemento el occorre (almeno una volta) nella collezione  
*postc: se  $L = \emptyset$  restituisce -1, altrimenti se l'elemento e' presente  
restituisce 1 e in caso contrario 0.*
```

```
{if (!L) return -1;
```

```
if (L->elem == el) return 1;
```

```
if (L -> nextPtr) return cerca2(L->nextPtr, el);
```

```
else return 0;
```

```
}
```

“cerca2” è una funzione totale

## **OBIETTIVO CORRETTEZZA: specificazione attraverso precondizioni e postcondizioni**

**E' importante capire che le precondizioni **non** richiedono alcuna verifica nel codice della funzione, anzi spesso sono introdotte proprio per evitare il controllo di condizioni troppo onerose da verificare.**

**Ovviamente una precondizione non banale rende la funzione parziale e questo può essere un inconveniente per il chiamante che deve assicurarsi di non chiamare la funzione su valori che non soddisfano le precondizioni.**

**La decisione di utilizzare le precondizioni per gestire casi particolari di dati in input deve essere determinata**

- dall'ambito di applicazione della funzione. Se si prevede un uso solo locale di un'applicazione, si può pensare di verificare al momento della chiamata che le precondizioni siano rispettate, se la funzione ha un uso più ampio, bisogna fare un uso molto prudente delle precondizioni.**
- dal costo dei controlli. Esempio: la ricerca binaria opera correttamente solo su elementi ordinati, ma se controlliamo questo al momento della chiamata della ricerca binaria perdiamo tutto il vantaggio in termini di prestazioni della ricerca binaria!**