8 - Funzioni e procedure

Introduzione alla Programmazione Modulare

La programmazione modulare rappresenta un approccio che suddivide un programma in frammenti di codice più piccoli, noti come funzioni o procedure. Questi frammenti sono separati dal programma principale e consentono una gestione più efficace della complessità, migliorando leggibilità, astrazione, riusabilità e modularità. Questo approccio è particolarmente utile perché affronta problemi come la difficoltà di tracciare errori logici, lavorare in team e riutilizzare porzioni di codice in diversi progetti.

Astrazione e Funzioni

Le funzioni e le procedure sono meccanismi di astrazione, utilizzati per estendere operatori e istruzioni disponibili nel linguaggio. Ad esempio, una funzione può calcolare un valore (come il BMI) con una sintassi più leggibile rispetto a includere i calcoli direttamente nel programma principale.

Principi della Programmazione Modulare

Un programma modulare deve:

1. Essere suddiviso in moduli indipendenti.
2. Permettere il test autonomo di ogni modulo.
3. Definire chiaramente input e output di ciascun modulo.
   La modularità è resa possibile in C tramite funzioni e procedure, consentendo l'organizzazione del codice in blocchi con ruoli specifici, come acquisizione di input, elaborazione e output.

Funzioni e Procedure: Struttura e Utilizzo

Una funzione in C è definita da quattro elementi fondamentali:

1. **Nome**: Identificativo unico utilizzato per richiamare la funzione.
2. **Parametri**: Variabili necessarie per eseguire il compito della funzione.
3. **Valore di Ritorno**: Risultato restituito al termine dell'esecuzione.
4. **Implementazione**: Insieme di istruzioni eseguite dalla funzione.
   I primi tre elementi costituiscono il **prototipo** della funzione, che deve essere dichiarato prima della sua implementazione per migliorare leggibilità e garantire correttezza nella chiamata.

Sintassi delle Funzioni

La struttura base di una funzione è:

```
<tipo_di_ritorno> <nome_della_funzione>(<parametri>) {
   <implementazione>
}
```

Ad esempio, la funzione `promosso` verifica se un voto supera una soglia e restituisce 1 se la condizione è soddisfatta, 0 altrimenti:

```
int promosso(int voto, int soglia) {
    return voto > soglia ? 1 : 0;
}
```

Passaggio dei Parametri

In C esistono due modalità principali di passaggio dei parametri:

1. **Per Valore**: Viene creata una copia del valore del parametro attuale; le modifiche effettuate dalla funzione non influenzano il valore originale.
2. **Per Riferimento**: Il parametro formale punta alla stessa locazione di memoria del parametro attuale, rendendo persistenti le modifiche.

Nel passaggio per valore, le modifiche sono locali alla funzione, mentre nel passaggio per riferimento, come nell'esempio con puntatori, le modifiche si riflettono sul chiamante.

Esempio di passaggio per riferimento:

```
void quadrato(int* y) {
    *y = (*y) * (*y);
}
```

Chiamata nel `main`:

```
int x = 5;
quadrato(&x);
```

Differenza tra Funzioni e Procedure

In C, una procedura è una funzione che non restituisce alcun valore, dichiarata con la parola chiave `void`. Questo tipo di funzione esegue azioni senza produrre un risultato da memorizzare.
Esempio:

```c
void stampaNumero(int numero) {
    printf("%d", numero);
}
```

Passaggio di Array

Gli array in C vengono passati per riferimento, permettendo alle modifiche effettuate all'interno della funzione di essere visibili anche nel programma principale. È importante passare anche la dimensione dell’array.

```c
void quadrato(int x[], int n) {
    for (int i = 0; i < n; i++) {
        x[i] *= x[i];
    }
}
```