

Sommario

Chiamata di una procedura.....	3
Ambito di visibilità delle variabili.....	4
Funzioni.....	4
Conclusione	8

Impareremo in questo modulo

- Chiamata ad una procedura
- Ambito di visibilità delle variabili
- Funzioni
- Ricorsione

Modulo 4 - Procedure e funzioni

Spesso, creiamo algoritmi che vogliamo riutilizzare in varie parti del codice o voglia suddividere gli algoritmi in pezzi più piccoli per ridurre la complessità e facilitare la comprensione del programma. Per questo motivo utilizziamo procedure e funzioni.

Procedure

Una procedura può essere vista come un programma di dimensioni ridotte che può essere chiamato all'interno del programma principale o anche all'interno di un'altra procedura. La struttura di una procedura è illustrata di seguito:

```
PROCEDURE: <nome della procedura> ([<nome del parametro>: <tipo>] *)  
  
VARIABILI  
  
    {dichiarazione delle variabili}  
  
INIZIO  
  
    (operazioni);  
  
FINE;
```

I parametri sono variabili che possono avere i cui valori possono essere già definiti quando chiamiamo una procedura. Essi hanno un nome e un tipo e se ne possono impostare diversi separandoli con il punto-e-virgola. Il carattere '*' indica che la procedura può contenere zero, uno o più parametri. Nel caso di una procedura senza parametri, possiamo inserire una parentesi tonda aperta e chiusa "()".

Possiamo anche definire variabili che saranno utilizzate solo all'interno della procedura (illustreremo tale caratteristica nella parte riguardante il campo di applicazione delle variabili).

Ecco un esempio di procedura:

```
PROCEDURE: StampaSomma (numero1: intero; numero2: intero)  
  
VARIABILI  
  
INIZIO  
  
    WRITE numero1 + numero2;  
  
FINE;
```

Questa procedura riceve due numeri come input e ne stampa la somma a monitor.

Le procedure devono essere inserite dopo la dichiarazione di variabili e prima dell'inizio del programma principale (come mostrato nell'esempio riportato nella pagina successiva).

Chiamata di una procedura

Abbiamo visto come creare una procedura. Tuttavia, abbiamo bisogno di vedere come è possibile richiamarla dall'interno di un programma. La sintassi per realizzare ciò è:

```
<Nome della procedura> ([<valore>] *);
```

Dove valore è il valore di un parametro. Nel caso di più parametri, essi devono essere passati nello stesso ordine in cui sono stati definiti nella dichiarazione della procedura. Se non c'è alcun parametro il nome della procedura è seguito dalle parentesi tonde aperte e chiuse, come già detto prima.

Un valore può essere passato come valore costante, espressione o variabile, purché rispetti il tipo di parametro definito nella dichiarazione della procedura.

Quanto segue è un suggerimento di come implementare la nostra procedura StampaSomma vista precedentemente:

```
PROGRAMMA: ChiamataProcedura;

VARIABILI

    VAR valore: intero;

PROCEDURE: StampaSomma (numero1: intero; numero2: intero)

VARIABILI

INIZIO

    WRITE numero1 + numero2;

FINE;

INIZIO

    WRITE "Digitare un numero, per favore";
    READ valore;

    StampaSomma (5, valore);

FINE.
```

In questo esempio, il programma chiede all'utente di digitare un numero e poi ne stampa il valore incrementato di 5. Possiamo vedere nel codice della chiamata un parametro che riceve un valore costante e l'altro una variabile. Nonostante la semplicità dell'esempio, StampaSomma potrebbe essere utilizzato in altre parti del codice, anche se fosse un algoritmo più complesso. La procedura potrebbe essere utilizzata, per esempio per disegnare personaggi differenti in un gioco, stampare messaggi di errore, scrivere file e molte altre funzioni ripetitive che potremmo riutilizzare.

Ambito di visibilità delle variabili

Abbiamo parlato in precedenza circa la possibilità di definire variabili all'interno di procedure che saranno utilizzate solo durante la loro esecuzione. Questo è quello che noi chiamiamo l'ambito di visibilità di una variabile.

Definire l'ambito di visibilità (scope) di una variabile è molto importante perché permette di utilizzarle solo se necessario, evitando così che esse possano produrre effetti collaterali in altre parti del codice.

Vediamo un esempio:

```
PROGRAMMA: VisibilitàProcedura;

VARIABILI
    VAR valore: intero;

PROCEDURE: StampaValore ()
    VARIABILI
        VAR valore: intero;
    INIZIO
        valore ← 5;
        WRITE valore;
    FINE;

INIZIO
    valore ← 3;
    StampaValore ();
    WRITE valore;
FINE.
```

Questo programma stampa sullo schermo del computer prima il numero 5 (a causa della chiamata di StampaValore) e quindi di seguito il valore 3. La variabile valore dichiarata nell'ambito della procedura StampaValore è considerata come una variabile indipendente da quella dichiarata nel programma principale.

Funzioni

Una funzione ha un comportamento e una struttura simile a una procedura. Ma c'è una grande differenza: le funzioni restituiscono un valore, le procedure no!

LOGICA DI PROGRAMMAZIONE - MODULO 4

La dichiarazione di una funzione è fatta in un modo molto simile a quella di una procedura. Deve similmente essere situata tra la dichiarazione delle variabili del programma principale (variabili globali) e l'inizio di questo.

```
FUNCTION: <nome della procedura> ([<nome del parametro>: <tipo>] *): <tipo>

VARIABILI

    {dichiarazione delle variabili}

INIZIO

    (operazioni);

    RETURN <valore>;

FINE;
```

Il <tipo>, alla fine della dichiarazione, è il tipo di valore che sarà restituito dalla funzione attraverso l'esecuzione del comando RETURN, posizionato nell'ultima riga della funzione. È importante ricordare che il RETURN deve essere l'ultima operazione che deve essere eseguita nella funzione: ulteriori operazioni presenti dopo il RETURN non saranno eseguite.

Potremmo convertire la procedura StampaSomma facendola diventare una funzione che restituisce il valore della somma:

```
FUNCTION: Somma (numero1: intero; numero2: intero): intero

VARIABILI

INIZIO

    RETURN numero1 + numero2;

FINE;
```

La chiamata di una funzione è identica a quella di una procedura. Tuttavia è utile ricordare che essa restituisce un valore, che può essere utilizzato per assegnare un contenuto in una variabile (a condizione che i tipi siano uguali o compatibili) o come parte di un'espressione, come nell'esempio seguente esempio:

```
PROGRAMMA: ChiamataFunzione;
```

```
VARIABILI
```

```
VAR valore: intero;
```

```
VAR ritorna: intero
```

```
FUNCTION: Somma (numero1: intero; numero2: intero): intero
```

```
VARIABILI
```

```
INIZIO
```

```
    RETURN numero1 + numero2;
```

```
FINE;
```

```
INIZIO
```

```
    WRITE "Digitare un numero, per favore";
```

```
    READ valore;
```

```
    ritorna ← Somma (5, valore);
```

```
    WRITE ritorna;
```

```
FINE.
```

Ricorsione

La ricorsione è la possibilità che possiamo richiamare la stessa procedura, o funzione, dall'interno della sua esecuzione, generando un processo ripetitivo. Un esempio della ricorsione nel mondo reale è quando due specchi sono posti faccia a faccia e l'uno riflette l'immagine dell'altro, in successione.

"Nel tentativo di risolvere il problema,

ho incontrato ostacoli all'interno di ostacoli.

Quindi, ho adottato una soluzione ricorsiva. "

- Frase di uno studente della USP sulla ricorsione

Questa tecnica è molto importante nei linguaggi di programmazione in quanto può fornire agli sviluppatori soluzioni che ottimizzano i loro codici, in funzione dei problemi da risolvere. Inoltre, le chiamate ricorsive della stessa procedura o funzione, sono praticamente indipendenti per l'ambito di visibilità delle variabili, discusso precedentemente.

Un classico esempio di algoritmo che può essere implementato utilizzando la ricorsione è il calcolo del fattoriale di un dato numero.

Per ricordare: Il fattoriale di un numero "n": si calcola moltiplicando tutti i numeri naturali dal numero "n" fino a raggiungere il numero 1 ed è rappresentata dalla notazione "n!" ("n fattoriale"). Ad esempio, il

LOGICA DI PROGRAMMAZIONE - MODULO 4

risultato di 4! (4 fattoriale) è 24, poiché viene eseguita la seguente moltiplicazione: $4 * 3 * 2 * 1 = 24$. Un modo equivalente di scrivere 4! è $4 * 3!$, che restituisce 24. Se espandiamo questo sviluppo agli altri numeri utilizzati nel prodotto, avremmo:

$$4! = 4 * 3!$$

$$4! = 4 * 3 * 2!$$

$$4! = 4 * 3 * 2 * 1$$

Quindi una possibile soluzione per il calcolo del fattoriale è l'uso della ricorsione come presentato di seguito:

```
PROGRAMMA: CalcoloFattoriale;
```

```
VARIABILI
```

```
    VAR num, risultato: intero;
```

```
FUNCTION: Fattoriale (numero: intero): intero
```

```
VARIABILI
```

```
    VAR risultato: intero;
```

```
INIZIO
```

```
    IF (numero > 1)
```

```
    THEN
```

```
        risultato ← numero * Fattoriale (numero -1);
```

```
    ELSE
```

```
        risultato ← 1;
```

```
    ENDIF;
```

```
    RETURN risultato;
```

```
FINE;
```

```
INIZIO
```

```
    WRITE "Digitare il numero desiderato per il calcolo del fattoriale ";
```

```
    READ num;
```

```
    risultato ← Fattoriale (num);
```

```
    WRITE "Il fattoriale è: ", risultato;
```

```
FINE.
```

Cosa importante da notare in questo esempio è che trattiamo il caso più semplice, quando il numero è uguale a 1, in maniera diversa dagli altri casi. Se questo non avviene, l'algoritmo innesca ricorsivamente un processo senza fine. Ogni volta che usiamo la ricorsione dobbiamo pensare ad una condizione di arresto, così come quando trattiamo comandi ripetitivi.

Conclusione

In questa sezione abbiamo conosciuto le procedure e le funzioni. Esse servono a rendere il codice modulare e a facilitare il riutilizzo degli algoritmi. Abbiamo visto la struttura di base di una procedura e di una funzione e le modalità di chiamata, abbiamo inoltre visto l'ambito di visibilità delle variabili. Infine, abbiamo appreso un concetto molto importante, quello della ricorsione. Se si hanno dubbi, rivedere il contenuto!