Sottoprogrammi

Cap 9 - Procedure e Funzioni

Prof. Ivan Gentile

ITIS Ferraris - 2024

Introduzione

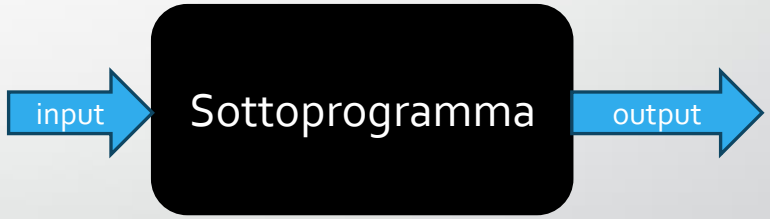
- Spesso alcune istruzioni vengono ripetute in diverse parti all'interno del medesimo programma, al limite cambiano solo i nomi di alcune variabili
  - Esempio Lettura/Stampa array
- E' utile definire **una sola volta** questo gruppo di istruzioni e darne un **nome**

Sottoprogramma

- E' un insieme di istruzioni che svolgono uno specifico e ben definito compito, a cui diamo un nome
- Definito una sola volta
- Attivabile più volte all'interno del programma o di un altro sottoprogramma
- Nomi (ormai) equivalenti: funzione, procedura, routine, subroutine, metodi

Esempi

- Alcuni sottoprogrammi li usiamo abitualmente
  - scanf, printf, cin, cout
- Spesso ci interessa solo **cosa** fanno e non come (black box)



Utilità

- Astrazione e leggibilità
- Compattezza ed efficienza del codice
- Modificabilità e riuso

Funzioni e procedure

- A voler essere più precisi Sottoprogrammi si dividono in
  - Procedure (subroutine, routine)
  - Funzioni (metodi)
- **Procedura**: generico compito da svolgere (0 o più output e input)
  - `leggi(N); /* Procedura */`
- **Funzione**: calcolo di una specifica grandezza (un solo output)
  - `fatt = fattoriale(N);  /* Funzione */`

Parametri (1/2)

- Ai sottoprogrammi possono essere passati dei parametri (dati)
  - **Parametri di Input**: utili al sottoprogramma
  - **Parametri di Output**: utili al programma chiamante
  - **Parametri di Input e Output**: utili sia al sottoprogramma che al programma chiamante
- In linea di principio le funzioni hanno uno o più parametri di input e un solo parametro di output
- Le procedure **zero o più** parametri di input, output, input e output

Struttura di un sottoprogramma in C/C++
Definizione

```
tipoRestituito nomeSottoprg(lista dei parametri) {
    // Istruzioni di svolgimento
}
```

1 input, 1 output

```
long long int fattoriale(int n) {
  long long int f;
  // calcolo del fattoriale f
  return f;
}
```

2 input, 1 output

```
double imposta(double reddito, int aliquotaPerc) {
  double imposta;
  // calcolo dell'imposta
  return imposta;
}
```

Sottoprogramma con parametri in Input/Output

- In C/C++ la lista dei parametri può contenere anche parametri di **output** oppure parametri di input/output
- I parametri di output o di input/output vanno specificati con una &
- E' indispensabile sfruttare questa possibilità quando il sottoprogramma deve restituire più di 1 output
  - Esempio: devo avere 2 output: uno lo gestisco con il return e l'altro con &, oppure entrambi con &
- Se il sottoprogramma non restituisce nulla (nessun return) bisogna mettere void nel tipo restituito

2 input/output (a,b)

```
void scambioVariabili(int &a, int &b) {
  // scambia il valore di a con il valore di b
}
```

2 input (v, n) e 2 output (maxArrayAndPos, pos)

```
int maxArrayAndPos(int v[], int n, int &pos) {
  // ritorna il valore massimo di v e la sua posizione
}
```

Parametri (2/2)

- `Parametro` o argomento **attuale**: nome date dal programma chiamante
- Parametro o argomento **formale**: nome dato nella definizione della funzione
- Il nome del parametro attuale può differire da quello formale ma il tipo deve essere lo stesso.

Parametri (3/3)

- I parametri e le variabili dichiarate all'interno di una funzione sono locali (dette anche automatiche) cioè sono **invisibili** all'esterno di una funzione,
- nascono quando la funzione viene chiamata e muoiono quando la funzione termina

Sottoprogrammi in C/C++

- Esistono **solo** le funzioni
- Ma c'è una flessibilità tale da realizzare anche procedure (o misti)

Esempio: Combinazioni di n elementi su k posti (in C++)

```cpp
#include <iostream>
using namespace std;

unsigned long long fattoriale(int);

int main(void){
  int n, k;
  cout << "Inserisci il numero di elementi dell'insieme n = ";
  cin>>n;

  cout << "Inserisci il numero di posti k = ";
  cin>>k;

  cout<< "C(n,k) = " << fattoriale(n)/(fattoriale(k)*fattoriale(n-k));

  }

unsigned long long fattoriale(int x){
  int i;
  unsigned long long f = 1;
  for (i = 1; i <= x; i++)
   f= f * i;
  return f;
}
```

prototipo (o dichiarazione)

chiamate (invocazioni)

Definizione

Esempio: calcolo area del cerchio (in C)

```c
/* Calcolo dell'area del cerchio dato il raggio */
/* Autore: Ivan Gentile */

#include <stdio.h>

/* Prototipo: Calcolo dell'area di un cerchio dato un raggio */
float areaCerchio(float);

int main () {
    float area, raggio;

    printf("Calcolo dell'area di un cerchio dato il raggio\n");
    printf("Inserisci il raggio:");
    scanf("%f",&raggio);

    area = areaCerchio(raggio);
    printf("L'area del cerchio e':%.2f",area);

    fflush(stdin);
    getchar();
    return 0;
}

/* Definizione: Calcolo dell'area di un cerchio dato il raggio */
float areaCerchio(float r) {
    const float pi = 3.14;
    float a;

    a = pi * r * r;

    return a;

}
```

Prototipo: interfaccia, messo prima dell'uso (indispensabile se la def. è fatta dopo il main)

Chiamata

Valore restituito

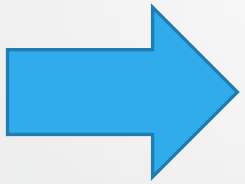
Definizione (può andare in un punto qualunque, ma fuori dal main)

Passaggio dei parametri: per valore/copia

Parametro attuale

n

10



Parametro formale

n1

10

Esempio in C: chiamata

```
fattoriale(n);
```

Viene copiato il valore: n non è modificato

Esempio in C: definizione

```
int fattoriale(int n1) { ... }
```

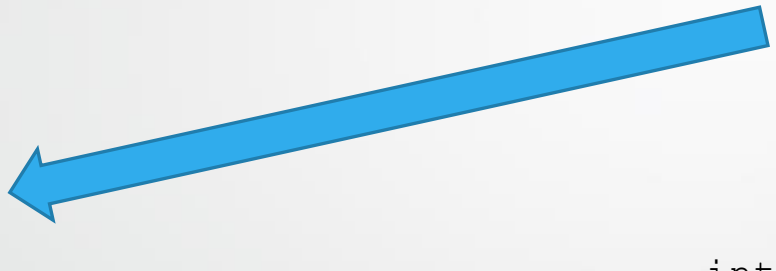
Passaggio dei parametri: per indirizzo (C)

Parametro attuale

a

Parametro formale

a1



Esempio in C: chiamata

```
swap(&a, &b);
```

Viene copiato l'indirizzo

Esempio in C: definizione

```
int swap(int* a1, int* b1) {
*a1 =...
...
 }
```

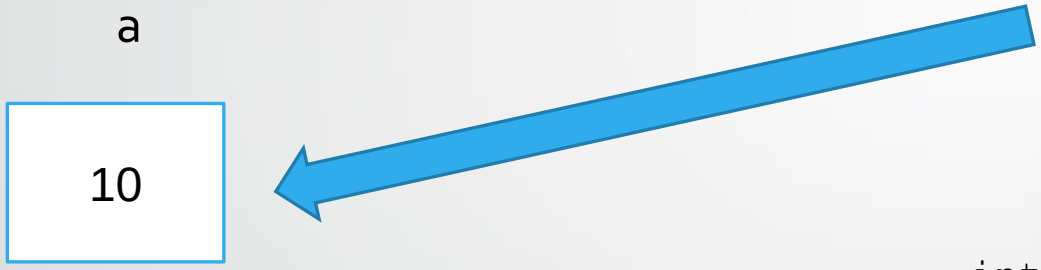
Passaggio dei parametri: per indirizzo (C++)

Parametro attuale

a

Parametro formale

a1



Esempio in C: chiamata

```
swap(a, b);
```

Viene copiato l'indirizzo (alias)

Esempio in C++: definizione

```
int swap(int &a1, int &b1) {
 a1 =...
 ...
}
```

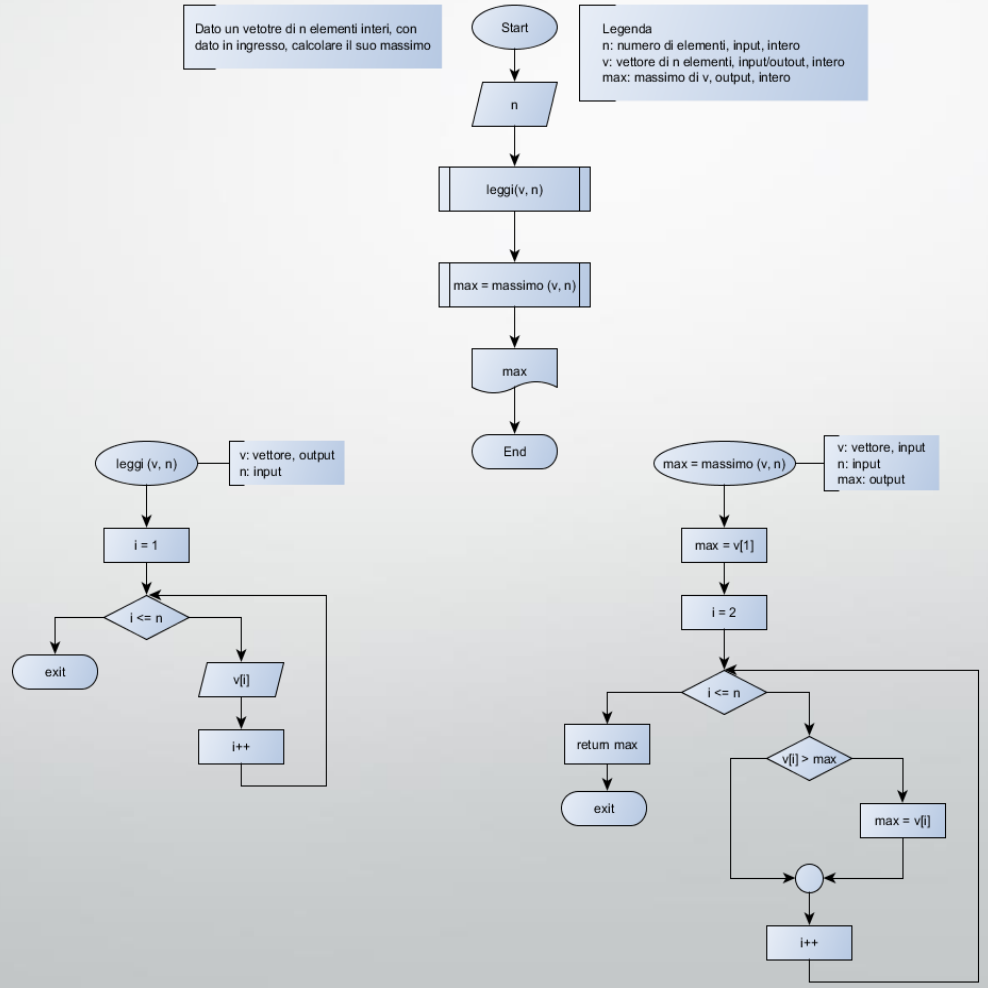
Passaggio di un vettore

- In C esiste solo il passaggio per valore
- Per quello di indirizzo si usano i puntatori
- Per i vettori il passaggio per valore è in realtà un passaggio per riferimento
  - Chiamata: `ordinaVettore (v);`
  - Dichiarazione/definizione: `void ordinaVettore (int v[ ]);`
  - Altro metodo per i vettori: usare direttamente i puntatori

Attributo const

- Per evitare duplicazioni di variabili si può usare il passaggio per riferimento anche per i parametri di ingresso
- In generale sarebbe opportuno per i parametri di ingresso (sia essi passati per riferimento o per valore) usare l'attributo const

Flow-Chart


Dato un vetotre di n elementi interi, con dato in ingresso, calcolare il suo massimo
Start
Legenda
n: numero di elementi, input, intero
v: vettore di n elementi, input/outout, intero
max: massimo di v, output, intero
n
leggi(v, n)
max = massimo (v, n)
max
End
leggi (v, n)
v: vettore, output
n: input
i = 1
i <= n
exit
v[i]
i++
max = massimo (v, n)
v: vettore, input
n: input
max: output
max = v[1]
i = 2
i <= n
return max
exit
v[i] > max
max = v[i]
i++

Approfondimenti

- Studiare i valori di default per i parametri di una funzione
- Studiare il meccanismo dell'overloading dei nomi di funzione

Esercizi

- Riscrivere tutti i programmi delle categorie `Sequenza`, `Selezione`, `Operatori Logici`, `Iterazione 1`, `Caratteri`, `Iterazione 2`, `Vettori`, in termini di sottoprogrammi chiamando le cartelle precedenti col nome a le nuove con b.