Funzioni

Maurizio Rebaudengo

Politecnico di Torino
Dip. di Automatica e Informatica

1

Funzioni

I problemi esaminati sino ad ora sono stati risolti con programmi C costituiti da un solo modulo.

2

Svantaggi di un singolo modulo

Se il programma assume, pero' dimensioni ragguardevoli, questo approccio presenta una serie di inconvenienti.

L'eccessiva estensione di un modulo provoca problemi di comprensibilita', manutenzione e correzione del programma stesso.

Inoltre se il problema e' complesso e' probabile che alcuni blocchi di operazioni siano ripetute piu' volte; e' quindi conveniente che le relative istruzioni compaiano in un unico punto, e vengano attivate piu' volte in fasi diverse del programma.

3

Programma = insieme di funzioni

Si tende a scrivere programmi composti da una serie di moduli che nel loro complesso si distinguono in:

- ***Programma principale***
- **Una serie di *sottoprogrammi*, detti anche *funzioni*.**

4

Attivazione di sottoprogrammi

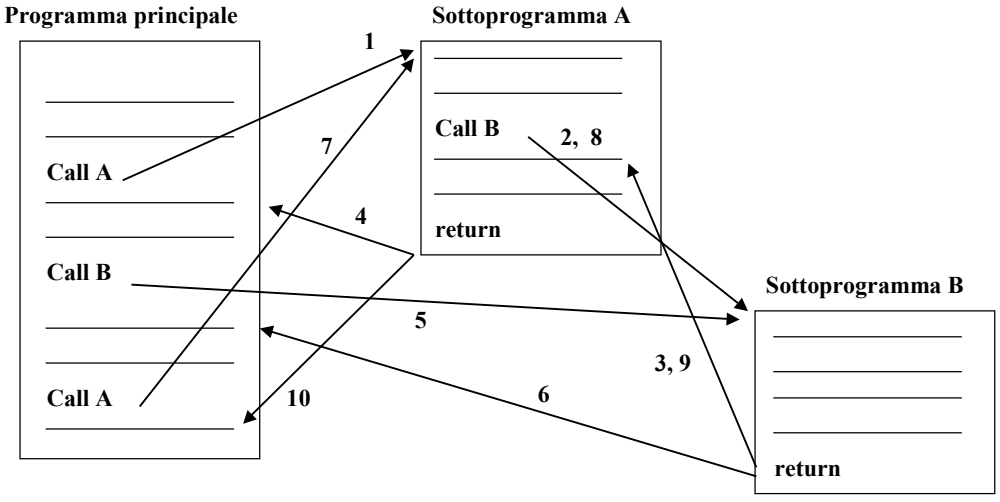
L'unicita' del flusso di controllo, tipica dei linguaggi di programmazione quali il C, richiede che uno solo dei moduli sia attivo in ogni istante dell'esecuzione di un programma.

Il meccanismo di esecuzione di un sottoprogramma si basa sui seguenti passi:

- **attivazione della funzione attraverso una istruzione di *chiamata*, ricevendo eventuali dati in ingresso**
- ***esecuzione* di un insieme di istruzioni interne che agiscono su dati locali e su quelli ricevuti in ingresso**
- ***ritorno* del controllo al sottoprogramma chiamante mediante un'istruzione di ritorno, restituendo, come dati di uscita, i risultati delle elaborazioni effettuate.**

5

Esempio



6

Uno stesso modulo puo' essere attivato:

- **piu' volte in punti diversi dello stesso modulo**
- **da moduli diversi.**

7

Scambio di dati

La necessita' di far interagire piu' moduli richiede, oltre alle regole per il controllo del flusso, un meccanismo di scambio di dati tra sottoprogrammi. Cio' si realizza mediante:

- **un esplicito trasferimento di dati in corrispondenza a chiamata a funzione e ritorno da funzione**
- **l'eventuale utilizzo di variabili globali, cioe' esterne (non locali) ai moduli interessati.**

8

Allocazione e visibilita'

L'*allocazione* di una variabile e' il processo di generazione della variabile mediante l'attribuzione di una porzione di memoria, a cui viene associato l'indentificatore della variabile stessa.

La visibilita' (*scope*) di una variabile e' la porzione di programma all'interno della quale la variabile e' conosciuta ed accessibile.

9

Variabili globali

Le variabili globali (o esterne) sono accessibili a tutte le funzioni del programma.

10

Allocazione e visibilita' delle variabili globali

L'allocazione delle variabili globali e' permanente.

Le variabili globali sono visibili in tutte le funzioni dichiarate nello stesso file sorgente.

11

Parametri

L'invio dei dati da parte di un programma chiamante ad un sottoprogramma chiamato si basa sui cosiddetti *parametri*.

I parametri sono detti *attuali* dalla parte del programma chiamante e sono detti *formali* dalla parte del sottoprogramma chiamato.

I parametri formali sono visti dal sottoprogramma come vere e proprie variabili locali (con nome unico all'interno del sottoprogramma stesso).

Il programma chiamante puo' utilizzare variabili diverse per chiamate diverse dello stesso sottoprogramma.

12

Passaggio dei parametri

Il tipo di comunicazione puo' essere effettuato nei modi seguenti:

- ***By value* (*per valore*), se il parametro attuale e' un valore (ossia il contenuto di una variabile o piu' in generale il risultato di una espressione) che viene assegnato al parametro formale in corrispondenza della chiamata. Il parametro formale e' a tutti gli effetti una variabile locale (opportunamente inizializzata) del sottoprogramma; ogni modifica eseguita dal sottoprogramma sui parametri non ha effetto alcuno sui corrispondenti parametri attuali del programma chiamante.**

13

Passaggio dei parametri (II)

- ***By reference* (*per riferimento*), se il parametro attuale indica l'indirizzo di una variabile. In questo caso il parametro formale e' una variabile locale che permette di accedere ad una variabile del programma chiamante. Ogni eventuale modifica di tale variabile effettuata all'interno del sottoprogramma si riflette direttamente su di essa.**

14

Struttura di un programma C

Un programma C risultera' come un insieme di funzioni (o sottoprogrammi) e variabili globali conformi alle regole seguenti:

- **Una delle funzioni deve essere denominata *main* e costituisce il cosiddetto programma principale, attivato all'avvio del programma**
- **Tutte le funzioni (compreso il *main*) e le variabili globali sono poste allo stesso livello, cioe' non esiste alcuna gerarchia tra di esse e ogni funzione puo' chiamare tutte le altre (tranne il main) come pure accedere a tutte le variabili globali.**

15

Definizione di una funzione

Il formato per la definizione di una funzione in linguaggio C e' il seguente:

<tipo funzione> <nome funzione> (<parametri formali>)

{

< definizioni locali>

<corpo della funzione>

}

16

Valore di ritorno

<tipo funzione> rappresenta il tipo del valore ritornato dalla funzione. Esistono i seguenti casi:

- **La funzione non ritorna nessun valore; in tal caso si utilizza il tipo *void***
- **La funzione ritorna un valore; in tal caso si utilizza un tipo scalare (*char, int, float*), un puntatore, oppure un tipo strutturato**

17

Definizioni locali

Insieme delle istruzioni dichiarative di definizione delle variabili locali utilizzate dalla funzione. Tali variabili sono utilizzabili solo dalla funzione stessa e non sono visibili da altre funzioni o dal programma principale.

Le variabili locali sono allocate soltanto durante l'esecuzione della funzione stessa (cioe' nascono quando questa viene attivata e muiono quando il flusso ritorna al chiamante) e sono visibili soltanto all'interno di essa.

18

Allocazione e visibilita' delle variabili locali

L'allocazione delle variabili locali e' temporanea alla sola durata della funzione.

Le variabili locali sono visibili solo all'interno della funzione stessa.

19

Parametri formali

Corrisponde alla lista dei parametri, cioe' dei valori di ingresso alla funzione. Essi possono mancare, ed in tal caso la lista dei parametri e' sostituita dalla parola chiave *void*.

20

Corpo della funzione

Contiene le istruzioni operative che caratterizzano le azioni svolte dalla funzione.

Tra di esse va ricordata l'istruzione *return* che assolve ai seguenti compiti:

- **Fa ritornare il controllo dell'esecuzione al programma chiamante, all'istruzione successiva a quella di chiamata alla funzione stessa**
- **Restituisce al programma chiamante un valore di tipo specificato in <tipo funzione>, come specificato nella definizione della funzione stessa.**

21

Istruzione di ritorno

Il formato dell'istruzione di ritorno e' il seguente:

return <espressione>;

In cui <espressione> rappresenta l'espressione che produce il valore da ritornare al programma chiamante.

Spesso tale espressione viene inclusa in parentesi tonde per aumentarne la chiarezza.

22

Esempio

```
void messaggio (void)
{
printf'("Messaggio di errore\n");
return;
}
```

23

Esempio (II)

```
int maggiore (int a, int b)
{
int max;

if (a>b)
        max = a;
else max = b;

return(max);
}
```

24

Variabili globali

La dichiarazione delle variabili globali viene fatta nella parte iniziale del codice sorgente prima della dichiarazione delle funzioni.

Lo scope di una variabile globale corrisponde a tutte le funzioni dichiarate successivamente ad essa nello stesso modulo sorgente.

25

Chiamata di funzione

Il programma chiamante attiva l'esecuzione della funzione mediante un'istruzione del tipo:

<nome funzione> (<parametri attuali>);

26

Passaggio di parametri

Esiste una corrispondenza posizionale tra i parametri *formali* ed i parametri *attuali*.

Il numero dei parametri attuali deve coincidere con il numero dei parametri formali ed il tipo dei due parametri corrispondenti deve coincidere.

All'atto della chiamata a ciascun parametro formale viene assegnato il valore del corrispondente parametro attuale.

27

Esecuzione di una chiamata

L'esecuzione dell'istruzione di chiamata ha i seguenti effetti:

- **Viene attivata la funzione <nome funzione>, gestendo opportunamente il passaggio dei relativi parametri: ciascun parametro formale assume cioe' il valore del corrispondente parametro attuale**
- **L'esecuzione del programma chiamante viene sospesa ed il controllo del flusso di esecuzione passa alla prima istruzione della funzione chiamata**
- **Il controllo del flusso di esecuzione resta alla funzione chiamata sino all'esecuzione di un'istruzione *return***
- **Il controllo ritorna al chiamante ed il valore ritornato e' a disposizione del chiamante stesso.**

28

Lettura del valore di ritorno

Se la funzione ritorna un valore esso puo' essere utilizzato direttamente oppure puo' essere assegnato ad una variabile.

Il formato dell'istruzione di chiamata diventa il seguente:

<var> = <nome funzione> (<parametri attuali>) ;

Il tipo della variabile <var> deve essere lo stesso del valore ritornato dalla funzione.

29

Esempio

```
void main()
{
int a, b, max;
a=5;
b=6;

max = maggiore(a,b);
printf("il maggiore e' %d\n", max);
}
```

30

Esempio (II)

```
void main()
{
int a, b;
a=5;
b=6;

printf("il maggiore e' %d\n", maggiore(a,b));

}
```

31

Prototipi

Una funzione puo' essere chiamata solo nella parte del file sorgente successiva alla sua definizione. Per ovviare a questa limitazione, in C e' possibile dichiarare una funzione mediante il cosiddetto *prototipo*: questo e' un duplicato dell'intestazione di una funzione (seguito da un ;) contenente le informazioni sufficienti per eseguirne correttamente la chiamata.

La struttura del prototipo e' la seguente:

<tipo funzione> <nome funzione> (<parametri formali>);

Ad esempio:

```
int maggiore (int a, int b);
```

32