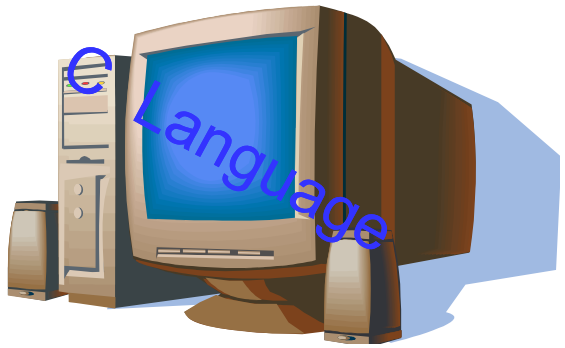
Funzioni e passaggio parametri

Alessandra Giordani
agiordani@disi.unitn.it
Mercoledì 16 maggio 2012
http://disi.unitn.it/~agiordani/

Cosa vedremo oggi

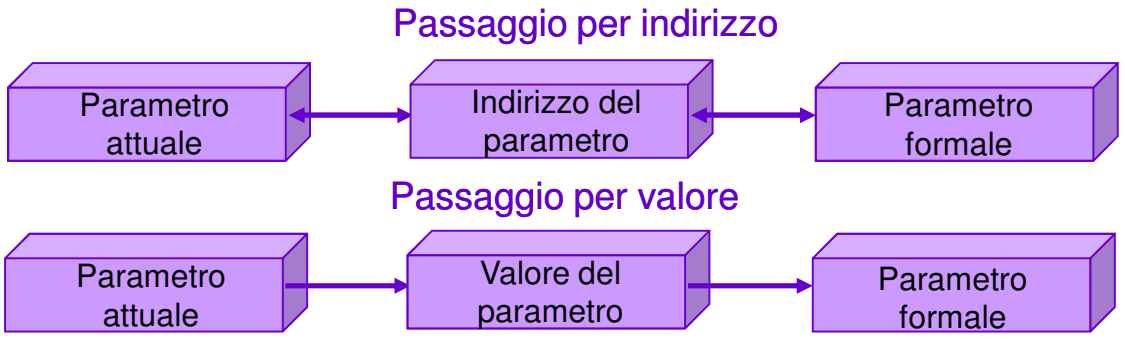
- Le funzioni
  - Il passaggio dei parametri
  - Le dichiarazioni e le chiamate
  - I prototipi di funzione
  - La ricorsione
- La funzione *main()*



Il passaggio dei parametri – 1

- Gli argomenti di una funzione costituiscono il mezzo per passare dati al sottoprogramma
- In C, gli argomenti sono passati per valore (*by value*), ovvero viene passata una copia dell'argomento

  $\Rightarrow$ la funzione chiamata può modificare il valore della copia, ma non dell'argomento originale



Nella chiamata per indirizzo, i parametri formale ed attuale fanno riferimento alla stessa area di memoria; nel caso di chiamata per valore, il parametro formale è una copia del parametro attuale

- L'argomento passato viene detto parametro attuale, mentre la copia ricevuta è il parametro formale

Il passaggio dei parametri – 2

- Dato che in C gli argomenti vengono passati per valore, la funzione chiamata può modificare il parametro formale, senza che ciò si ripercuota sul parametro attuale

La funzione *printf()* non stampa 3, ma 2, in quanto il parametro formale b in f() è una copia del parametro attuale a

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        extern void f();
        int a=2;
        f(a); /* passa una copia di a ad f() */
        printf("%d\n", a);
        exit(0);
}
void f(int b)
{
        b = 3;
}
```

Il passaggio dei parametri – 3

- In C, il parametro attuale, usato nella chiamata, ed il corrispondente parametro formale, usato nella definizione della funzione, vengono associati, indipendentemente dai nomi adottati
  - Il primo parametro attuale viene a corrispondere al primo parametro formale, il secondo parametro attuale al secondo parametro formale, etc.
- È necessario che i tipi dei parametri attuali coincidano con quelli dei corrispondenti parametri formali
- Inoltre, se occorre che una funzione modifichi il valore di un oggetto, è necessario passare un puntatore all'oggetto ed assegnare all'oggetto un valore mediante l'operatore di accesso all'indirizzo contenuto nel puntatore

Il passaggio dei parametri – 4

Esempio

```
void scambia(int *x, int *y)
{
        int temp;
        temp = *y;
        *y = *x;
        *x = temp;
}
main()
{
        int a=2, b=3;
        scambia(&a, &b)
        printf("a=%d\t b=%d\n", a, b);
}
```

L'esecuzione del programma produce

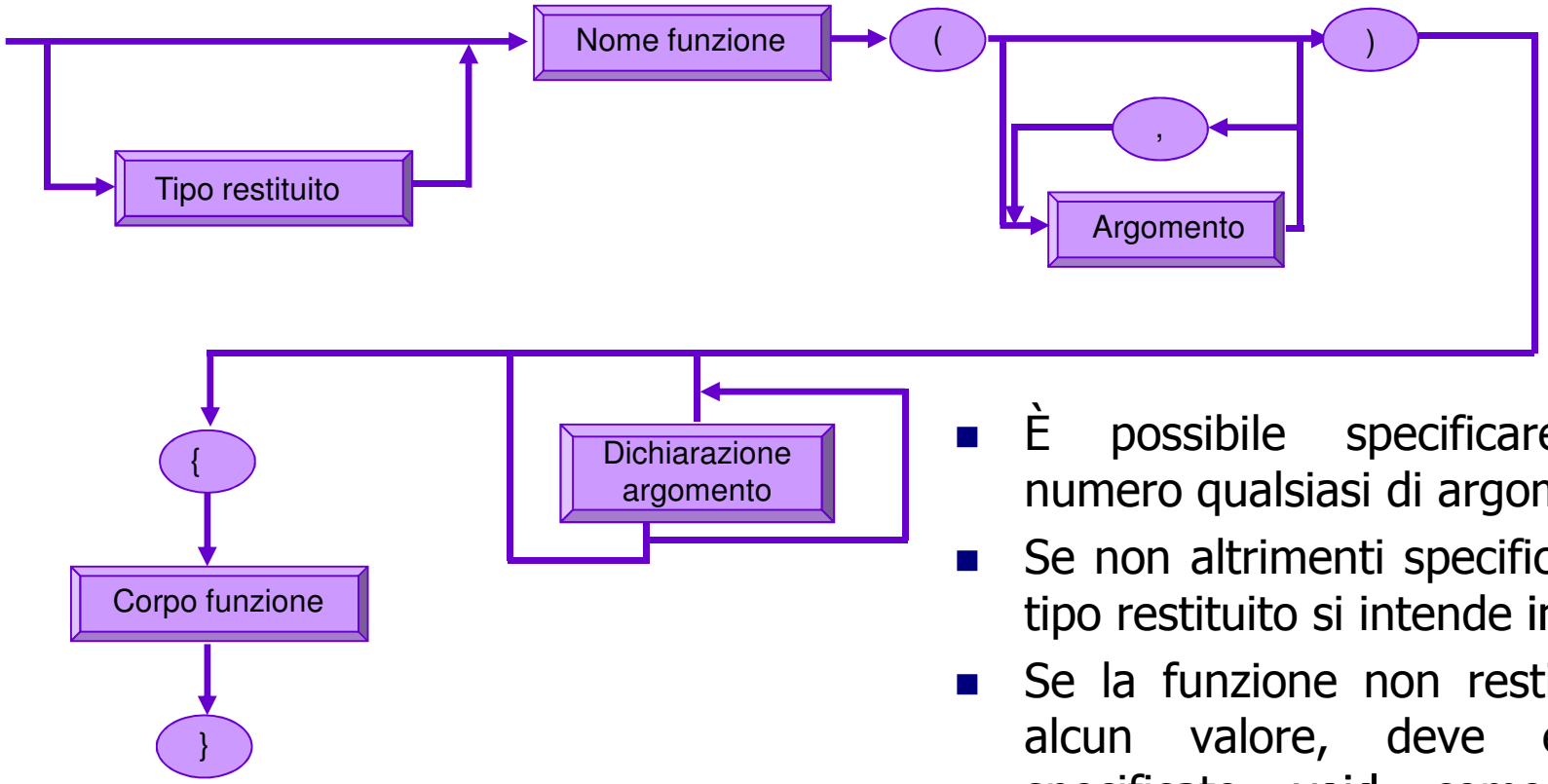
a=3 b=2

Nota: Il passaggio degli argomenti per valore chiarisce la necessità di usare l'operatore "indirizzo di", &, nelle chiamate alla funzione *scanf()* : se venissero passate direttamente le variabili, *scanf()* non potrebbe modificarle; passandone gli indirizzi, *scanf()* può accedere alle variabili ed assegnare i valori letti

Le dichiarazioni e le chiamate

- Le funzioni possono apparire in un programma in una delle tre forme seguenti:
  - Definizione — dichiarazione, che definisce il numero ed il tipo degli argomenti ed il comportamento della funzione
  - Allusione a funzione — dichiarazione di una funzione definita altrove, che specifica la natura del valore restituito dalla funzione (con la prototipazione possono essere specificati anche il numero ed il tipo degli argomenti)
  - Chiamata di funzione — invocazione di una funzione, che ha come effetto il trasferimento del controllo del programma alla funzione chiamata; al termine della funzione chiamata, l'esecuzione riprende dall'istruzione (del programma chiamante) immediatamente successiva alla chiamata completata

La sintassi della definizione di funzione



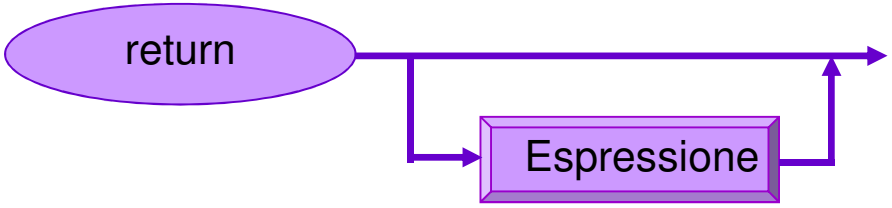
- È possibile specificare un numero qualsiasi di argomenti
- Se non altrimenti specificato, il tipo restituito si intende int
- Se la funzione non restituisce alcun valore, deve essere specificato void come tipo restituito

La dichiarazione degli argomenti

- La dichiarazione degli argomenti segue le stesse regole delle dichiarazioni di variabili, con le seguenti eccezioni:
  - Oggetti di tipo char e short sono convertiti in int, mentre oggetti di tipo float sono convertiti in double (con la prototipazione, le conversioni automatiche possono essere evitate)
  - Un parametro formale dichiarato come array viene convertito in un puntatore ad un oggetto del tipo base dell'array
  - In una dichiarazione, non può essere presente una inizializzazione
- La dichiarazione di un argomento può essere omessa: il tipo dell'argomento è int per default

I valori restituiti – 1

- Le funzioni possono restituire direttamente un solo valore, mediante l'istruzione return
- Il valore restituito può essere di tipo qualunque, eccetto array o funzione
- È possibile restituire indirettamente più di un valore mediante la restituzione di un puntatore ad un tipo composto
- Inoltre, una struttura o un'unione possono essere restituite direttamente (anche se ciò è sconsigliato per motivi di efficienza)
- La sintassi di un'istruzione return è



I valori restituiti – 2

- In una funzione possono essere presenti un numero qualsiasi di istruzioni return: la prima incontrata nell'evolvere del flusso restituisce il controllo al programma chiamante
- Il valore restituito deve essere compatibile con il tipo della funzione
- Se non esiste alcuna istruzione return, il controllo del programma ritorna alla funzione chiamante quando si raggiunge la parentesi graffa chiusa che conclude il corpo della funzione: il valore restituito è indefinito
- Un'istruzione return senza espressione può essere utilizzata (senza effetti collaterali) per restituire il controllo al chiamante dall'interno di una funzione dichiarata void

I valori restituiti – 3

- Esempio: valori restituiti da return, implicitamente convertiti dal compilatore, per una funzione che dovrebbe restituire un float

```
float f()
{
        float f2;
        int a;
        char c;

        f2 = a;      /* OK, conversione implicita di a in float */
        return a;    /* OK, conversione implicita di a in float */
        f2 = c;      /* OK, conversione implicita di c in float */
        return c;    /* OK, conversione implicita di c in float */
}
```

Le allusioni a funzione

- Un'allusione a funzione è una dichiarazione di una funzione definita altrove: serve principalmente per informare il compilatore sul tipo restituito dalla funzione
- Poiché, se non specificato, il tipo di una funzione è int per default, le allusioni a funzioni intere potrebbero essere omesse
- È comunque buona norma inserire le allusioni a tutte le funzioni

  $\Rightarrow$ possibilità di determinare tutte le funzioni richiamate dalla lettura delle frasi dichiarative, senza dover scandire tutto il sorgente

I prototipi di funzione – 1

- I prototipi di funzione (introdotti nel C++) consentono di includere nelle allusioni la specifica del tipo degli argomenti
  - ⇒ Il compilatore può controllare che il tipo dei parametri attuali nelle chiamate di funzione sia compatibile con il tipo dei parametri formali specificati nell'allusione
  - ⇒ Le conversioni automatiche non sono più necessarie (e non vengono effettuate) migliorando le prestazioni dei programmi che utilizzano pesantemente interi corti e floating–point in singola precisione
- Esempi:

```
extern void func(int, float, char *);

extern void func(int a, float b, char *pc);
```

I due prototipi sono uguali: i nomi degli argomenti aumentano la leggibilità, ma non viene loro riservata memoria, né si creano conflitti con variabili omonime

I prototipi di funzione – 2

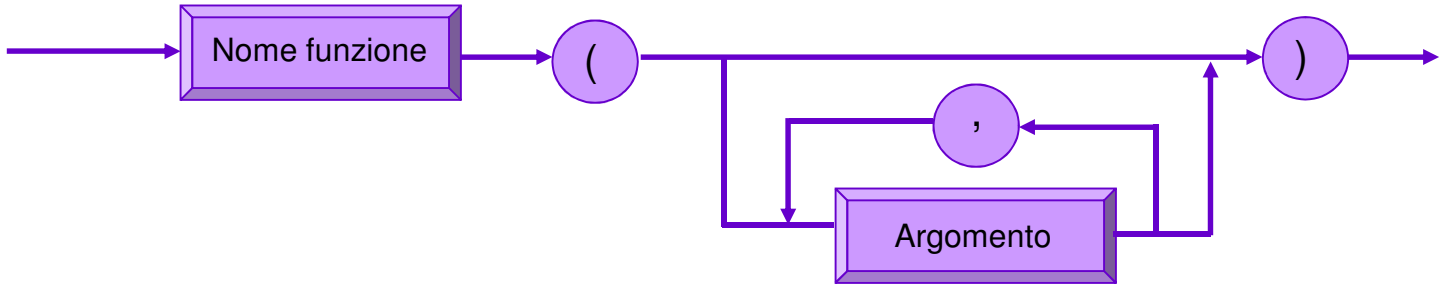
- La prototipazione assicura che venga passato il numero esatto di argomenti e impedisce il passaggio di argomenti che non possono essere convertiti implicitamente nel tipo corretto
- In assenza di argomenti, occorre specificare il tipo void
- Se una funzione gestisce un numero di argomenti variabile, può essere utilizzata la forma "..."
- Esempio: il prototipo di *printf()* è:

  int printf(const char *format, ...);

  che asserisce che il primo elemento è una stringa di caratteri costante, seguita da un numero non specificato di argomenti

Le chiamate di funzione – 1

- Una chiamata di funzione, detta anche invocazione di funzione, trasferisce il controllo del programma alla funzione specificata

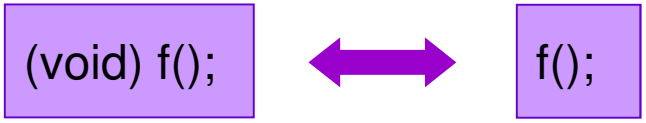


- Una chiamata di funzione è un'espressione e può quindi comparire ovunque sia ammessa un'espressione
- Salvo il caso in cui il tipo restituito è void, il valore restituito dalla funzione viene sostituito alla chiamata
- Esempio: se f() restituisce 1…

```
a = f()/3;   ⟺   a = 1/3;
```

Le chiamate di funzione – 2

- I valori di ritorno vengono ignorati solo quando la funzione restituisce void
- Se si desidera ignorare deliberatamente un valore restituito, è buona norma convertirlo esplicitamente in void



- Questa regola è stata trasgredita ogni volta che sono state usate le funzioni *printf()* e *scanf()*, che restituiscono valori interi (in particolare l'intero restituito da *printf()*/*scanf()* è il numero di oggetti scritti/letti)

Le conversioni automatiche degli argomenti – 1

- In mancanza di prototipazione, tutti gli argomenti scalari di dimensioni minori di un int vengono convertiti in int e gli argomenti float vengono convertiti in double
- Se il parametro formale è dichiarato essere un char o uno short o un float, anche la funzione chiamata, che si aspetta di ricevere int o double, rispettivamente, riconverte ai tipi più corti

$\Rightarrow$ Ogni volta che viene passato come parametro un char, uno short o un float vengono effettuate due conversioni, nel programma chiamante e nella funzione chiamata

Le conversioni automatiche degli argomenti – 2

- Esempio

```
{
    char a;
    short b;
    float c;
    foo(a,b,c);  /* a e b vengono convertiti in interi
                    * c viene convertito in double
                    */
    … … …
}

foo(x,y,z)
    char x;     /* L'argomento ricevuto è convertito
                   da int a char */
    short y;    /* L'argomento ricevuto è convertito
                   da int a short */
    float z;    /* L'argomento ricevuto è convertito
                   da double a float */
{
   … … …
}
```

- Nell'ipotesi di coincidenza fra i tipi dei parametri formali ed attuali, gli argomenti vengono passati correttamente
- Tuttavia… le conversioni automatiche possono far diminuire l'efficienza del programma
- La prototipazione evita le conversioni automatiche degli argomenti

La ricorsione – 1

- Una funzione è ricorsiva se richiama se stessa

```
void recurse()
{
        static count=1;
        printf("%d\n", count);
        count++;
        recurse();
}

main( )
{
        recurse();
}
```

La funzione stampa il valore di count (che è 1, inizialmente), incrementa count ed infine richiama se stessa; la seconda volta count vale 2… il procedimento viene ripetuto all'infinito

1

2

3

…

Infine, il calcolatore esaurirà la memoria disponibile per lo stack ed il programma verrà interrotto (con segnalazione di errore)

- Nei programmi ricorsivi, è necessario prevedere una condizione di terminazione, altrimenti il programma non si interrompe

La ricorsione – 2

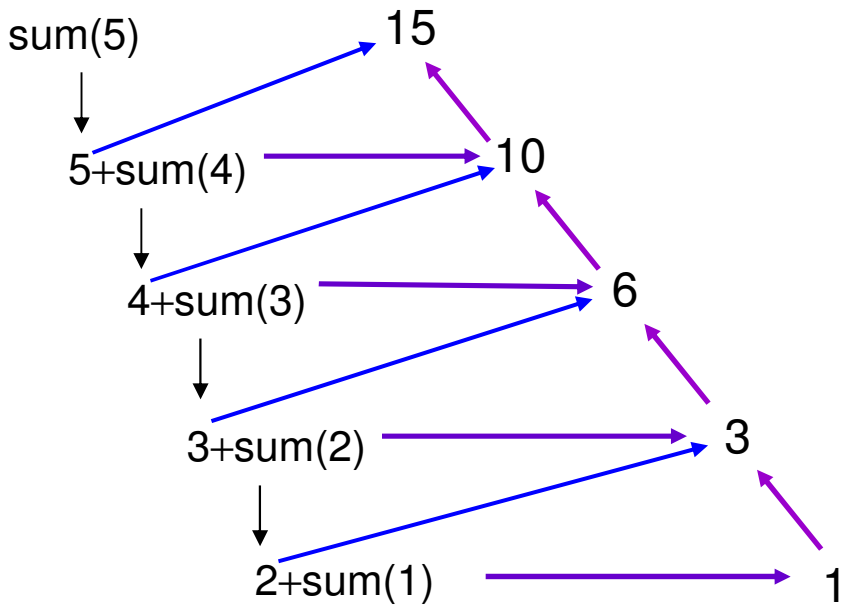
```
void recurse()
{
        static count=1;
        if (count > 3)
            return;
        else
        {
            printf("%d\n", count);
            count++;
            recurse();
        }
}
main( )
{
        extern void recurse();
        recurse();
}
```

- La condizione che conclude la ricorsione è detta condizione base
- Se count avesse durata automatica anziché fissa, il programma non terminerebbe mai perché, ad ogni nuova chiamata, count verrebbe re–inizializzata ad 1
- Nella ricorsione, ad ogni nuova chiamata di funzione, il compilatore crea un nuovo insieme completo di variabili automatiche che, pur avendo lo stesso nome, occupano aree di memoria distinte

Il valore restituito da chiamate ricorsive

- L'uso di variabili fisse è una delle modalità di controllo della ricorsione, l'altra è l'uso di valori d'ingresso
- Esempio: Calcolo della somma degli interi da 1 ad n

```
int sum(n)
int n;
{
        if  (n <= 1)
            return n;
        else
            return (n+sum(n–1));
}
```



Esempi di funzioni ricorsive – 1

1) Fattoriale

```
int fatt (n)
int n;
{
        int t, result;

        result = 1;
        for (t=2; t<=n; t++)
              result *= t;
        return result;
}
```

Funzione non ricorsiva

```
int rfatt (n)
int n;
{
        if  (n==0)
             return 1;
        else
             return (n*rfatt(n–1));
}
```

Funzione ricorsiva

La funzione *main()* – 1

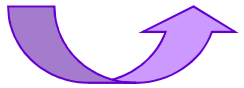
- Tutti i programmi C devono contenere una funzione, il *main()*, che è la prima eseguita all'interno di un programma e la cui conclusione determina la fine del programma
- Il compilatore gestisce *main()* come le altre funzioni, con l'eccezione che, al momento dell'esecuzione, l'ambiente deve fornire due argomenti:
  - argc, è un int che rappresenta il numero di argomenti presenti sulla linea di comando all'atto dell'invocazione del comando
  - argv, è un array di puntatori agli argomenti della linea di comando

La funzione *main()* – 2

- Un puntatore al comando stesso è memorizzato in argv[0]: nell'esempio, è stato utilizzato l'operatore di incremento prefisso per non visualizzare il comando
- Nei sistemi UNIX esiste un programma simile, detto echo, che visualizza gli argomenti della linea di comando

```
main (int argc, char *argv[])
{
/* visualizza  argomenti
della linea di comando */

        while (--argc > 0)
                printf("%s ", *++argv);
        printf("\n");
        exit(0);
}
```



- Quando viene eseguito un programma, gli argomenti della linea di comando devono essere separati da uno o più caratteri di spaziatura

La funzione *main()* – 3

- Gli argomenti della linea di comando sono sempre passati a *main()* come stringhe di caratteri
  - Se rappresentano valori numerici, devono essere convertiti esplicitamente
  - Esistono le apposite funzioni della libreria di run–time: *atoi()* converte una stringa in int, *atof()* converte una stringa in float, etc.

```
#include <math.h>
#include <stdlib.h>

main (int argc, char *argv[])
{
        float x, y;

        if (argc < 3)
        {
          printf("Uso: power <number>\n");
          printf("Restituisce arg1 ^ arg2\n");
          return;
        }
        x = atof(*++argv);
        y = atof(*++argv);
        printf("%f\n", pow(x, y));
}
```