Funzioni

FUNZIONI

Spesso può essere utile avere la possibilità di costruire nuove istruzioni che risolvono parti specifiche di un problema

- Una *funzione* permette di
  - *attribuire un nome ad un insieme di istruzioni*
  - *parametrizzare l'esecuzione del codice*

FUNZIONI

- **Sintassi di definizione di una funzione**

<tipo di ritorno> <nome>(<lista argomenti>){

<sequenza di istruzioni>

- **Esempi**:

float f (){ return (2 + 3 * sin(0.75)); }

float f1 (int x){return(2 + x * sin(0.75)); }

La lista degli argomenti può essere vuota

Quando il tipo di ritorno è int può essere omesso

Argomenti di una funzione

<lista_argomenti> ::=

<tipo> <nome> {, <tipo> <nome> }

La lista degli argomenti può essere vuota

()

(int x)

(int x, int y)

(int x, float y);

E' rilevante l'ordine degli argomenti

Esempi di definizione di funzioni

- **Esempi**:

```
float f (){ 2 + 3 * sin(0.75); }
float f1 ( int x ){
        2 + x * sin(0.75); }
void f2() {printf("ciao");}
```

STRUTTURA DI UN PROGRAMMA C

<programma> ::=

<dichiarazioni globali> <funzione>

{<funzione>}

<funzione> ::=

<tipo_ritorno> <nome_funz>

(<lista_argomenti>)

{ <sequenza_istruzioni> }

Librerie e compilazione separata

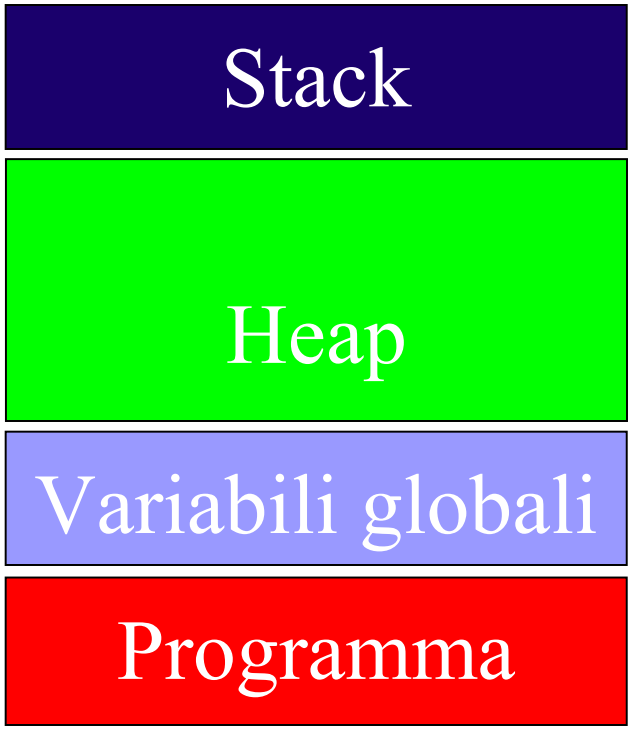
- Compilazione di un programma C

  creazione del programma

  compilazione del programma

  collegamento del programma con le funzioni di libreria richieste

Suddivisone concettuale della memoria per un programma C



Chiamata di una funzione

- Sintassi della chiamata di una funzione

  <nome>(<lista valori>)

- Esempi

```
int funzione1(int x)          int a;
                              a = funzione1(5);

void funzione2(int x);        funzione2(3);
```

Passaggio dei parametri

- Parametri formali:
  - quelli che compaiono nella definizione della funzione
  - sono identificatori che rappresentano variabili
  - si possono usare tipi primitivi, tipi puntatori, tipi struct
  - tali variabili vengono allocate in memoria all'atto della chiamata della funzione
  - all'atto della chiamata della funzione vengono allocate in memoria le variabili locali cioè quelle definite all'interno della funzione
  - parametri formali e variabili locali vengono deallocata alla fine dell'esecuzione della funzione
- Parametri attuali
  - quelli che compaiono nella chiamata della funzione
  - sono espressioni
- Corrispondenza posizionale
  - il numero è il tipo dei parametri formali e dei parametri formali deve essere uguale

Passaggio dei parametri per valore

- Chiamata della funzione

  ogni parametro formale viene allocato in memoria e viene inizializzato al valore del parametro attuale corrispondente

  la funzione agisce sul parametro formale

- Terminazione della funzione

  return

  alla fine della funzione il parametro formale viene rilasciato

Passaggio dei parametri per valore

```
int y = 10;
float a;
a = funzione (y,5.0);
.
.
a = funzione(3+a*2,4.0);
```

```
int funzione (int x; float y) {
    x = x * 2;
    y = x * y;
    return y
```

812.0

10

~~20~~

~~100.0~~

~~203~~

~~812.0~~

```
#include <stdio.h>
int massimo (int a, int b)
{    if (a > b) return a;
     return b;
}

int sommamax (int a1, int a2, int a3, int a4)
{       return (massimo(a1, a2) + massimo(a3, a4));
}

main()
{       int A, B, C, D;
        scanf(%d %d %d %d", &A, &B, &C, &D);
        printf("%d\n", sommamax(A,B,C,D);
}
```

```
#include <stdio.h>
int massimo (int a, int b) {      /*calcola il massimo fra a e b) */
      if (a > b) return a;
      return b;
}

void print_int (int a) {          /*stampa un intero*/
      printf("%d\", a);
      return;
}

void dummy() {          /*stampa stringa "Ciao"*/
      printf("Ciao");
}
```

```
main()
{    int A, B;
     printf(“Dammi A e B”);
     scanf(%d %d”, &A, &B);
     print_int(massimo(A,B);
     dummy();
}
```

```
#include <stdio.h>

main()
{       int A, B;
        scanf(%d %d , &A, &B );
        printf("%d\n", massimo(A,B);
}

int massimo (int a, int b)
{   if (a > b) return a;
    return b;
}
```

In questo caso il compilatore segnala un errore in corrispondenza della chiamata **massimo(A,B)**

Definizione di un prototipo

Descrive la proprietà della funzione senza definirne la realizzazione.
Serve per anticipare le caratteristiche di una funzione definita successivamente.

<tipo> <nome_funzione> (<argomenti>) ;

```
#include <stdio.h>
int massimo (int a, int b);
main()
{       int A, B;
        scanf(%d %d , &A, &B );
        printf("%d\n", massimo(A,B);
}
int massimo (int a, int b)
{    if (a > b) return a;
     return b;
}
```

Array passati come parametri

Poiché un array in C è un puntatore costante che punta ad un'area di memoria pre-allocata di dimensione prefissata, il nome dell'array:

- non rappresenta l'intero array
- è un alias per il suo indirizzo iniziale

int V[16];

V = &V[0] = indirizzo

Quindi passando un array come parametro di una funzione:

- non si passa l'intero array
- si passa solo (per valore) il suo indirizzo iniziale

Array passati come parametri

```
void funzione (int W[dim]);

void main()
{   int V[10];
    ….
    Funzione (V)
    …..
}
```


```
void funzione (int *W);
void funzione (int W[]);
```

Esempio

```
int lunghezza(char s[])
{   int lung = 0;
    for (;s[lung]!='\0';lung++);
    return lung;
}

int lunghezza(char *s)
{   int lung = 0;
    for (;s[lung]!='\0';lung++);
    return lung;
}
```

Esempio

```
/* Visualizza le potenze dei numeri compresi fra 1 e 10 */
#include <stdio.h>
#include <stdlib.h>
#define N 10
#define M 4
int potenza(int a, int b);
void tabella (int p[M][N]);
void stampa_tabella (int p[M][N]);
void main ()
{   int p[M][N];
    tabella(p);
    stampa_tabella(p);
}
```

Esempio

```
int potenza(int a, int b){
    int t=1;
    for ( ; b; b--) t = t*a;
    return t;
}
```

1	2	3	4	…	9	10
1	4	9	16		81	
1	8	27	64			
1	16	81				

```
void tabella (int p[M][N]){
    int i, j;
    for (j = 0; j<N; j++)
        for (i = 0; i<M; i++)
       p[i][j] = potenza(j+1, i+1);
}
```

```
void stampa_tabella (int p[M][N]){
      int i, j;
      for (i = 0; i<M; i++){
            for (j = 0; j<N; j++)
              printf("%10d", p[i][j] );
            printf ("\n");
      }

}
```

Il modello a run-time

- Creazione di una nuova attivazione (istanza) della funzione
- allocazione della memoria per i parametri e per le variabili locali
- passaggio dei parametri
- trasferimento del controllo dalla funzione chiamante alla funzione chiamata
- esecuzione del codice della funzione chiamata

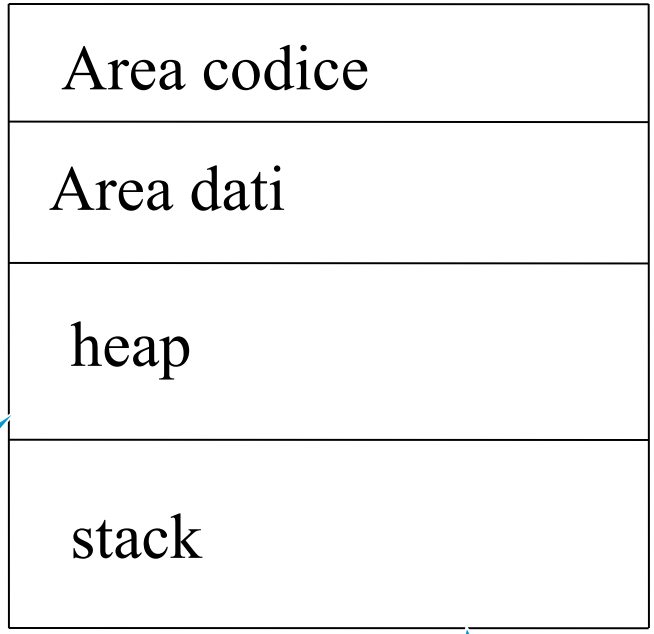
Al momento della chiamata di una funzione si crea un nuovo ambiente

Organizzazione della memoria

Contiene il codice del programma

Contiene le variabili non definite all'interno delle funzioni

Area destinata alle variabili dinamiche



Contiene il record di attivazione delle funzioni

Gestione dello stack

Ogni volta che è chiamata una funzione viene creato un record di attivazione e viene posto in cima alla pila

Il record di attivazione è distrutto al momento in cui termina l'esecuzione di una funzione

La ricorsione

Una funzione matematica è definita per ricorsione (o per induzione) quando è espressa in termini di se stessa.

$$f(n) \begin{cases} 1 & \text{se } n = 0 \\ n * f(n-1) & \text{se } n > 0 \end{cases}$$

La base teorica è il principio di induzione:

se una proprietà vale per un certo naturale $n=n_0$

e si può dimostrare che, assumendola valida per n, essa è valida anche per n+1,

allora la proprietà vale $n \geq n_0$

In pratica, in questo modo si specifica la funzione definendo

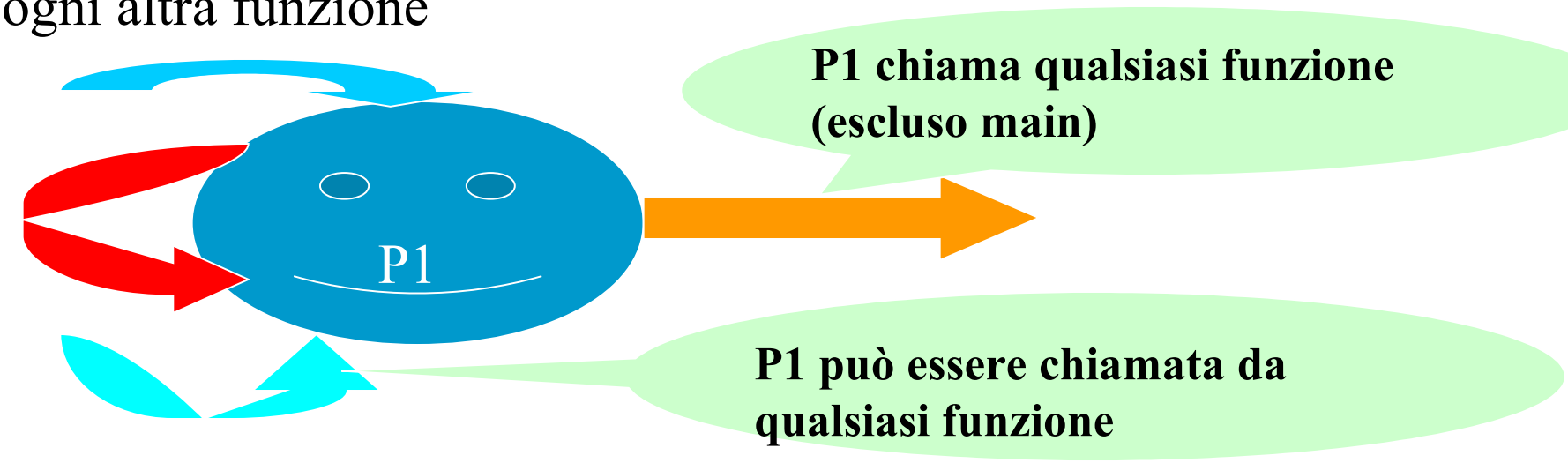
quanto vale in un caso base

come si può ricondurre il generico caso "di grado n" a uno o più casi più semplici (di grado < n).

Ricorsione e programmazione

Una funzione C è una astrazione di funzione matematica.

Ogni funzione (ad eccezione della funzione main) può sempre assumere il ruolo di funzione chiamante e funzione chiamata di ogni altra funzione



Come caso particolare
una funzione può richiamare se stessa

Funzione RICORSIVA

Ricorsione e programmazione

SPECIFICA DELLA FUNZIONE

unsigned long int fattoriale(int n);
<se n vale 0, restituisci 1; altrimenti, calcola il fattoriale di n-1, e restituisci tale valore moltiplicato per n>

LA CODIFICA

```
unsigned long int fattoriale(int n){
    if (n ==0) return 1;
    return n * fattoriale(n-1);
}
```

Ricorsione e programmazione

Calcolare la somma dei primi N numeri positivi

Specifica iterativa
ripeti per N volte l'operazione elementare

sum = sum + i

Specifica ricorsiva
considera la somma (1+2+3+...+(N-1)+N) come composta di due termini, (1+2+3+...+(N-1)), e N
il secondo termine è un valore singolo, il primo non è altro che il risultato dello stesso problema in un caso più semplice
è facile identificare un caso base: la somma fino a 1 vale 1.

Ricorsione e programmazione

```
int somma (int n){
      if (n == 1) return 1;
     return n+somma(n-1)
}
```

OSSERVAZIONI:

l'approccio iterativo richiede di vedere la soluzione del problema "tutta insieme" in termini di mosse elementari

l'approccio ricorsivo richiede invece solo di esprimere il problema in termini dello stesso problema in casi più semplici, più qualche mossa elementare;

inoltre, deve essere identificato un caso banale che non richiede tale decomposizione (per terminare).

Ricorsione e programmazione

Calcolare l'ennesimo numero di Fibonacci

$$fib(n) \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2), & \text{altrimenti} \end{cases}$$

SPECIFICA
<se n vale 0, restituisci 0; se n vale 1, restituisci 1; altrimenti, calcola (N-1)-esimo e (N-2)-esimo numero di Fibonacci, e restituisci la somma di tali valori>

Ricorsione e programmazione

LA CODIFICA

```
unsigned long int fib(int n){
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}
```

NOTA:
questo è un esempio di **ricorsione NON lineare** (la definizione della funzione si basa su **più** chiamate ricorsive)
I due casi precedenti erano invece esempi di ricorsione lineare (una sola chiamata ricorsiva)

Esempi

Calcolo del **minimo di una sequenza di elementi** [a1, a2, a3, ... aN]

Specifica ricorsiva
Considera la sequenza [a1, a2, a3, ..., aN] come composta di due parti:
  il primo elemento, a1
  la sequenza di tutti gli altri, [a2, a3, ..., aN]

Allora:
se la sequenza è lunga 1, il minimo coincide con a1
in ogni altro caso, il minimo è il minore fra a1 e il minimo della sequenza [a2, a3, ..., aN]

Esempi

L'algoritmo per trovare il minimo si esprime allora così:

min([a1, a2, a3, ..., aN]) =
= min(a1, min([a2,a3,... aN])) =
....
= min(a1, min([a2, min([a3,...])])])

NOTA: si inizia a sintetizzare il risultato solo quando
si sono aperte tutte le chiamate (cioè quando si arriva a trovare min([aN]),
che vale aN.

Esempi

Calcolare il Massimo Comune Divisore fra due numeri.
Specifica ricorsiva della soluzion

$$MCD(m,n) \begin{cases} n & \text{se } n = m \\ MCD(m-n, n), & \text{se } m>n \\ MCD(m, n-m), & \text{se } m<n \end{cases}$$

LA CODIFICA

```
int MCD(int m, int n){
    if (n == m) return n;
    if (m > n)   return MCD(m-n,n);
    return MCD(m,n-m)
}
```

Così, ad esempio:
mcd (36, 15) =
= mcd (21, 15) = mcd (6, 15) =
= mcd (6, 9) = mcd (6, 3) =
= mcd (3, 3) = 3
NOTA: il risultato viene sintetizzato via via che le chiamate ricorsive si succedono

Esempio

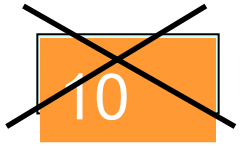
- Scrivere una funzione che date due variabili restituisce le due variabili con il valore scambiato.

```
int main(){
    int x,y;
    x = 3;
    y = 10;
    scambia (x,y);
    printf (“%d”,x);
    printf (“%d”,y);
}
```

```
void scambia(int a, int b){
    int aux;
    aux = a;
    a = b;
    b = aux;
    return;
}
```

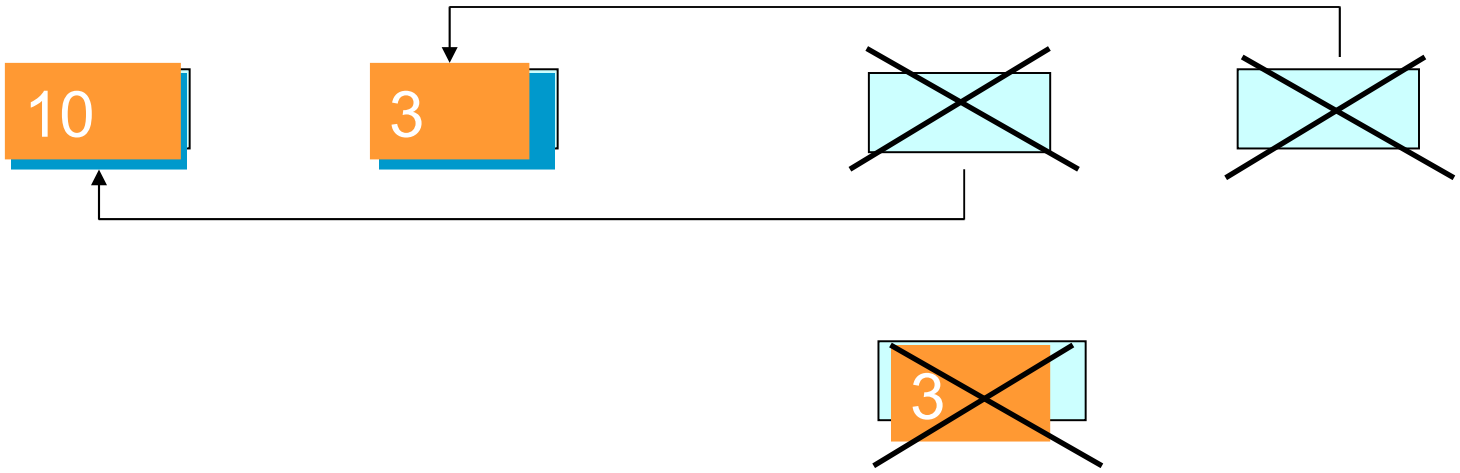
```
int main(){
      int x,y;
      x = 3;
      y = 10;
      scambia (x,y);
      printf (“%d”,x);
      printf (“%d”,y);
}
```

```
void scambia(int a, int b){
      int aux;
      aux = a;
      a = b;
      b = aux;
      return;
}
```



```
int main(){
      int x,y;
      x = 3;
      y = 10;
      scambia (&x,&y);
      printf (“%d”,x);
      printf (“%d”,y);
}
```

```
void scambia(int *a, int *b){
      int aux;
      aux = *a;
      *a = *b;
      *b = aux;
      return;
}
```



Passaggio di parametri per riferimento

- In C non esiste il passaggio di parametri per riferimento
- Ogni qualvolta un parametro di una funzione è un parametro di uscita o un parametro di ingresso/uscita è necessario passare per valore il puntatore alla variabile che deve essere modificata
- Deve essere quindi utilizzato
  - come parametro formale un puntatore di tipo appropriato
  - come parametro attuale l'indirizzo della variabile che la funzione deve modificare

Esempi

- void scambia(int *x, int * y);
- scambia (&a,&b) /*a e b sono variabili int*/

- **int scanf(char * stringa, tipo *a1)**
- **scanf(“%d”, &a)**