Array e Stringhe

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 23 maggio 2010
http://disi.unitn.it/~agiordani/

Ripasso sulle funzioni

Una funzione é un insieme di istruzioni che realizzano un algoritmo. È caratterizzata dalla sua signature:

- parametri formali: il tipo dei parametri dell'algoritmo (es: funzione che calcola il delta di un'eq di secondo grado, deve conoscere i coefficienti dell'equazione)
- nome: un nome arbitrario per identificarla (es: `delta`)
- ritorno: il tipo del risultato ritornato dalla funzione (es: `double`)

Una funzione senza argomenti é detta procedura

Ciclo di vita di una funzione: dichiarazione, implementazione, invocazione

Dichiarazione di una fuzione

Informa il compilatore che esiste una funzione con una certa signature:
`<tipo di ritorno> <nome> ( <parametri formali> );`

Le regole per la validità dei nomi di funzioni sono le stesse che nel caso delle variabili. Alcuni esempi:

```
double delta(double a, double b, double c);
int max(int x, int y);
double radice_quadrata(double x);
```

- La dichiarazione di una funzione non dice nulla sulla sua semantica (cosa fa) né sulla sua implementazione (come lo fa)
- black-box reuse: se conosciamo la signature di una funzione e sappiamo cosa fa (documentazione), possiamo usarla anche ignorandone l'implementazione

Utilizzare le funzioni

- Programma main per il calcolo delle radici di un equazione di 2°grado.
- Questo programma è una possibile implementazione

```
#include <stdio.h>
#include <math.h>
int main()
{
float x1,x2,a=2,b=3,c=1,delta;

delta=b*b-4*a*c;
if (delta<0)
      printf("Non esistono radici reali\n");
      else
      {
      if (delta==0)
                x1=x2=-b/(2*a);
           else
                {
                x1=(-b+sqrt(delta))/(2*a);
                x2=(-b-sqrt(delta))/(2*a);
                }
      printf("x1=%f,x2=%f\n",x1,x2);
      }
return 0;
}
```

- Dichiarazione
- Utilizzo
- Definizione

```c
#include <stdio.h>
#include <math.h>
double delta (double a,double b,double c);

int main()
{
float x1,x2,a=2,b=3,c=1,d;
d=delta(a,b,c);
if (d<0)
      printf("Non esistono radici reali\n");
      else
      {
      if (d==0)
                x1=x2=-b/(2*a);
           else
                {
                x1=(-b+sqrt(d))/(2*a);
                x2=(-b-sqrt(d))/(2*a);
                }
      printf("x1=%f,x2=%f\n",x1,x2);
      }
return 0;
}

double delta (double a,double b,double c)
{
 double res;
 res=b*b-4*a*c;
 return res;
}
```

Memorizzazione Variabili

- La dichiarazione di una variabile associa un tipo di dato ad un nome simbolico
- La dichiarazione “alloca” (cioè “riserva”) memoria per una variabile di un certo tipo
- L'allocazione della memoria in C è contigua
  - un dato *int* viene memorizzato in 4 byte di memoria adiacenti
  - un dato *double* in 8 byte di memoria adiacenti
  - un *array di 5 interi* in 20 byte di memoria adiacenti
  - …

Memorizzazione Array 1/2

- In C gli array sono intrinsecamente legati al concetto di **puntatore**
- Per capire veramente come funzionano gli array (e per sfruttarli a dovere) bisogna capire i puntatori
- L'utilizzo dei puntatori (e dell'allocazione dinamica della memoria) consentono un uso molto più flessibile degli array

Memorizzazione Array 2/2

- La dichiarazione di un array di n elementi di tipo T causa l'allocazione contigua di **n x sizeof (T)** byte. Ad es:

```
int myarr[7];
```

- dichiara un array di 7 interi chiamato myarr, e alloca 7xsizeof(int) = 28 byte di memoria contigua
- Il nome dell'array (`myarr`) è un puntatore al 1°byte di memoria allocata
  - L'indirizzo del 2° elemento dell'array e dato da `myarr + 1`
  - L'indirizzo del 3° elemento dell'array e dato da `myarr + 2`
  - … e così via, ma lo vedremo meglio più avanti!
- Attenzione: il 1° elemento dell'array ha indice 0, non 1!

Accesso ad elementi di array

Le parentesi quadre [*n*] si utilizzano:

- Per dichiarare un array di un *n* elementi
  - int a[10]; *//n deve essere un intero!*
- Per accedere al'*n*-esimo elemento dell'array
  - Sia in lettura (se a dx di un = o in printf)
  - Sia in scrittura (se a sx di un = o in scanf)
  - a[5] = a[5] + 1;

Ricerca del minimo in un vettore

- Il vettore si può inizializzare in fase di dichiarazione
- Non occorre dichiarare la dimensione tra []
- Per inizializzarlo a valori 0 usare static

```
#include <stdio.h>
int main()
{
     float min,v[]={3,4.5,10,2,9};
     int i,n=5;

     min=v[0];
     for (i=1;i<n;i++)
     {
          if (min>v[i])
            min=v[i];
     }
     printf("Min = %f\n",min);
     return 0;
}
```

Calcolo del vettore somma

```c
#include <stdio.h>
#define MAX 5

int main()
{
    int a[MAX],b[MAX],c[MAX];
    int i,tmp;

    for (i=0;i<MAX;i++)
    {
        printf("Inserire il %d° numero del vettore a: ", i+1);
        scanf("%d",&tmp);
        a[i]=tmp;
    }
    for (i=0;i<MAX;i++)
    {
        printf("Inserire il %d° numero del vettore b: ", i+1);
        scanf("%d",&tmp);
        b[i]=tmp;
    }
    printf("\nContenuto del vettore c:\n", i+1);
    for (i=0;i<MAX;i++)
    {
        c[i]=a[i]+b[i];
        printf("[%d]",c[i]);
    }
    printf("\n");
    return 0;
}
```

//o equivalentemente `scanf("%d",a+i);`

Inversione di un vettore

- Leggo *dim* numeri (dim<10)
- Li inserisco via via nel vettore *a*
- Inserisco in *b* gli elementi di *a* in ordine inverso
- Stampo vettori

```c
#include <stdio.h>
int main()
{
    int a[10], b[10];
    int i, dim=5;
    /*input elementi*/
    for(i=0;i<dim;i++)
    {
        printf("Inserire l'elemento a[%d]: ",i);
        scanf("%d", &a[i]);
    }
    /*stampa vettore*/
    printf("\nIl vettore e':\n\n");
    for(i=0;i<dim;i++)
        printf("a[%d] = %d\n", i, a[i]);
    printf("\n");
    /*inversione vettore*/
    for(i=0;i<dim;i++)
        b[i]=a[dim-1-i];
    /*stampa vettore invertito*/
    printf("Il vettore invertito e':\n\n");
    for(i=0;i<dim;i++)
        printf("b[%d] = %d\n", i, b[i]);
    return 0;
}
```

Input di stringhe

- Queste considerazioni sono particolarmente delicate nel caso in cui vogliamo usare scanf per leggere stringhe
- La sequenza %s all'interno di un template consuma qualunque sequenza di caratteri che non contenga spazi
- Per leggere una stringa, dobbiamo prima avere allocato un buffer (cioe un array di caratteri) abbastanza capiente. Ad esempio:

```
char buf [256]; // crea un buffer
scanf("%s", buf);
```

- Se la stringa immessa dall'utente è piu lunga del buffer (nell'esempio la lunghezza massima è 255, tenendo conto del null character '\0') abbiamo un errore di **segmentation fault** (violazione di accesso in memoria)

Inversione di una stringa

- Leggo una parola *a* di lunghezza sconosciuta (dim max 20)
- Inserisco in *b* i caratteri di *a* in ordine inverso
- Stampo stringhe

```c
#include <stdio.h>
#define MAX 20
int main()
{
    char a[MAX], b[MAX];
    int i, dim;
    /*leggo parola*/
    printf("Inserire una parola (dim.max %d): ",MAX-1);
        scanf("%s", a);
    /*stampo parola - e calcolo lunghezza*/
    printf("Ho letto: '");
    for(i=0;a[i]!='\0';i++)
        printf("%c", a[i]);
    dim=i;
    printf("' lunga %d caratteri\n",dim);
    /*inverto caratteri*/
    for(i=0;i<dim;i++)
        b[i]=a[dim-1-i];
    b[dim]='\0';
    /*stampo parola invertita*/
    printf("Ecco la parola invertita '%s'\n",b);
    return 0;
}
```

Esempi di esami

All'indirizzo

http://disi.unitn.it/~agiordani/teaching11.htm

troverete tracce e soluzioni degli appelli degli anni passati.