Funzioni, puntatori, strutture

sommario

In questa lezione integriamo le informazioni fornite su puntatori e funzioni approfondendo i seguenti punti

- Puntatori a puntatori
- Puntatori e array come parametri di funzioni
- Il qualificatore const
- Puntatori a funzione
- Funzioni di funzione
- Strutture
- Typedef

Puntatori a puntatori

- Un puntatore non e' altro che una variabile, che viene salvata in un certo indirizzo
- E' quindi possibile definire un altro puntatore in cui salvare l'indirizzo del primo puntatore

```
int a[5];
int *pi; **pk;
a[0]=137;
pi=a;
pk=&pi;
```

Puntatori come parametri

- Il passaggio dei parametri in C avviene sempre *by value*
- Se un parametro della funzione e' un puntatore la funzione se ne copia il valore e lo puo' usare per accedere al contenuto della locazione, come nel caso di

```
void test2(double A[],int *Nadd)
```

- Esempio:

```
void swap(double *a, double *b){
    double temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

- `swap(&A,&B)` scambia i valori di A e B

Array come parametri

- Il nome dell'array e' il puntatore all'array stessa, quindi data un'array `int data[10]` possiamo passarne l'indirizzo alla funzione `f` che ha come prototipo `void f(int * d)` semplicemente chiamando `f(data)`.
- Di solito si preferisce il prototipo `void f(int data[])` che indica esplicitamente che si tratta di un vettore
- Tipicamente e' necessario aggiungere un parametro intero per indicare il numero degli elementi dell'array da accedere nella funzione (<= alla dimensione)
- Poiche' si trasmette un puntatore la funzione ha la possibilita' di alterare il contenuto degli elementi del vettore...

Il qualificatore const

- Per impedire (nello stadio di compilazione) la modifica del valore di una variabile questa puo' essere dichiara const

```
const float pi=3.1415;
pi=0; // questa istruzione non compila
```

- Cio' e' particolarmente utile non tanto per le costanti numeriche (definibili anche tramite la direttiva `#define`) quanto per i puntatori (a variabili singole o ad array)
  - `char * const p1;` // p1 puntatore costante a char
  - `char const *p2;` // p2 puntatore a carattere costante
  - `const char *p3;` //p3 puntatore a carattere costante
- Per proteggere il contenuto di una locazione di memoria si usa quest'ultima espressione, come nel caso di

```
void test3(const double A[], int N);
```

Puntatori a funzione

- La memoria assegnata ad un programma contiene variabili ed istruzioni del programma stesso
- Al punto di entrata di una funzione corrisponde un indirizzo di memoria ben preciso che in C e' associato al nome della funzione
- Possiamo definire un puntatore ad una funzione come una variabile contenente l'indirizzo della funzione cui punta. Per una funzione di tipo int ad esempio

```
int (*pf)(); /* attenzione alle parentesi */
```

- E inizializzarlo ponendo

```
pf=&nome_della_funzione;
```

- Si puo' passare una funzione ad un'altra funzione come parametro di input!

Funzioni di funzione: esempio 7.12 dal testo Barone et al.

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

double func(double);
double derivative(double (*)(double), double, double);

main() {
  double epsilon = 1.;
  printf("epsilon      derivata\n\n");
  while (epsilon > 1.e-18) {
    printf("%e %f\n", epsilon, derivative(func, 1., epsilon));
    epsilon /= 10.;
  }
}

double func(double x) {
  return sqrt(x);
}

double derivative(double (*f)(double), double x, double epsilon) {
  return ( f(x + epsilon) - f(x) ) / epsilon;
}
```

Struct

- Abbiamo visto
  - variabili singole di tipi predefiniti (int, float...)
  - array di variabili tutte dello stesso tipo
- Abbiamo visto che le funzioni possono ricevere in input variabili singole o puntatori a variabili o array, e che possono restituire in output variabili singole o puntatori (solo nel caso di blocchi di memoria riservati dinamicamente o di array dichiarate globali)
- Possiamo definire dei nuovi tipi, detti strutture, e scambiare le variabili di questi tipi sia in input che in output.

Struct

- Le strutture contengono piu' variabili anche di tipi diversi associate tra loro per rendere agevole la loro manipolazione da parte di funzioni
- Esempio di definizione di una struttura:

```
struct punto2D {
  double X;
  double Y;
};
```

- Esempio di dichiarazione di una variabile del nuovo tipo:

```
struct punto2D origine;
```

Struct

- Esempio di accesso ai membri di una struttura:

```
origine.X=0;
origine.Y=0;
printf("X0=%lf,Y0=%lf",origine.X,origine.Y);
```

- si puo' definire un puntatore ad una struttura:

```
struct punto2D * punto;
punto=&origine;
```

- ed accedere ai membri della struttura in altri 2 modi diversi:

```
(*punto).X e punto->X
```

typedef

- In C possiamo definire dei tipi derivati, cio' serve soprattutto ai fini della compatibilita' tra diverse piattaforme ma anche per una maggiore leggibilita' del codice
- Ad esempio definendo

```
typedef float posizione;
typedef float tempo;
typedef float velocita;
```

Potremo dichiarare in un nostro programma per lo studio del moto di un grave variabili razionali come di tipo `posizione`, di tipo `tempo` e di tipo `velocita`

esempio: gettimeofday

•La libreria standard del C include funzioni chiamabili per conoscere data e ora sul vostro sistema (vedi nota sulla misura del tempo nei computer a fine paragrafo 5.6 del testo Barone et al.)

•Le definizioni utili sono in /usr/include/sys/time.h, che si include con la direttiva
`#include <sys/time.h>`

•una funzione utile (ivi dichiarata) e'

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

(digitate `man gettimeofday` per ulteriori dettagli)

che utilizza in input e in output le strutture

```
struct timeval {
        time_t          tv_sec;         /* seconds */
        suseconds_t     tv_usec;  /* microseconds */
                };
```

e timezone (che non vi serve nell'uso pratico).

time_t e suseconds_t sono dei nuovi tipi dichiarati con dei typedef.

•Una tipica chiamata sara'

```
timeval t0;
gettimeofday(&t0,NULL);
```

Linguaggio C

- Con questa lezione si conclude la panoramica sul linguaggio C
- Abbiamo incluso anche elementi “avanzati” che nel testo trovate nella seconda parte (prenotazione dinamica della memoria e strutture)
- Vedremo qualche applicazione cercando di utilizzare tutti gli strumenti del linguaggio a nostra disposizione
- Non ci sara' tempo per esaminare in dettaglio tutte quelle proposte dal testo Barone et al. ma voi studiatele!

Applicazioni

- esempi di applicazioni alla vostra portata sviluppati nel testo di Barone et al.:
  - Ricerca degli 0 di una funzione mediante il metodo di bisezione e mediante il metodo di Newton (4.3.2)
  - Ricerca dei numeri primi (4.3.3)
  - Problemi di arrotondamento (4.5)
  - Riordinamento degli elementi di un vettore: Bubblesort (5.2.1)
  - Ricerca binaria (5.2.2)
  - Soluzione di sistemi di equazioni lineari (5.4, 7.3.2)
  - Generazione di numeri casuali (5.5) - Visto
  - Manipolazione di testi (5.6.3, 6.5.1)
  - Istogrammi (7.4.1) – Visto
  - Calcolo del χ^2 di una distribuzione (7.4.2)
  - Calcolo di una derivata (7.6)
  - Interpolazione e integrazione numerica (cap. 8)