Sistemi Operativi

Unità 3: Programmazione in C

I puntatori

Martino Trevisan
Università di Trieste
Dipartimento di Ingegneria e Architettura

Argomenti

1. Puntatori in C
2. Puntatori e vettori
3. Puntatori e stringhe
4. Puntatori e funzioni
5. Puntatori a `struct`
6. Puntatori a funzione

Puntatori in C

Un **puntatore** è una variabile che contiene un indirizzo di memoria.
Non ci interessa come è strutturato l'indirizzo

- Esso è comunque un indirizzo virtuale, che viene tradotto in un indirizzo fisico dalla Memory Management Unit

In C, una variable puntatore contiene un indirizzo di memoria dove è contenuta una variabile di un certo tipo.

- Tutte le volte che dichiaro un puntatore, devo dichiarare anche che tipo di dato contiene l'indirizzo di memoria che contiene
- Fondamentale per l'utilizzo pratico

Puntatori in C

Dichiarazione di un puntatore: `tipo * nome;`

Esempio:

```
int * pi; // Puntatore a int
float * pf; // Puntatore a float
```

Il tipo `int *` indica una variabile puntatore a intero: contiene l'indirizzo di memoria alla quale troviamo una variable intera.

Puntatori in C

Assegnazione un puntatore: si può usare l'operatore `&` per ottenere l'indirizzo di una variabile esistente.

Esempio:

```
int a = 5;
int * pi;
pi = &pa; // pi contiene l'indirizzo di a
```

Nota: gli indirizzi sono dei numeri interi. Non è dato sapere quanto lunghi, dipende dall'architettura.

Puntatori in C

Accesso alla variabile puntata: l'operatore `*` applicato a un puntatore serve per ottenere il variabile puntata. Detto **operatore di dereferenziazione**.

Esempio:

```
int a = 5, b;
int * pi;
pi = &a; // pi contiene l'indirizzo di a
b = *pi; // b contiene il valore 5
```

L'operatore `*` è l'inverso di `&`:

Pertanto: `*(&a) == a` e `&(*pi) = pi`

Puntatori in C

Esempio:

```c
#include <stdio.h>

int main ()
{
    int a=5, *p;

    p = &a;
    printf("a=%d\n", a);   // Stampa: 5
    printf("p=%p\n\n", p); // Stampa un indirizzo, e.g., p=0x7ffc6de0703c

    printf("&a=%p\n", &a); // Stampa lo stesso indirizzo, e.g., p=0x7ffc6de0703c
    printf("*p=%d\n", *p); // Stampa: 5

    return 0;
}
```

Puntatori in C

Osservazione: ora appare più chiaro perchè nella funzione `scanf` bisogna usare l'operatore `&` per passare gli argomenti in lettura.

```
float a;
scanf("%f", &a);
```

Significa che la funzione `scanf` riceve come argomento l'indirizzo di una variabile `float`.
La funzione scriverà in quell'indirizzo il valore letto da tastiera.
Internatmente effettuerà un'operazione del tipo:

```
*pf = valore;
```

Puntatori e vettori

In C, puntatori e vettori hanno una stretta relazione.

Il nome di un vettore senza indice, ritorna l'indirizzo del primo elemento del vettore.

Esempio:

```
int v[5] = {5,6,7,8,9};
int * pi;
pi = v; // Operazione consentita
printf("%d\n", *pi); // Stampa: 5
```

Puntatori e vettori

Dato il vettore: `int v[5]`, sono equivalenti - `v` e `&(v[0])` - `v[0]` e `*v`

Aritmetica dei puntatori

E' possibile sommare interi a un puntatore. Si accede alle locazioni contigue. Sono quindi equivalenti:

- `&(v[2])` e `v+2`
- `v[2]` e `*(v+2)`

Si può iterare su un vettore facendo:

```
for (i=0; i<N; i++)
    v[i] = ...
    ... equivale a ...
    *(v+i) = ...
```

Puntatori e vettori

Differenze tra puntatori e vettori:

- Un puntatore può essere riassegnato per puntare a un altro indirizzo
- Un vettore tecnicamente è un puntatore costante. Non gli può essere assegnato un altro valore.

```
char v[10], *pv;
pv = v;     // Consentito
pv = v + 3; // Consentito
v = pv;     // Errore!
v = pv + 2; // Errore!
```

Puntatori e stringhe

Sappiamo che una stringa è un vettore di `char` terminato dal terminatore `'\0'`.
Un puntatore a `char` può riferirsi a una stringa.

```
char stringa [] = "ciao";
char * ps = stringa;
```

Il puntatore `ps` contiene l'indirizzo del primo elemento di `stringa`.
Tutte le funzioni di manipolazione delle stringhe prendono come argomento un puntatore a `char`

```
strlen(stringa);
```

La funzione `strlen` prende come argomento un `char *`

Puntatori e stringhe

Errori gravi:

Dereferenziare un puntatore non inizializzato:

```
int * pi;
int a = *pi; // Errore! pi contiene un indirizzo a caso!
```

Dereferenziare un intero:

```
int i = 12;
int j = *i; // Errore: i contiene 12, non un indirizzo
```

In questo caso, il compilatore solleva un errore.

Puntatori e funzioni

I puntatori si usano per passare parametri **per riferimento**.
In questo modo, la funzione può modificare gli argomenti che riceve e il chiamante vederne gli effetti.

- Come la `scanf` che modifica il valore di una variabile argomento.
- Tecnica usata quando una funzione deve ritornare più di un valore
  - I valori di ritorno aggiuntivi sono puntatori forniti dal chiamante
  - In cui la funzione colloca il risultato

Puntatori e funzioni

Esempio: si scriva una funzione che prende due interi per riferimento e ne scambia il valore.

```
void swap ( int * a, int * b ){
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Utilizzo:

```
int i = 5;
int j = 7;
swap (&i, &j);
```

Dopo l'esecuzione: `i=7` e `j=5`

Puntatori e funzioni

Esempio: si scriva una funzione che calcola la lunghezza di una stringa terminata da `'\0'`.

```
void len ( char * s ){
    int i = 0;
    while ( *(s+i) != '\0' ) // Aritmetica dei puntatori
        i++;
    return i;
}
```

Utilizzo:

```
char stringa [] = "ciao!";
int a;
a = len(stringa); /* Non è necessario l'operatore &.
                     Il nome di una variabile vettore
                     già indica l'indirizzo del primo elemento */
```

Puntatori a `struct`

Puntatori a `struct`: un puntatore può tranquillamente puntare una `struct`.

```
struct punto {float x; float y;}
struct punto p1 = {1, 4};
struct punto * pp;
pp = &p1; // Assegno a pp l'indirizzo di p1
```

Se può accedere agli elementi di una struct tramite puntatore.

```
(*pp).x; // Contiene il valore 1
(*pp).y; // Contiene il valore 4
```

Puntatori a `struct`

Operatore `->`: si utilizza su puntatori a `struct`.
permette di accetere direttamente a un campo della `struct` puntata.

Sintassi: `puntatore->campo`

Esempio:

```
struct punto {float x; float y;}
struct punto p1 = {1, 4};
struct punto * pp = &p1;
```

Le seguenti istruzioni si equivalgono e ritornano l'`int` 1.

```
(*pp).x;
pp->x;
```

Puntatori a funzione

E' possibile passare delle funzioni come argomento a una funzione.
Si usa per rendere il codice generico e modulare, per il multithreading, ecc...

Vedremo qualcosa quando ci occuperemo di *thread*.

- Esempio: utilizzo una funzione per creare un thread, e come argomento le specifico quale funzione del mio programma il thread esegue.

```
void mytask (){....}
pthread_create(..., mytask, ...)
```

Puntatori a funzione

Un **puntatore a funzione** rappresenta la posizione di una funzione.

Dereferenziarlo significa **invocare** la funzione.

Un puntatore a funzione è tipizzato:

- Può puntare funzioni che ritornano un tipo ben definito
- E accettano un certo tipo di argomenti

Puntatori a funzione

Dichiarazione:

La sintassi è:

`tipoDiRitorno (* nome) (tipoArg1, tipoArg2, ...)`

Esempio:

```
int (*pf) (int, int);
```

Dichiara il puntatore a funzione di nome `pf` che può puntare a funzioni che:

- accettano due `int` come argomento
- ritorano un `int`

Puntatori a funzione

Assegnazione e utilizzo: si usano gli operatore `=` per assegnazione (senza `&`) e `*` per dereferenziazione.

Esempio:

```
int somma (int a, int b){return a+b;}
...
int (*pf) (int, int); // Dichiarazione

pf = somma;         // Assegnazione. Non serve &
res = (*pf)(3,5); // Invoca la funzione
                    // res contiene 8
```

Puntatori a funzione

Funzioni che accettano come argomento puntatori a funzione:

E' uno degli utilizzi più frequenti. Rendono generiche le funzioni.

Esempi:

- Funzione che ordina secondo un criterio fornito dall'utilizzatore
- Funzione del SO che avvia un thread che esegue una funzione fornita dall'utente

Puntatori a funzione

Esempio:

```c
#include <stdio.h>

void combineAndPrint(int a, int b, int (*comb)(int,int) ){
    int p = (*comb)(a, b); // Dereferenziazione di comb
    printf("Combinazione: %d\n", p);
}

int add(int a, int b) {return a+b;}
int mult(int a, int b){return a*b;}

int main(){
    combineAndPrint(3, 4, mult); // Stampa 12
    combineAndPrint(3, 4, add); // Stampa 7
    return 0;
}
```

Nota:

`combineAndPrint(3, 4, mult);` e `combineAndPrint(3, 4, &mult);` sono equivalenti