Programmare in C

Gemma Martini

23/04/2014

1 Da terminale

1.1 Comandi generici, per orientarsi tra le cartelle

- cd "nome directory", *entra nella directory*
- cd ../, *torna nella directory precedente*
- mkdir "nuovo nome", *crea una nuova directory*
- mv "nome file" "nome directory", *sposta quel file nella directory citata*
- ls, *fa vedere le cartelle*

1.2 Per compilare

- gedit "nome file.c" `&`, *crea un file in C, lo apre e libera il terminale*
- gedit "nome file".cpp `&`, *crea un file in C++, lo apre e libera il terminale*
- gcc "nome file".c -o "nome file", *compila il file in C*
- g++ "nome file".cpp -o "nome file", *compila il file in C++*
- ./"nome file", *esegue il programma*

2 Linguaggio C - parte operativa

2.1 Tipi di variabili

Interi

- *short*. Per conoscere il range di validità di tale dichiarazione si usano i comandi *SHRT—MIN* e *SHRT—MAX*, che si trovano nel file *limits.h*
- *int*. Per conoscere il range di validità di tale dichiarazione si usano i comandi *INT—MIN* e *INT—MAX*, che si trovano nel file *limits.h*

- *long*. Per conoscere il range di validità di tale dichiarazione si usano i comandi *LONG—MIN* e *LONG—MAX*, che si trovano nel file *limits.h*

Interi senza segno

- `unsigned short`. Per conoscere il range di validità di tale dichiarazione si usa il comando *USHRT—MIN*, che si trova nel file *limits.h*, poichè il minimo è 0.
- *unsigned int*. Per conoscere il range di validità di tale dichiarazione si usa il comando *UINT—MIN*, che si trova nel file *limits.h*, poichè il minimo è 0.
- *unsigned long*, Per conoscere il range di validità di tale dichiarazione si usa il comando *ULONG—MIN*, che si trova nel file *limits.h*, poichè il minimo è 0.

Reali

float, *double*, *long double*. Per trattare numeri reali con più o meno cifre decimali.

Costanti

- const "tipo di variabile" PiGreco = 3.14.
- #define GEMMA 3.
- Da fare subito dopo aver dichiarato le librerie da utilizzare. Le costanti NON possono essere modificate durante l'esecuzione del programma.

Caratteri

- *char*. E' la dichiarazione di variabile associata ai caratteri. N.B. Lo spazio è a tutti gli effetti considerato un carattere.
- *'a'*. Per conoscere il valore numerico associato al carattere *a* nel codice ASCII.
- *sizeof(char)*. Per conoscere i bit con cui viene rappresentata una variabile di tipo *char*.

2.2 Input ed output

Scanf

- La **sintassi** di questo comando è la seguente: scanf("%pippo", &nome variabile).

- Chi è **pippo**?
  - * *non legge quello che viene immesso*;
  - f, e *per le variabili float*;
  - lf, le *per le variabili double*;
  - Lf, Le *per le variabili lng duble*;
  - X, lX, hX *per i numeri in esadecimale*;
  - o, lo, ho *per i numeri in ottale*;
  - c *per i caratteri.*

Printf

- La **sintassi** di questo comando è la seguente: scanf ("%pippo", nome variabile). **ATTENZIONE**, non ci vuole il % nel printf.
- Chi è **pippo**?
  - f, *per le variabili float* (es. "%8.3f" = il numero sarà scritto con 8 cifre, di cui 3 decimali);
  - e *per la notazione esponenziale*;
  - lf, le *per le variabili double*;
  - Lf, Le *per le variabili long duble*;
  - X, lX, hX *per i numeri in esadecimale*;
  - o, lo, ho *per i numeri in ottale*;
  - c *per i caratteri.*

2.3 Switch

La **sintassi** di questo comando è la seguente:

{
switch (espressione)
case valore 1[1];
    break;
.
.
.
case valore n: istruzione ;
    break;
default: istruzione d;
}

[1]NON ci vogliamo una variabile, ma possiamo metterci una cosa del tipo *case val1:...:case valk*

2.4 For - Zucchero sintattico

La **sintassi** di questo comando è la seguente:

{
for (inizializzazione i; condizione per rimanere nel ciclo; operazione su i)
{
Corpo del ciclo
}
}

2.5 Abbreviazioni utili

- a = a+b; *è equivalente a* a+=b;
- a = a-b; *è equivalente a* a-=b;
- a = a*b; *è equivalente a* a*=b;
- a = a/b; *è equivalente a* a/=b;
- a = a %b; *è equivalente a* a%=b;
- if(x>y) z=y;
  else z=x; *è equivalente a* z=(x>y) y:x;

2.6 Array

La **sintassi** per usare questo tipo di dato è la seguente:

- La *dimensione* (numero di elementi dell'array) può essere impostata nel seguente modo: # define DIM 10
- **tipo** nome[DIM];
- nome[i], con $0 <= i <= DIM - 1$;

L'**utilizzo** di un array è il seguente:

- *Come si acquisisce un array dall'esterno:*

```
{
int v[DIM], i
for (i=0; i < DIM; i++)
    {
    printf("Inserisci l'elemento di indice %d\n",i);
    scanf("%d", &v[i]);
    }
}
```

- *Come si dichiara un array costante:*

  {

  int n[4] = {11,22,33,44};

  }

- *Come si conta il numero di occorrenze di un certo carattere c in un array:*

  freq[c];

Verificare una proprietà P(j) negli array

Talvolta capita di voler cercare gli elemnti di un array che soddisfino una certa proprietà, ma per fare questo occorre stare attenti, perchè c'è una discriminazione preliminare da fare:

- **Siamo sicuri che la proprietà sia verificata almeno una volta**
  In tal caso si fa un ciclo che termina non appena la proprietà si verifica;

- **Altrimenti**
  Il ciclo può terminare in due casi: la proprietà P(j) è verificata oppure siamo arrivati alla posizione DIM -1 *dell'array*;

2.7 Funzioni

- *Per dichiarare una funzione si specifica il* **tipo** *di variabile che vogliamo come risultato, il* **nome** *della funzione e, tra parentesi, il* **tipo** *di variabile che la funzione prende in input ed il* **nome** *della stessa. Ecco un esempio*

  int Somma(**int** a, **int** b)

  {

  int s=a+b;

  return s;

  };

- *Come si usa:* sum = Somma(a, b);

- *La funzione restituisce un valore se specifichiamo, prima di chiudere le graffe, tramite il comando* ***return****, la variabile scelta*;

- *Passare un vettore ad una funzione - un esempio*

  FunzioneVettore(v[], DIM)

  {

  [...];

  };

- *Passare una matrice ad una funzione - un esempio*

  FunzioneMatrice(A[],COLONNE,RIGHE)

  {

  [...];

  };

Funzione chiamata VS Funzione chiamante

La **prima** è una sottofunzione che, nell'istante in cui viene chiamata, si avvia, mettendo in pausa la funzione precedente, ossia la *Funzione chiamante*. La **seconda** è per esempio **main**, che permette l'assegnamento di valori alle variabili. **Variabili locali e blocchi.** Un programma in C è strutturato a blocchi e dentro ogni blocco ogni variabile può essere ripetuta una sola volta.

Ambienti:

Globale. Parte dichiarativa del main;
Locale di funzione. Parte dichiarativa della funzione;
Locale di un blocco. L'insieme di tutti gli elementi dichiarati nel blocco.

Variabili:

Automatiche. Hanno vita solo nel blocco in cui sono dichiarate;
Statiche. Durano per tutto il programma(**static int** x).

Funzioni ricorsive

Scrivere una funzione utilizzando il **metodo ricorsivo** permette di utilizzare un linguaggio conciso ed efficace, lasciando il "lavoro sporco" al calcolatore. É molto semplice, poichè si tratta soltanto di formalizzare un problema utilizzando l'**induzione**, per poi tradurre il tutto in C.

2.8 Puntatori

- `&`a *fa riferimento alla cella in cui è memorizzata la variabile a*;
- *b *indica cosa sta puntando la variabile, ossia il contenuto della cella di memoria di indirizzo* `&`a;

- *&a = a;
- &b *ha lo stesso valore di b, ma non è una variabile*;
- **NON FARE!!** a = pi; *perchè sono due tipi diversi*;
- SINTASSI:*un puntatore si definisce nel segunte modo:*

  int* pi; **float*** pi; **char*** pi;
- *Ogni volta che assegno ad un puntatore un altro puntatore questi* **cambiano contemporaneamente**: pi=qi, allora OGNI cambiamento fatto su pi si ripercuote su qi;
- **sizeof**(*pi) != **sizeof**(pi), *poichè uno occupa lo spazio del tipo di ciò che è puntato, mentre l'altro occupa lo spazio di un inirizzo*;
- SOMMA DI PUNTATORI:*Sia* pi *un puntatore, allora* pi + 1 *è il primo indirizzo utile per la memorizzaione di un'altra variabile.*

Vettori VS puntatori

Il primo elemento di un vettore ha la stessa posizione di memoria del vettore stesso: pi=vet sse pi=&vet[0].

- pi+k **punta al k-esimo elemento e** k *è chiamato* **offset**;
- x=pi-qi **in questo caso** x *prende il valore del numero di interi che si trovano tra gli indirizzi delle variabili puntate da pi e qi.*

2.9 Tipi user-defined

Tipi semplici

SINTASSI

- **typedef** "tipo esistente" "nuovo tipo". *Quando viene scelta una sottocategoria di un tipo già esistente*;
- **typedef enum** $\{v_1, v_2, ..., v_k\}$"nuovo tipo".

COSA NE FA IL CALCOLATORE

Li ordina, possiamo quindi utilizzare le operazioni aritmetiche.

!!ATTENZIONE!!

Non sappiamo come stampare un tipo definito con la seconda strategia.

Tipi strutturati

SINTASSI

struct "nome struttura" {tipo "nome tipo"}

SÌ

- Una variabile di tipo **struttura** può essere dichiarata alla creazione della struttura stessa;
- Anzichè usare "**struct** pippo" si può rinominare la struttura usando **typedef**;
- Accedere ai vari campi:
  - oggi.giorno =10;
  - (*pd).giorno=10; N.B. L'uso di parentesi è fondamentale;
  - *pd*− >giorno è EQUIVALENTE a (*pd).giorno;

NO

- Un campo della struttura **NON** può essere del tipo della struttura stessa;
- **NON** si possono confrontare due variabili di tipo struttura globalmente;
- La dimensione della struttura **NON** è necessariamente uguale alla somma delle dimensioni dei tipi usati.

2.10 Liste

Eliminare un elemento di una lista

```
{
 prec=*lista;
 corr=prec− >next;
 prec− >next=corr− >next;
 free(corr);
 corr=prec− > next;
 }
```

Inserire un nuovo elemento in una lista

```
{
 prec=*lista;
 corr=prec− >next;
 nuovo= malloc(sizeof(ElementoLista));
 nuovo− >info=x;
```

```
nuovo− >next=corr;
prec− >next=nuovo;
}
```