

Da problema a programma: introduzione al problem- solving in linguaggio C

Capitolo 2: L'essenziale del linguaggio C

G. Cabodi, P. Camurati, P. Pasini, D.
Patti, D. Vendraminetto



Il linguaggio C ... In breve

- ❑ Tipi base e I/O
 - **tipi di dato primitivi (scalari)**, costanti simboliche
 - **operazioni di I/O** (su stdin/stdout e su file testo)
- ❑ Costrutti di controllo
 - **costrutti condizionali e iterativi**
 - **funzioni** e passaggio parametri (by value/reference)
- ❑ Dati aggregati
 - **vettori e matrici** (di interi, float e caratteri)
 - **stringhe** e vettori di stringhe
 - **strutture** (tipi aggregati)



□ Altro

- Espressioni e conversioni (cast) fra tipi numerici
- gestione della linea di comando (argc e argv)
- le direttive al pre-compilatore (#define, #include)
- (il concetto di puntatore)



Tipi e IO

□ tipi di dato primitivi (scalari)

- int, char, float

- costanti:

 - #define N 100

 - const int N=100;

 - sono "quasi" equivalenti

□ operazioni di I/O e file testo

- formattato: (f)printf e (f)scanf (%d, %c, %f, %s)

- righe/stringhe: (f)gets, (f)puts ((f)printf ("%s"))

- caratteri: (f)getc/getchar, (f)putc ((f)printf ("%c"))



Tipi di dato primitivi (scalari)

Tipi (scalari) e limiti di rappresentazione:

- int, char, float
- (*unsigned/signed, short, long, double*)

Costanti:

- letterali:

10 -72 0250x24

3.14159 1.7E+12

'H' ';' '\0' '\n' "Ciao Mondo!\n"

- Simboliche:

```
#define N 100
#define PIGRECO 3.14159
#define cancelletto '#'
```

```
const int N = 100;
const float PIGRECO = 3.14;
const char cancelletto = '#';
```



Definizione/dichiarazione:

```
int numero;  
char a, b, c;  
float num_reale;
```

Espressioni:

```
5 - 10  
3 * 3.14  
a - 10  
b * 3.14  
(x + 5) * (x - y)  
2*(a + b) - (a*a - b*b)  
(2 * PIGRECO * r)
```



Assegnazione:

```
tmp = a;  // assegnazioni standard  
a = b;  
b = a;  
// assegnazioni con cast esplicito
```

Inizializzazione:

```
int a = 1, b = 2;  
// definizione con inizializzazione
```

Cast (conversione di tipo):

```
x = 10.5;  
y = (float)((int)x * (3/2));  // y = 10.0  
w = x * (float)(3/2);  // w = 10.5  
z = x * (float)3/(float)2;
```



Input/output (compresi file)

Apertura/chiusura file:

```
FILE *fp;  
fp=fopen("myfile.txt", "r");  
...  
fclose(fp);
```

Tipi di I/O:

- formattato (*include quasi totalmente gli altri*):
fscanf, fprintf, scanf, printf, (sscanf)
- stringhe:
fgets, fputs, gets, puts
- caratteri:
fgetc, fputc, getchar, putchar



Esempi

```
fscanf(fp, "%s", str1);
```

```
fgets(fp, 50, str2);
```

```
for(i=0;i<RIGHE;i++) {  
    for(j=0;j<COLONNE;j++)  
        fscanf(fp, "%d", &matrice[i][j]);
```

```
for(i=0;i<N_ELEMENTI;i++)  
    fscanf(fp, "%d%c", &el[i].int,  
           &el[i].re, &el[i].car);
```



Test di fine file

Costante EOF (di solito EOF = -1)

- `fscanf(...)==EOF`
se file non finito, `fscanf` ritorna quanti campi % ha letto correttamente
- `getc/fgetc(...)==EOF`
se file non finito, viene ritornato (come int) il codice (ASCII) di un carattere

`fgets(...)==NULL`

- se file non finito, `fgets` ritorna puntatore a stringa (destinazione dell'input)

Funzione `feof()`: es. `if (eof(fp)), while (!feof(fp))`

- ATTENZIONE: `feof()` vero solo DOPO aver tentato di leggere oltre end-of-file !!! (spesso si fa una lettura in più)



Costrutti di controllo

□ **costrutti condizionali**

- if (con o senza parte else)
- switch-case
selettore multiplo, ma solo per valori interi o char

□ **costrutti iterativi**

- while, do-while
consigliati per condizioni di controllo generali (espressioni logiche)
- for
consigliato per iterazioni numerate (con conteggio)



Costrutti condizionali

Espressioni logiche:

- condizioni per abilitare/disabilitare o selezionare il blocco da eseguire

`if`

```
if (condizione) {  
    // istruzioni caso VERO  
}  
else {  
    // istruzioni caso FALSO  
}
```

Condizioni multiple e costrutti `if` annidati



Switch: selezione multipla

```
switch (selettore) {  
    case 0: printf("Scelta n. 0\n"); break;  
    case 1: printf("Scelta n. 1\n"); break;  
    ...  
    default: printf("Scelta non valida\n");}  
}
```



Costrutti iterativi

while

```
while (continua) {  
    ...  
}
```

do ... while

```
do {  
    printf("scrivi numero positivo");  
    scanf("%d", &x);  
} while (x<=0);
```

for

```
for(i = 0; i < 10; i++) {  
    printf("Inserisci intero: ");  
    scanf("%d", &vet[i]);  
}
```



Funzioni

Funzione come sotto-programma

- scritto una sola volta, riutilizzato più volte
 - contenuto: elaborazioni su parametri e variabili locali, risultato con **return**
- interfaccia
 - prototipo, chiamate
 - Passaggio di parametri
 - **by value**
 - **by reference** (in C NO: si realizza, in pratica, con *puntatore by value*)



Tipi aggregati

□ **vettori e matrici**

■ Aggregati omogenei con indici

- `int v[100]; float M[10][10];`
- `X = V[i]*M[j][k];`
- Dimensioni NOTE (costanti) -> spesso sovradimensionati e sotto-utilizzati.

□ **stringhe**

- vettori di caratteri “speciali”
- manipolate mediante funzioni di libreria (`strlen`, `strcmp`, `strcpy`, ...)
- Purchè terminati con `'\0'` (terminatore di stringa)



Vettori (monodimensionali)

Dati AGGREGATI, accesso mediante INDICI

```
int eta[20], altezza[20], i;  
float etaMedia = 0.0;  
i = 0;  
for(i=0; i<20; i++) {  
    scanf("%d %d", &eta[i], &altezza[i]);  
    etaMedia += eta[i];  
}  
etaMedia = etaMedia/20;
```



Matrici (multidimensionali)

```
int matrice_diagonale[3][3] = { { 1, 0, 0 },  
                                { 0, 1, 0 },  
                                { 0, 0, 1 } } ;  
  
float M2 [N][M], V[N], Y[M];  
for (r=0; r<N; r++) {  
    Y[r] = 0.0;  
    for (c=0; c<M; c++)  
        Y[r] = Y[r] + M2[r][c]*V[c];  
}
```



Stringhe

NON sono un tipo nuovo, ma:

- vettori di caratteri
- terminati da '\0'

Possono essere gestiti:

- carattere per carattere (come vettori)
- come dati unitari, mediante:
 - funzioni di I/O per stringhe: es. fgets, sscanf, fscanf/fprintf (con formato %s)
 - con funzioni di libreria (includendo <string.h>): es. strcmp, strlen, strcpy, strcat, ...



Vettori adimensionati come parametri

Come parametro formale è lecito omettere la dimensione di un vettore:

```
int confronta (char s1[], char s2[]);
```

La dimensione è nota al programma chiamante

```
char a[20], b[20];
```

...

```
if (confronta(a,b)==0) ...
```

Se necessario, parametro aggiuntivo con dimensione (o parte usata del vettore):

```
int ordina(float V[], int n);
```



Argomenti al main

Due parametri al main:

- argv (vettore di stringhe): argv[0] sempre presente, rappresenta il nome del file eseguibile (del programma)
- argc: dimensione del vettore di stringhe (quanti sono gli argomenti)

Esempio:

```
int main (int argc, char *argv[]) {  
    FILE *fp1, *fp2;  
    if (argc<3) {  
        printf("ERRORE: mancano argomenti al main\n");  
        return 1;  
    }  
    fp1 = fopen(argv[1], "r");  
    ...  
}
```



Struct

❑ **Strutture** (tipo struct)

- tipo di dato aggregato
- campo riferito mediante nome
- differenze rispetto a vettori

❑ Es.

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```



Aggregato di dati eterogenei (struct)

Più informazioni eterogenee possono essere unite come parti (campi) di uno stesso dato dato (aggregato)

cognome: Rossi	
nome: Mario	
matricola: 123456	media: 27.25



```
struct studente
```

```
{
```

```
    char cognome[MAX], nome[MAX];
```

```
    int matricola;
```

```
    float media;
```

```
};
```

Nuovo tipo di
dato

- Il nuovo tipo definito è **struct studente**
- La parola chiave **struct** è obbligatoria

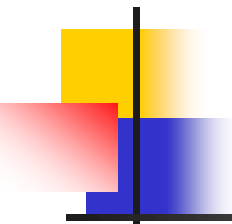


I tipi `struct`

Il dato aggregato in C è detto `struct`. In altri linguaggi si parla di `record`

Una `struct` (struttura) è un dato costituito da campi:

- i campi sono di tipi (base) noti (eventualmente altre `struct`)
- ogni campo all'interno di una `struct` è accessibile mediante un identificatore (anziché un indice, come nei vettori)



```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

Nuovo tipo di
dato

Nome del tipo
aggregato

- Stesse regole che valgono per i nomi delle variabili
- I nomi di **struct** devono essere diversi da nomi di altre struct (possono essere uguali a nomi di variabili)



```
struct studente
```

```
{
```

```
    char cognome[MAX], nome[MAX];
```

```
    int matricola;
```

```
    float media;
```

```
};
```

Campi
(eterogenei)

Nuovo tipo di
dato

Nome del tipo
aggregato

- I campi corrispondono a variabili locali di una struct
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)



Schemi di dichiarazione

1. Schema base

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente s, t;
```



Schemi di dichiarazione

2. Dichiarazione/definizione contestuale di tipo struct e variabili

- tipo struct e variabili vanno definiti nello stesso contesto (globale o locale)

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} s, t;
```



Schemi di dichiarazione

3. (uso raro) Dichiarazione/definizione contestuale di tipo `struct` (senza identificatore) e variabili

- il tipo `struct` viene utilizzato unicamente per le variabili definite contestualmente
- NON si possono definire variabili dello stesso tipo in altre istruzioni dichiarative o in funzioni

```
struct  
{  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
} s, t;
```



Schemi di dichiarazione

4. Sinonimo di struct studente introdotto mediante typedef

```
typedef struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} studente;
...
studente s, t;
```

Schemi di dichiarazione

5. Sinonimo introdotto mediante typedef: variante senza identificatore di struct

```
typedef struct studente
{
    char cognome[MAX]; nome[MAX];
    int matricola;
    float media;
} studente;
...
Studente s, t;
```

Identificatore inutilizzato



Schemi di dichiarazione

5. Sinonimo introdotto mediante typedef: variante senza identificatore di struct

```
typedef struct
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} studente;
...
studente s, t;
```



Dal C ai programmi

- Costrutti e regole del linguaggio
 - Dati per scontati !!! (quasi)
- Programmare = "dal problema alla soluzione" (utilizzando il linguaggio C)
 - Strategia -> problem solving
 - Esperienza e abilità personale
 - Imparare da soluzioni proposte
 - NOVITA': **Classificazione di problemi**



Classi di problemi

Senza vettori

Con vettori/matrici

Numerici

Equaz. 2° grado
Serie e successioni numeriche
...

Statistiche per gruppi
Operazioni su insiemi di numeri
Generazione numeri primi
Somme/prodotti matriciali

Codifica

Conversioni di base (es. binario/decimale)
Crittografia di testo
...

Conversioni tra basi numeriche
Ricodifica testi utilizzando tabelle di conversione

Elab. Testi

Manipolazione stringhe
Menu con scelta
Grafico di funzione (asse X verticale)

Conteggio caratteri in testo
Grafico funzione (asse X orizzontale)
Formattazione testo (centrare, eliminare spazi)

Verifica/ selezione

Verifica di ordinamento/congruenza di dati
Verifica mosse di un gioco
Filtro su elenco di dati
Ricerca massimo o minimo
Ordinamento parziale

Verifica di unicità (o ripetizione) di dati
Selezione di dati in base a criterio di accettazione
Ricerca di dato in tabella (in base a nome/stringa)
Ordinamento per selezione



Esempio: Gomoku

(esame informatica 20/1/2014)

Gioco tradizionale giapponese di allineamento.

- ❑ Due giocatori posano alternativamente pietre (bianche/nere) su una scacchiera 19x19.
- ❑ Vince il primo che riesce a mettere 5 pietre del suo colore in fila (riga, colonna o diagonale).

Realizzare programma C che permetta di

- ❑ riprendere una partita a gomoku precedentemente salvata (su file, nome argomento al main)
- ❑ portarla a termine, acquisendo e controllando le mosse dei due giocatori.



Esempio: Gomoku

(esame informatica 20/1/2014)

Le mosse sono acquisite

- ricevendo da tastiera le coordinate della casella su cui posare la propria pietra
- controllando se la mossa porta alla vittoria.
- Il programma deve inoltre verificare che la mossa sia corretta: coordinate fra 1 e 19 e casella libera.

Il file che contiene la situazione del gioco è composto da

- 19 righe di 19 caratteri in cui
 - il punto “.” rappresenta una casella ancora libera
 - la “B” una pietra bianca, e la “N” una pietra nera.



Analisi

- Problema di verifica:
 - ◆ Per ogni mossa, controllare che sia legale e controllare l'eventuale vittoria
 - ◆ quantificatore esistenziale: vittoria se esiste almeno una sequenza di 5 caselle del colore in riga, colonna, diagonale o anti-diagonale
- A questo problema principale si sovrappongono problemi minori
 - ◆ Input da file, Gestione di matrice con controlli su righe, colonne e diagonali, Modularità con funzioni



Soluzioni proposte

- ❑ Versione base: un main che fa tutto (molte parti ripetute). Problema principale: come si fanno a controllare righe, colonne e DIAGONALI !
- ❑ Versione migliorata, con funzioni per acquisizione mossa, controllo vittoria e stampa matrice (le funzioni evitano ripetizioni)
- ❑ Ulteriore versione, ancora piu' compatta: evita 4 controlli distinti ed espliciti (riga, colonna, diagonale, anti-diagonale), sfruttando una opportuna struttura dati e un costrutto iterativo.
- ❑ Variante ulteriore, con soluzioni alternative e aggiunte non richieste dal testo.



Conclusione

- A partire da una soluzione iniziale, corretta (!), sono possibili molte migliorie.
 - ◆ In questo caso si tratta prevalentemente di programma più compatto, scritto meglio e più semplice da gestire/migliorare
 - ◆ In questo caso NON si affronta l'efficienza (o la complessità) della soluzione.
- L'obiettivo è illustrare più modi di affrontare un problema