Capitolo B71:
linguaggio di programmazione mC [2]

Contenuti delle sezioni

12 pagine

B71:**0.01** In questo capitolo si prosegue nella introduzione del linguaggio mC.

Si inizia introducendo i numeri che vengono chiamati numeri reali ma in realtà costituiscono insiemi finiti di $\mathbb{Q}$ e noi chiameremo "numeri real".
Si espongono poi le espressioni che conducono a questi numeri real.
Si allarga la gamma degli oggetti matematici controllabili con il linguaggio trattando arrays multidimensionali e geometria cartesiana discreta.

Una seconda parte riguarda l'introduzione dei sottoprogrammi e la organizzazione modulare dei programmi.
Si svolgono in nparticolare varie considerazioni sui sottoprogrammi per l'entrata e l'uscita delle informazioni.

L'ultima parte consiste nella discussione di vari esempi di programmi e di elaborazioni che si considerano di rilevante importanza propedeutica.

B71:a. numeri ed espressioni reali

B71:**a.01** Importanti tipi di dati sono quelli che forniscono una rappresentazione in genere buona dei numeri reali e che qui chiameremo tipi dei **numeri real**.

In effetti l'insieme dei numeri real rappresenta fedelmente un certo insieme di numeri razionali la cui precisazione conviene far partire dai literals che consentono di esprimerli nel testo del programma e dalle configurazioni di bits che costituiscono le loro realizzazioni nelle celle-m e che rendono tali quantità concretamente trattabili.

Un numero real positivo può essere espresso con la notazione che presenta la parte intera, il separatore "." e la parte decimale.
Esempi `15.085 , 0.000253 . 250000000`
Questa notazione permette di trattare agevolmente numeri il cui ordine di grandezza va dal miliardesimo al miliardo. In molti calcoli scientifici servono numeri ben inferiori al miliardesimo e ben superiori al miliardo. Per questi si adotta la notazione esponenziale

B71:b. arrays multidimensionali e geometria cartesiana discreta

B71:**b.01**

B71:**b.11** Aggiungiamo alcune prestazioni di questi elementi.

Si possono usare anche frasi esecutive della forma

espressione **;**

L'effeto di questa frase è la valutazione dell'espressione, processo che può comportare i cosiddetti **effetti collaterali** (*side effects*), cioè modifiche di variabili anche importanti; il risultato della valutazione viene invece trascurato.

In particolare si possono avere richiami di function seguiti dal solo "**;**"; ancora il valore assegnato al richiamo viene trascurato e gli effetti del richiamo possono essere importanti. Per esempio si possono avere manovre sopra unità periferiche (riavvolgimenti di nastri, aperture, chiusure, ...), oppure manovre sulla memoria disponibile su heap, invio di messaggi o segnali, manovre sul Web e altro.

Inoltre si possono organizzare functions senza risultati di ritorno, le cosiddette **non-value returning functions**.

Va anche raccomandato che l'effetto del richiamo sia ben chiarito, per evitare che tale controllo rimanga oscuro al programmatore.

B71:c. arrays multidimensionali

B71:**c.01** Per rappresentare una grande varietà di situazioni servono gli **arrays a due indici**, complessi di dati che consentono di implementare le matrici, strutture di dati con una gran quantità di applicazioni: un esempio
Nel linguaggio C gli arrays a due indici si possono considerare arrays a un indice le cui componenti sono a loro volta degli arrays a un indice, componenti tutti costituiti da uno stesso numero di componenti elementari.
Volendo disporre nella sottomatrice 4×4 di una matrice 10×10 dei primi coefficienti binomiali simmetrici [...] si possono utilizzare le seguenti frasi (la prima dichiarativa le successive di assegnazione)

```
int cbs[10][10];
cbs[0][0]=1; cbs[0]1[1]=1; cbs[0][2]=1; cbs[0][3]=1;
cbs[1][0]=1; cbs[1][1]=2; cbs[1][2]=3; cbs[1][3]=4;
cbs[2][0]=1; cbs[2][1]=3; cbs[2][2]=6; cbs[2][3]=10;
cbs[3][0]=1; cbs[3][1]=4; cbs[3][2]=10; cbs[3][3]=20;
```

Se in un programma è sufficiente disporre di una matrice 4×4, basta la seguente frase di inizializzazione

```
int cbs[4][4] = {
                    {1, 1, 1, 1}
                    {1, 2, 3, 4}
                    {1, 3, 6, 10}
                    {1, 4, 10, 20}
                }
```

A questo arrays si dedicano 16 celle per interi normali. In questa sequenza di celle di memoria si devono distinguere 4 sottosequenze, ciascuna costituita da 4 celle: la prima sottosequenza è dedicata alla prima riga della matrice, accessibile per intero scrivendo `cbs`[0], la seconda alla seconda riga accessibile mediante la scrittura `cbs`[1] e così via.

B71:**c.02** Talora servono arrays con 3 o più indici: con arrays a 3 indici si possono trattare grandezze fisiche attribuibili a una griglia spaziale, ad esempio le temperature all'interno di una caldaia a forma di cuboide. L'organizzazione in memoria di tali complessi di dati viene effettuata generalizzando il procedimento precedente.
Un array a tre dimensioni potrebbe essere introdotto con una dichiarazione come la

```
int valspaz[5][3][7];
```

la quale riguarda un array con 5 componenti ciascuna delle quali è un array bidimensionale di profilo 3×7.

B71:d. sottoprogrammi e organizzazione modulare delle procedure

B71:**d.01** Per affrontare problemi di interesse pratico si devono scrivere programmi di molte migliaia e anche di molti milioni di istruzioni. In effetti lo sviluppo delle apparecchiature elettroniche programmabili iniziato a metà del secolo scorso ha portato alla messa a punto di procedure molto elaborate e la tecnologia della programmazione costituisce da decenni uno dei fattori chiave per lo sviluppo dell'economia.
Le attività concernenti la produzione di programmi sono ormai sviluppate con criteri industriali e in esse si rende necessaria una sistematica suddivisione dei compiti; questa suddivisione riguarda sia le azioni svolte dai programmi, sia le attività delle persone che i programmi progettano, redigono, collaudano, fanno evolvere e analizzano criticamente.

Per la suddivisione delle azioni controllate dai programmi, quali che siano i linguaggi di programmazione utilizzati, assume grande importanza la organizzazione dei sottoprogrammi.
Un sottoprogramma si può definire come una porzione di un programma chiaramente, innanzi tutto a livello della sintassi, che al livello semantico, del significato operativo, ha il compito di effettuare un servizio specifico in risposta di ciascuna richiesta che gli viene rivolta.
Ciascuna richiesta a un sottoprogramma ***S***, in linea di massima, contiene specificazioni sulle particolari prestazioni che gli vengono richieste.

Le prestazioni richieste a un sottoprogramma devono essere adeguate alla portata operativa consentita al sottoprogramma stesso. Tuttavia i sottoprogrammi in generale, in accordo con la portata complessiva della programmazione e con l'efficienza e la versatilità delle risorse oggi disponibili, sono in grado di effettuare una vastissima varietà di servizi. Ed in effetti tutti i programmi che risolvono problemi applicativi di interesse concreto demandano la gran parte delle loro prestazioni a sottoprogrammi. Un programma un poco elaborato negli obiettivi e nelle prestazioni può richiedere l'intervento di molte migliaia di sottoprogrammi.

B71:**d.02** A questo punto può essere utile fare riferimento ad alcuni esempi.
In un programma di calcolo tecnico-scientifico sono necessari vari sottoprogrammi incaricati di effettuare calcoli specifici: calcoli di funzioni matematiche che avremo modo di incontrare più volte (radici quadrate, funzioni esponenziali e trigonometriche, altre funzioni speciali, trasformazioni di coordinate, conversioni, ...); soluzioni di sistemi di equazioni lineari e loro varianti; determinazione di zeri di polinomi e di altre funzioni di variabile reale; elaborazioni geometriche, meccaniche, statistiche; manovre su strutture combinatorie e in particolare su grafi variamente arricchiti (per esempio per rappresentare reti di trasporto e automi), ... Di questo genere sono i sottoprogrammi che incontreremo più spesso nel seguito.

Delle operazioni di ingresso e uscita si incaricano numerosi sottoprogrammi. Molti di questi si incaricano del controllo dei dispositivi hardware delle molteplici apparecchiature che si possono connettere a una macchina programmabile. Oggi si possono considerare periferiche tradizionali i terminali video con tastiera e mouse, le stampanti, i plotters, i dischi, i nastri, le memorie flash. A queste in tempi più recenti si sono aggiunti le apparecchiature per l'immissione e l'emissione dei suoni, le macchine fotografiche, le videocamere, monitors televisivi e i collegamenti con reti della telefonia e reti di computers. Tutti questi sottoprogrammi agenti sull'hardware sono evidentemente legati a molteplici tecnicismi e possono richiedere linguaggi specifici.

Venendo a prestazioni più vicine alle applicazioni e formulabili con linguaggi come miniC, per le operazioni di ingresso dei dati ricordiamo: lettura di stringhe e loro codifica in valori interni, tipicamente

lettura di scritture che seguono notazioni standard di numeri interi o razionali; lettura di denominazioni leggibili e loro trasformazioni in codifiche convenzionali che rendono più agevole la determinazione di loro caratteristiche (ad esempio le codifiche con una lettera e 3 cifre dei comuni italiani).
Ancora più variegate sono le prestazioni di scrittura: operazioni di decodifica in chiari speculari delle accennate operazioni di codifica; presentazione di grafici che raffigurano dati numerici (istogrammi, areogrammi, ...); arricchimento dei risultati con dati di sintesi (medie, varianze, totali parziali, ...); generazione di animazioni.

Assimilabili ai sottoprogrammi di entrata e uscita sono i sottoprogrammi che si incaricano di operazioni di inserimento e di reperimento di dati entro strutture informative e archivi.
Molti altri sottoprogrammi si possono incaricare di conversioni e di transcodifiche relative a formati e a strutture di dati definiti per rendere calcolabili determinati modelli e per trattare più agevolmente particolari manovre; in particolare vi sono programmi che organizzano le "visite" di strutture informative per analizzarle o individuarne elementi di rilievo (minimi, massimi, estremali, ottimali).
Per concludere, non possiamo non ricordare i sottoprogrammi che si occupano della gestione di situazioni erronee o anomale, operazioni che nei più recenti linguaggi di programmazione hanno ottenuto specifici riconoscimenti anche al livello della sintassi.

B71:**d.03** Nel linguggio miniC i sottoprogrammi assumono la forma di quelle che chiameremo **functions**, entità con alcune caratteristiche delle funzioni della matematica, ma per le quali useremo il termine inglese per sottolinearne le differenze.
Una function è una sezione sintatticamente ben definita del testo sorgente di un programma che può essere richiamata per eseguire manovre specifiche che è opportuno siano ben definite. La forma di una function prevede una intestazione e un corpo e si può schematizzare come segue.

tipo-della-function *identificatore-della-function* (*lista-degli-argomenti*)
`{`
corpo-della-function
`}`

Ogni esecuzione di una function produce un risultato primario che viene associato al suo richiamo ed è disponibile nell'espressione che contiene il richiamo stesso. Incontreremo prevalentemente functions che producono un risultato primario intero. Altre produrranno invece risultati dei tipi `double`, `char` e indirizzi di dati dei vari tipi. Inoltre alcune functions non producono risultati e vanno attribuite allo specifico tipo `void`.
L'identificatore di una function soddisfa le stesse regole lessicali degli identificatori delle variabili e come per questi si pone il problema della non omonimia. È opportuno che l'identificatore di ogni function sia scelto in modo da evidenziare ; questa scelta, in genere piuttosto critica, va fatta cercando di avere chiari i suoi possibili utilizzi e quali altre function saranno utilizzate in collegamento o in alternativa a essa.

La lista degli argomenti può contenere uno o più argomenti o anche nessun argomento. Di ogni argomento si deve dare il tipo e l'identificatore eventualmente accompagnato da un modificatore.
Il corpo di una function si presenta simile a un blocco di istruzioni. In genere inizia con dichiarazioni di variabili locali la cui visibilità è limitata al corpo della function. Successivamente può presentare una qualsiasi composizione di sequenze di comandi e di blocchi di selezioni e di iterazioni. Vogliamo che queste composizioni siano ben strutturate.
Nel corpo di una function deve comparire almeno una istruzione `return` caratterizzata da un argomento costituito da un'espressione valutabile in un risultato avente lo stesso tipo della function; l'esecuzione

di un tale return fornisce come risultato primario di ogni richiamo della function il valore attuale dell'espressione.

B71:**d.04** Ogni richiamo alla function è costituito dal suo identificatore seguito da una sequenza di argomenti attuali, espressioni costituite da operandi valutabili all'atto dell'esecuzione del richiamo le quali dovranno essere in grado di fornire un valore per ciascuno degli argomenti formali della intestazione della function richiamata.

Una function per essere utilizzabile deve essere anche dichiarata. Una dichiarazione di una function può essere una semplificazione della sua intestazione nella quale ogni argomento viene sostituito dalla semplice dichiarazione del suo tipo. Questo enunciato semplifica il lavoro al traduttore del testo sorgente del programma nel suo equivalente utilizzabile per l'esecuzione (dopo la manovra di linkaggio che vedremo più oltre) in quanto annuncia gli elementi essenziali, cioè i tipi, dei suoi argomenti e del suo risultato primario. Alla dichiarazione devono adeguarsi l'intestazione della function e tutti i suoi richiami.
In una dichiarazione di function la specificazione di un argomento può presentare, oltre al suo tipo, un identificatore fittizio che conviene scegliere in modo di fornire un'indicazione del ruolo dell'argomento stesso, ossia del suo significato operativo. In tal modo una dichiarazione di function può documentare in forma concisa il suo significato.

Anche il modulo principale di ogni programma viene strutturato come una function. Tuttavia l'identificatore di tale function è prefissato, deve essere `main`. Anche il modulo `main` può avere un tipo, ma spesso si riduce a `void`. La forma dell'intestazione di un modulo principale è la seguente:

tipo `main(int argc, char **argv)`

Si prevede che la richiesta dell'esecuzione di un programma sia ottenuta digitando il nome del file contenente il testo sorgente eventualmente seguito da un determinato numero di stringhe che possono influenzare la sua esecuzione. Il numero di tali stringhe è fornito dalla variabile identificata da `argc` (sta per "argument count") ed esse sono reperibili in sequenze di bytes le cui `argc` posizioni iniziali sono ottenibili dall'array il cui identificatore è `argv` (sta per "argument values").

B71:**d.05** Riprendiamo gli effetti che può avere un richiamo di una function. Per questo occorre distinguere tra gli argomenti della function quali vengono richiamati per valore e quali per indirizzo. All'inizio dell'esecuzione di un richiamo di function ciascuno degli argomenti fornisce un valore che viene fornito alla function. Ciascuno degli argomenti costituiti da espressioni da valutare o da variabili fornisce un valore che viene riprodotto nella corrispondente variabile fittizia. Similmente ciascuno degli argomenti che individuano un indirizzo porta alla riproduzione di tale indirizzo in una variabile interna alla function.
La riproduzione di un argomento (richiamato per valore) costituito da una variabile assicura che la function non sia in grado di modificare il valore di questa variabile nel modulo richiamante. Se questo accadesse si avrebbe una modifica nel modulo richiamante non ricavabile dal suo testo e quindi poco controllabile.
La riproduzione degli indirizzi lascia aperta la possibilità di modificare i contenuti delle celle di memoria associate agli indirizzi ed accessibili sia al modulo richiamante che alla function. Queste celle devono essere utilizzate come indirizzi e quindi le modifiche che la function può effettuare sono maggiormente evidenti e controllabili dai programmatori del modulo richiamante e della function delle modifiche che si potrebbero effettuare all'interno di una function che potesse agire direttamente sulle celle nelle quali sono registrati i valori degli argomenti.

Tutta questa organizzazione è un poco elaborata, ma ha lo scopo di rendere meno probabili le modifiche di dati del modulo chiamante da parte di istruzioni formulate nel testo sorgente della function.
La function dopo la riproduzione degli argomenti, procede a eseguire le manovre formulate nel blocco costituente il corpo della function. Queste manovre potrebbero essere molto complesse, anche perché molte functions che prestano servizi complessi in genere richiamano altre functions.
Le manovre implicate da un richiamo di function forniscono il servizio che costituisce la ragion d'essere della function stessa. Una function può servirsi delle componenti di arrays e può modificarle, ma solo attraverso gli indirizzi iniziali di questi complessi di dati.

B71:**d.06** Per servirsi correttamente di una function è necessario che i suoi effetti sui suoi argomenti siano definiti con precisione e completezza. Viceversa non è necessario che siano specificati i dettagli delle operazioni con le quali gli effetti vengono ottenuti. Anzi, in linea di massima è opportuno che questi effetti siano documentati solo nel testo sorgente della function o in un documento a essa associato. Infatti una function è tanto più utile quanto più agevolmente il programmatore responsabile di uno o di alcuni dei moduli che la richiamano riesce a mettere a punto i relativi richiami. Questo programmatore deve potersi documentare in modo completo e preciso, ma essenziale sugli effetti della function e deve poter evitare di addentrarsi nei dettagli delle operazioni che vengono effettuate quando la function viene eseguita.

Si prospetta quindi una divisione dei compiti tra programmatore della function e programmatore delle sue utilizzazioni: per un buon utilizzo delle risorse umane è bene che i dettagli delle operazioni effettuate dalla function rimangano nascosti ai programmatori che primariamente devono solo utilizzarle attraverso i loro richiami.
Questa ripartizione dei compiti si realizza con il cosiddetto **incapsulamento** delle manovre di una function.
Va ribadita l'importanza della documentazione verso l'esterno delle prestazione delle functions: questa deve essere precisa, completa e rapidamente leggibile quando della function si vogliano utilizzare solo prestazioni circoscritte.
Tenuto conto dell'importanza in quanto prodotti industriali delle functions in un linguaggio di programmazione diffuso, dovrebbe essere chiaro che nella pianificazione della messa a punto di un programma si devono dedicare molte attenzioni alla definizione e alla scelta di una cosiddetta library delle functions utilizzabili, alla documentazione delle loro prestazioni e alle prove che possono garantire la loro correttezza e la loro adeguatezza.

B71:**d.07** Occorre segnalare che per sviluppare programmi applicativi incisivi in genere servono ampie librerie di functions e che è di grande importanza la facilità del loro riutilizzo per applicazioni diverse da quelle che hanno condotto alla loro prima stesura.

La possibilità di disporre di ampie librerie come accade per i sistemi di sviluppo di linguaggi con una certa tradizione e di ampia diffusione costituisce un elemento di grande importanza per le attività di sviluppo del software.
La disponibilità di librerie di sottoprogrammi versatili e affidabili si può considerare come un importante arricchimento delle prestazioni del linguaggio stesso.
Per un buon uso di queste librerie di programmi è opportuno disporre anche di strumenti che consentano di reperire efficacemente le functions che possono servire. Attualmente sono disponibili vari sistemi di sviluppo per i linguaggi di programmazione più diffusi che posseggono strumenti efficaci e versatili per la gestione delle librerie. Per lo sviluppo dei programmi da alcuni anni si possono consultare efficacemente anche vari siti Web, in particolare siti attraverso i quali si possono ottenere suggerimenti da intere comunità di programmatori.

Queste considerazioni vogliono solo inquadrare il problema della gestione dei sottoprogrammi e le problematiche della programmazione modulare. Chi intende occuparsi professionalmente di questi problemi dovrà approfondirli adeguatamente. Qui nel seguito concretamente ci interesseranno invece functions di portata limitata, con finalità ben definite che consentano di esprimere in modo preciso e verificabile una gamma di algoritmi che sia significativa per le nozioni matematiche e computazionali che esamineremo.
Va comunque segnalata la vicinanza concettuale tra la disponibilità dei risultati della matematica attraverso terminologie e notazioni ampiamente condivisibili e le linee generali seguite per la organizzazione delle librerie di sottoprogrammi riutilizzabili per lo sviluppo di prodotti software. Questa vicinanza si rileva soprattutto quando si considerino i sottoprogrammi per la soluzione di problemi computazionali come implementazioni di risultati matematici.

B71:**d.08** Gli odierni ambienti per lo sviluppo dei programmi scritti in un linguaggio di programmazione di largo uso mettono a disposizione ampie librerie di sottoprogrammi predisposti per risolvere problemi che si presume si debbano affrontare spesso. Le librerie di sottoprogrammi possono anche essere messe a punto dai programmatori che affrontano problemi specifici operando singolarmente o in gruppi di lavoro, oppure essere acquisite da fornitori specializzati che rendono disponibili prodotti software di largo interesse o su misura, oppure possono essere reperite nei siti del Web finalizzati alla messa a disposizione di software libero.

Molte librerie di functions vengono rese disponibili mediante comandi `#include`; si tratta dei cosiddetti **comandi per il preprocessore**, frasi che occupano una linea del sorgente e iniziano con il carattere `#`. Questi comandi vanno posti nel sorgente prima dell'enunciato di inizio della function `main`.
Tra le librerie di base segnaliamo la `iostream`, la tt conio e la `stdio` riguardanti prestazioni di ingresso e uscita e la `math` che raccoglie functions dedicate a operazioni matematiche.

B71:**d.09** Presentiamo alcune routines per la manipolazione di stringhe.
Preliminarmente va detto che in C/C++ vengono trattate facilmente le stringhe ASCII organizzate in arrays del tipo `char` monodimensionali che fanno seguire i successivi da un carattere `null` che segnala la conclusione della stringa stessa. Una stringa di n caratteri deve avere a disposizione un array di almeno $n + 1$ bytes.

`strcat(str1,str2)` Concatena, ossia giustappone, due stringhe.

`strcmp(str1,str2)` Confronta le due stringhe argomento e fornisce 1 sse coincidono, 0 in caso contrario.

`strcmpi(str1,str2)` Confronta le due stringhe argomento e fornisce 1 sse coincidono oppure presentano differenze maiuscola/minuscola, 0 in caso contrario.

`strcpy(str1,str2)` Riproduce la stringa assegnata a `str2` nell'array `str1`.

`strstr(str1,str2)` Scandisce la stringa in `str1` alla ricerca della prima occorrenza come sua sottostringa della stringa in `str2`.

`strlen(str1)` Fornisce l'intero esprimente la lunghezza della stringa argomento.

`strupr(str1,str2)` Pone in `str1` la stringa fornita da `str2` dopo aver modificato ogni eventuale carattere minuscolo nel corrispondente minuscolo.

`sprintf()` Costruisce una stringa a partire da dati in numero variabile, cioè che in diverse frasi di richiamo possono presentare argomenti in diversi numeri.

B71:**d.10** Tra le functions, e in particolare tra le precedenti, occorre distinguere tra quelle che non presentano e quelle che presentano i cosiddetti **effetti collaterali**. L'effetto di un richiamo di una function

del primo tipo consiste solo nella fornitura di un valore del tipo assegnato alla function nella sua dichiarazione. Viceversa una function presenta effetti collaterali se un suo richiamo comporta modifiche alle variabili e agli arrays che del richiamo sono gli argomenti.

Tra le functions del paragrafo precedente non presentano effetti collaterali `strcmp`, `strcmpi`, `strstr` e `strlen`.
Ne comportano invece `strcat`, `strcpy`, `strupr` e `sprintf`.

B71:e. sottoprogrammi per l'entrata e l'uscita

B71:**e.01** Le operazioni di entrata e uscita si possono programmare servendosi di diverse modalità, ovvero utilizzando gruppi di sottoprogrammi facenti parte di diverse librerie. Tra queste si deve distinguere, innanzi tutto, tra quelle introdotte con il linguaggio C (tendenzialmente più elementari) e quelle adottate con il linguaggio C++ (che in genere sono di uso più impegnativo, ma sono più complete e danno maggiori garanzie di sicurezza).

Per usare questi sottoprogrammi si deve provvedere all'inserimento dei comandi iniziali per la predisposizione degli header della forma

`#include <`*specifica di header*`>`

L'header per i tradizionali sottoprogrammi di I/O del linguaggio C ha come specifica `stdio.h`.
Nell'ambito dei sistemi di sviluppo per C++ sono disponibili i seguenti headers:

`iostream.h` riguarda sottoprogrammi per l'entrata da e l'uscita su i cosiddetti streams, periferiche o files che gestiscono sequenze di bytes.

`iomanip.h` fornisce prestazioni di manipolazione dei dati;

`fstream.h` riguarda operazioni per l'emissione su e l'immissione da files su memorie di massa.

B71:**e.02** Le più semplici operazioni riguardano l'immissione e l'emissione di singoli caratteri attraverso le functions `getchar` e `putchar`. Vediamo le schema di un semplice programma che si serve di tali functions.

```
#include <stdio.h>
main();
char c,d;
c = getchar();    d = getchar();    // immissione dei due caratteri
if(('a'<=c && c<='Z') || ('a'<=c && c<='z') {// in c lettera
    if('0'<=d && d<='9') {   // in d cifra:  caratteri accettati.
    putchar(c); putchar(' '); putchar('e'); putchar(d);
        putchar(' '); putchar('o'); putchar('k'); putchar(EOF);
        return(1);
        }
    }
// caratteri rifiutati.
putchar('i'); putchar('i'); putchar('n'); putchar('p'); putchar('u');
putchar('t'); putchar(' '); putchar('n'); putchar('o'); putchar(EOF);
return(0);
}
```

Da questo programma si intuisce come mediante la lettura dei singoli caratteri dei dati si possono controllare tutti i loro dettagli. Si vede anche come si possano precisare le emissioni, ma risulta evidente anche la minuziosità di questo modo di organizzare letture e scritture e di conseguenza l'opportunità di disporre di strumenti che consentano controlli più sintetici.

B71:**e.03**

B71:**e.04**

B71:**e.05**

B71:**e.06**

B71:f. esempi di programmi e di elaborazioni

B71:**f.01** Iterazioni annidate e visita di matrici e di griglie tridimensionali

B71:**f.02**

B71:**f.03**

B71:**f.04**

B71:**f.05**

B71:**f.06** Conversioni di rappresentazioni Dati numerici, testi

B71:**f.07** Funzioni date da espressioni aritmetiche a tratti

B71:**f.08**

Testi dell'*esposizione* in http://www.mi.imati.cnr.it/alberto/ e in http://arm.mi.imati.cnr.it/Matexp/