Capitolo B70:
linguaggio di programmazione mC [1]

Contenuti delle sezioni

43 pagine

B70:**0.01** Questo capitolo introduce una prima parte delle prestazioni di un linguaggio di programmazione procedurale che consente di implementare algoritmi in grado di sostenere la comprensione di svariate nozioni della matematica.

Questo linguaggio, a cui diamo il nome **mC**, per mini C, può vedersi come sottoinsieme piuttosto ridotto del `linguaggio C++ (wi)` nella sua versione standard `C++11 (wi)` comprendente buone parte del linguaggio `C (wi)` nella versione standard ANSI.

Esso vuole essere uno strumento di programmazione semplificato ma utilizzabile concretamente con qualcuno dei molti sistemi per lo sviluppo del linguaggio C++ attualmente disponibili.

L'aver preferito C++ al più semplice C è dovuto a vari motivi.

C++ è più sicuro grazie alla più forte tipicizzazione; è dotato di librerie più evolute e ricche; rispetto a C C++ è più vicino ai problemi e meno alle esigenze della macchina, ma consente ancora di occuparsi dei dettagli della implementazione; consente commenti più elastici da collocare a destra dei segni `\\`; facilita il controllo di letture e scritture di caratteri grazie ai più semplici comandi `cin >>` e `cout <<`; agevola il controllo della memoria temporanea grazie agli operatori `new` e `delete`; attraverso il meccanismo delle classi apre la possibilità della programmazione per oggetti, tecnica impegnativa e poco curata da questa presentazione, ma con grandi potenzialità per la implementazione di strutture matematiche, a esempi di figure geometriche.

B70:**0.02** In questo capitolo mC viene introdotto limitandosi alle caratteristiche che consentono di richiedere elaborazioni sopra numeri naturali e stringhe mediante semplici piccoli programmi che non fanno uso di sottoprogrammi, cioè che sono costituiti da un unico modulo. Più avanti verranno introdotte alcune delle molte altre prestazioni di C++, ed in particolare le caratteristiche che consentono di trattare quelli che chiameremo **numeri reali-P**, cioè i numeri costituenti ,,,A70002
della programmazione vengono chiamati "real numbers".

Le pagine che seguono vogliono anche introdurre i primi concetti sulla programmazione e sul computer con considerazioni che cercano di essere coerenti con quelle svolte come prime motivazioni per le attività matematiche.

Con questo intendiamo sottolineare la naturale vicinanza tra le basi discrete della matematica e le basi del trattamento delle informazioni, vicinanza dovuta all'interesse di entrambe verso le attività di calcolo e delle conseguenti prospettive, ampie e ambiziose, riguardanti le possibilità di ottenere soluzioni condivisibili e di portata generale per problemi che si pongono nel mondo odierno.

Una seconda motivazione del capitolo sta nella apertura della possibilità di sperimentare concretamente l'esecuzione di calcoli concernenti costruzioni matematiche riguardanti configurazioni finite.

La possibilità di sperimentare, oltre ad avere evidente valenza pratica, può essere un robusto supporto alla comprensione di molte nozioni matematiche, a cominciare da quelle delle strutture discrete, e di molte delle applicazioni della disciplina.

Dopo questa introduzione, e ancor più dopo i successivi capitoli sul linguaggio di programmazione, si potranno presentare mediante piccoli programmi o mediante frammenti di programma le implementazioni di alcuni dei procedimenti costruttivi che si incontreranno.

B70:a. linguaggio mC per implementare della matematica

B70:**a.01** Prima di introdurre le prime nozioni sul linguaggio mC, riprendiamo le due sostanziali motivazioni della sua ascelta.

Il linguaggio C, predecessore del linguaggio C++, è ampiamente conosciuto e utilizzato e i suoi costrutti sono stati ampiamente ripresi in molti altri linguaggi di programmazione, sia da linguaggi specialistici che che linguaggi a un livello superiore del C. Quindi mC consente di presentare algoritmi che si collocano nel mainstream della programmazione e consente di avvalersi di una vasta letteratura di supporto e approfondimento. Sono numerosi i testi e i siti Web che trattano i linguaggi C e C++, sia a livello introduttivo, sia a più avanzati livelli di critica e di progettazione. Tali pubblicazioni rendono disponibile una vasta gamma di esempi con i quali i contenuti presentati nelle pagine che seguono possono trovare utili varianti, completamenti ed esempi complementari.

La seconda motivazione di mC risiede nella possibilità di rendere i programmi, i frammenti di programma e i sottoprogrammi presentati nelle pagine che seguono concretamente sperimentabili e modificabili attraverso i numerosi compilatori e ambienti di sviluppo per i linguaggi C e C++. Questo sarebbe più problematico se si fosse scelto qualche linguaggio meno praticato e non sarebbe direttamente attuabile se si fosse adottato qualche genere di `pseudocodice (wi)`.

La disponibilità di manuali sui linguaggi C e C++, tra l'altro, ci consente di introdurre, con discorsi concisi che evitano di soffermarsi su molti dettagli, le prestazioni di mC che riteniamo poco cruciali per questa *esposizione*.

B70:**a.02** La scelta di mC, oltre che rendere disponibili algoritmi effettivamente verificabili, intende avviare la costituzione di una libreria di programmi che porti anche a considerare come problema culturale e di ampia influenza quello della disponibilità delle costruzioni procedurali.

Inoltre si vuole la possibilità di sottolineare che i procedimenti costruttivi si riducono a operare su sequenze finite, a partire da quelle su interi e stringhe, e si vuole stimolare l'esame delle differenze tra alcune formule matematiche e le loro possibili implementazioni.

B70:**a.03** È opportuno anche segnalare che le attività di implementazione di algoritmi di interesse matematico conducono naturalmente a porsi una vasta gamma di problemi che riguardano questa disciplina dalla millenaria tradizione.

Un prima tema che emerge dalle implementazioni riguarda le relazioni che intercorrono tra nozioni ampiamente presenti nei discorsi di matematica e nozioni di uso comune nella programmazione e nel trattamento dei dati.
In particolare meritano di essere ben chiarite la relazione tra insieme finito e sequenza, la relazione tra funzione e sottoprogramma, la relazione tra trasformazione lineare e matrice vista anche come array bidimensionale.

Tra i problemi di natura matematica emersi dalle attività di trattamento dei dati possiamo ricordare quelli riguardanti le valutazioni quantitative sopra alcune classiche famiglie di algoritmi (selezione, ordinamento, visita di strutture, ...) e le questioni sulla computabilità (complessità, simulazione, ...) che si trovano in stretta relazione con i fondamenti della matematica.

B70:**a.04** Dovrebbe essere naturale per chi si interessa di matematica osservare come le odierne apparecchiature elettroniche consentano di effettuare una vasta e crescente gamma di calcoli di interesse matematico. Il problema della implementazione della matematica dovrebbe essere considerato di primaria importanza e dovrebbe essere affrontato con prospettive ampie e generali.

Un prima constatazione che conviene avere presente quando si pensa a macchine per elaborazioni automatiche dice che ogni dispositivo fisico, disponendo di risorse di memoria e di tempo finite può trattare solo un numero finito di entità matematiche; va anche osservato che questa finitezza è condizionata anche dal fatto che può fornire solo informazioni che devono essere percepite distintamente sia da esecutori umani che da esecutori meccanici.

B70:b. informazioni booleane e numeri interi

B70:**b.01** Ogni dispositivo per elaborazioni automatiche, e quindi ogni macchina in grado di implementare entità matematiche, deve innanzi tutto essere capace di trattare le entità informative più semplici ed essenziali, le cifre binarie, i bits.
Innanzi tutto questi oggetti sono in grado di rappresentare i due valori di verità, il vero e il falso, in genere associati rispettivamente ai numeri interi 1 e 0. L'essenzialità dei bits è legata al fatto che essi consentono di controllare le scelte operative: un bit consente di scegliere tra due possibili percorsi operativi o conoscitivi. Quindi ogni meccanismo automatico dovendo essere in grado di gestire molte scelte operative, deve essere capace di operare efficientemente sui bits

In effetti le odierne tecnologie forniscono molteplici dispositivi che consentono di trattare i bits per immagazzinarli, trasformarli, ripresentarli, conservarli e trasmetterli a grandi distanze. Questi dispositivi presentano costi complessivi di realizzazione e di esercizio estremamente bassi e possono operare con velocità e tempestività molto elevate; inoltre essi consentono di immagazzinare ed elaborare grandissime quantità di informazioni binarie in spazi e con costi di conservazione molto contenuti.

B70:**b.02** Come già segnalato, i dati più semplici e più fondamentali che si possono trattare sono le cifre binarie o bits.
Un bit consente di trattare i valori di verità true e false, vero e falso; in genere, e in particolare nei sistemi di sviluppo di C e C++, il valore true viene rappresentato con 1 e il false con 0. Sui valori di verità si possono attuare le operazioni booleane concernenti le sentenze, come vedremo in particolare in B60.
Un bit, in quanto rappresentazione di un valore di verità, in date circostanze, ovvero dopo aver fissate opportune convenzioni, fornisce l'informazione che determina una scelta dicotomica, cioè una scelta tra due alternative mutuamente esclusive. Una tale scelta potrebbe precedere la effettuazione o meno di una data azione, oppure l'esecuzione di una di due manovra diverse. Questi ruoli rendono i valori di verità fondamentali per l'organizzazione delle procedure.

Un bit, inoltre, potrebbe essere usato per esprimere la presenza o l'assenza di un particolare oggetto in una data collezione. Per trattare questioni concernenti l'appartenenza o meno ad un insieme finito E presentato con una sequenza, ovvero con una elenco, di n componenti in genere risulta utile servirsi di sequenze di n cifre binarie: infatti ciascuna di tali sequenze consente di individuare un sottoinsieme di E; in effetti di tale insieme costituisce la funzione indicatrice [B13b]. Le sequenze binarie sono quindi assai utili per il controllo delle collezioni di dati (in particolare nell'ambito dei DBMS (we), i sistemi per la gestione delle basidati).

Mediante sequenze di bits si possono rappresentare numeri interi, [B12c. Osserviamo che anche questa prestazione si può ricondurre ad indicazioni di presenze oppure assenze di oggetti: precisamente per esprimere i numeri naturali costituenti l'intervallo $\mathbf{[}0 : 2^h - 1\mathbf{]}$, gli oggetti che possono essere presenti o meno sono le potenze 2^i nelle espressioni numeriche $\sum_{i=0}^{h} b_i\, 2^i$ nelle quali $b_i \in \{0, 1\}$.

B70:**b.03** La tecnologia odierna consente di registrare, trasmettere ed elaborare con grande efficienza e velocità grandi moli di cifre binarie. Sono disponibili dischi magnetici e ottici in grado di registrare centinaia e migliaia di miliardi di bits e dispositivi circuitali statici con capacità di memorizzazione non molto inferiori. I circuiti operativi dei processori attuali sono in grado di elaborare informazioni binarie a velocità di miliardi di operazioni al secondo.

Come avremo modo di esemplificare, tutte le informazioni trattabili con le attuali apparecchiature informatiche possono essere rappresentate mediante sequenze binarie. Questo fatto, insieme alle precedenti indicazioni quantitative, rende ben comprensibile la attuale tendenza generale di servirsi di sequenze di bits per la registrazione, la trasmissione e l'elaborazione di tutte le informazioni gestite da automatismi (dati numerici, testi, firme, immagini, suoni, animazioni, programmi, catene dimostrative, ...).
Procediamo ora a illustrare le rappresentazioni binarie dei tipi più semplici di informazioni da trattare mediante automatismi.

B70:**b.04** Per le informazioni delle diverse specie si rendono necessarie unità di registrazione di diverse taglie.
Per quanto riguarda le rappresentazioni di numeri interi, le varie apparecchiature usate, e quindi i vari linguaggi di programmazione, prevedono di trattare diverse collezioni di interi che nelle diverse circostanze possono essere preferibili o in quanto più efficienti o, all'opposto, in quanto di maggiore portata. Prevalentemente si trattano interi rappresentabili con 16, 32, 64 o 128 bits i quali possono essere o meno dotati di segno.

Con unità di registrazione da 16 bits si possono trattare gli interi naturali facenti parte dell'intervallo $\mathbf{[}0 : 65\,535\mathbf{]}$ ($2^{16} = 65\,536$), oppure i numeri interi relativi [B20a, `Complemento a due (wi)`) che costituiscono $\mathbf{[}-32\,768 : 32\,767\mathbf{]}$

Con sequenze di 32 bits si possono trattare gli interi naturali dell'intervallo $\mathbf{[}0 : 4\,294\,967\,295\mathbf{]}$ ($2^{32} = 4\,294\,967\,295$), oppure gli interi relativi appartenenti a $\mathbf{[}-2\,147\,483\,648 : 2\,147\,483\,647\mathbf{]}$.

Con sequenze di 64 bits si possono trattare gli interi naturali di $\mathbf{[}0 : 18\,446\,744\,073\,709\,551\,616\mathbf{]}$ ($2^{64} = 18\,446\,744\,073\,709\,551\,616$), oppure gli interi relativi in $\mathbf{[}-9\,223\,372\,036\,854\,775\,808 : 9\,223\,372\,036\,854\,775\,807\mathbf{]}$.

B70:**b.05** Nel seguito ci occuperemo quasi esclusivamente di personal computers di uso comune in quanto facilmente disponibili per sperimentare, ma gran parte delle considerazioni che seguono possono essere applicate ai molti altri tipi di apparecchiature per la gestione di informazioni digitali (tablets, smart phones, fotocamere, dispositivi telematici, smart TV, rilevatori, servomeccanismi, apparecchiature medicali, ...).

Inizialmente ci occuperemo delle funzioni per la registrazione delle informazioni digitali, ovvero descriveremo i computers (altre apparecchiature digitali) come contenitori di grandi quantità di bits.

I dispositivi per la registrazione delle informazioni binarie, ovvero i **dispositivi di memoria** degli odierni computers, possono rappresentarsi come sequenze molto estese di registri per singoli bits.
I dispositivi, sostanzialmente circuiti elettronici, che consentono di trattare (immettere, trasformare ed emettere) le informazioni digitali per la massima parte riguardano sottosequenze di lunghezza ben definita di registri binari.
Quindi per molte questioni i dispositivi di memoria si possono convenientemente considerare costituiti da sequenze di queste sottosequenze. Queste sottosequenze binarie dal punto di vista dei linguaggi di programmazione le chiameremo **celle indirizzabili di memoria**, in breve celle-m.

Nelle odierne apparecchiature elettroniche vengono trattate soprattutto celle-m riguardanti sottosequenze di 8, 16, 32, 64 e 128 bits. Le celle-m di s registri binari le diremo in breve celle di s bits o anche celle-sb.

Chiameremo **bytes** o ottetti le celle-m di 8 bits.), **halfwords** o semiparole le celle di 16 bits, **words** o parole quelle di 32 bits, **doublewords** o **doppie parole** quelle di 64 bits e **quadwords** , le sottosequenze di 128 registri binari.

Per molte considerazioni le posizioni binarie di ciascuna delle celle-sb conviene raffigurarle come caselle quadrate disposte orizzontalmente e caratterizzate con gli interi 0, 1, 2, ..., $s-1$ crescenti quando si procede da sinistra verso destra. Queste caselle hanno il ruolo delle celle binarie, ovvero sono destinate a contenere valori binari che possono variare nel tempo e gli interi che le caratterizzano si possono considerare i rispettivi indirizzi interni.

Abbiamo quindi raffigurazioni come le seguenti:

//input pB70b05

B70:**b.06** Ciascuna delle celle di memoria delle quali può disporre un programma per il computer può essere raggiunta attraverso il suo **indirizzo**, l'intero naturale che rappresenta la sua posizione nella sequenza che costituisce memoria centrale del computer stesso.

Anche gli indirizzi delle celle-m sono gestiti attraverso celle di memoria e l'estensione di queste ultime dipende, per evidenti motivi, dalla ampiezza della memoria disponibile. Non entreremo nei dettagli del trattamento degli indirizzi che viene effettuato con tecniche anche complesso le quali cambiano da macchina a macchina. Accade però che chi programma in un linguaggio di livello medio o alto, come con C re C++, vede il problema della gestione delle memorie in modo semplificato e dipendente dal computer in uso solo dalla sua capacità complessiva. Su questo problema ritorneremo in seguito in relazione alle caratteristiche comuni ai diversi sistemi di sviluppo per C e C++.

Per ora ci basta rilevare che tra i diversi ruoli delle sequenze di bits in memoria vi è anche quello di individuare le posizioni di altre celle-m.

B70:**b.07** Coerentemente con quanto visto per le celle binarie di una cella-m, conviene raffigurare le sequenze di celle-m con rettangoli disposti orizzontalmente caratterizzati da indirizzi che crescono procedendo da sinistra verso destra. Si può fotografare il contenuto di una sequenza di celle-m indicando in questi rettangoli i valori dei rispettivi contenuti correnti. Il valore variabile nel tempo di una cella-m può essere rappresentato diversamente a seconda del ruolo ovvero dell'utilizzo al quale è destinato o che si vuole evidenziare nella presentazione della memoria in esame.
Torneremo sull'argomento parlando di variabili e operazioni relative richieste nel linguaggio.

Occorre precisare che vanno distinti gli indirizzi assegnati ai bytes, dagli indirizzi per le halfwords, dagli indirizzi delle words etc.
Va anche segnalato che l'estensione della memoria di un computer o di altri dispositivi digitali di solito viene espressa mediante il numero dei suoi bytes.

B70:**b.08** Per trattare gli indirizzi e i contenuti della memoria centrale conviene avere presenti le potenze di 2 [W10:a.01].
Vediamo un esempio abbastanza realistico di una memoria di $2^{30} = 1\,073\,741\,824$ bytes, cioè di $2^{33} = 8\,589\,934\,592$ bits; per la misura di queste grandezze si usano notazioni come 1 073 741 824 B e 8 589 934 592 b.
La nostra memoria può anche essere vista come sequenza di $2^{29} = 536\,870\,912$ halfwords, come sequenza di $2^{28} = 268\,435\,456$ words, come sequenza di $2^{27} = 134\,217\,728$ doublewords e come sequenza di $2^{26} = 67\,108\,864$ quadwords.
Si può quindi indirizzare un byte in memoria con un intero che, idealmente, va da 0 a 1 073 741 823, una halfword con un intero compreso tra 0 e 536 870 911, una word con un intero da 0 a 134 217 727, una doubleword con un intero appartenente a $[0 : 134\,217\,728]$ ed una quadword con un intero naturale minore o uguale a 67 108 863.

I bytes individuabili nella memoria centrale occupano ottetti la cui prima posizione è un multiplo di 8, le halfwords sequenze la cui prima posizione è un multiplo di 16 e così via.
In una sequenza di 128 bits della memoria centrale il cui primo bit occupa la posizione i (intero multiplo di 128, la lunghezza delle doublewords) si possono vedere 16 bytes, oppure 8 halfwords, oppure 4 words, oppure due doublewords oppure una quadword, come dal seguente schema.

//input pB70b08

B70:**b.09** Come si è detto, i contenuti di ciascuna delle celle-m possono rappresentare oggetti diversi in dipendenza del ruolo che alla cella si sta attribuendo. Questo ruolo a livello di hardware dipende dai circuiti operativi che si fanno operare sulla unità, mentre a livello di software viene assegnato da una dichiarazione nel linguaggio di programmazione con il quale si programmano le elaborazioni che riguardano la unità di memoria.

Una celle-m di s bits può rappresentare un intero naturale compreso tra 0 e $2^s - 1$. Per esempio la halfword che contiene la sequenza di bits $b_0 b_1 b_2 \cdots b_{15}$ può rappresentare l'intero $\sum_{i=0}^{15} b_i \, 2^i$.

Gli interi dell'intervallo $[0 : 2^s - 1]$ sono detti **unsigned integers** su s bits.

Conviene soffermarsi a osservare alcuni interi naturali rappresentati dalle halfwords. Lo zero è rappresentato da 00000000 00000000, i successivi 5 interi da
10000000 00000000, 01000000 00000000, 11000000 00000000, 00100000 00000000 e 10100000 00000000.
Gli interi 10, 100, 1000 e 10000 sono forniti, risp., dalle sequenze
01010000 00000000, 00100110 00000000, 00010111 11000000 e 00001000 11110100.
Il massimo degli unsigned integers su 16 bits, $2^{16} - 1 = 65\,535$, è rappresentato da 11111111 11111111.

B70:**b.10** Una cella di memoria di s bits può rappresentare anche una sequenza di s valori booleani, valori da interpretare come falso se il bit vale 0 e come vero se il bit vale 1.

Ricordando la nozione di funzione caratteristica di un sottoinsieme entro un insieme finito, possiamo anche affermare che una cella-sb consente di individuare un sottoinsieme di un insieme ambiente di al più s elementi che sia stato dotato di un ordine.
Per esempio il sottoinsieme dell'insieme dei mesi dell'anno i cui nomi italiani contengono la lettera "r" è esprimibile con la halfword $011100001111_2 = 3854_{10}$.

B70:**b.11** L'insieme (finito) di interi che è più utile rappresentare con s bits è l'intervallo $[-2^{s-1} : 2^{s-1} - 1]$. In particolare le celle-16b possono rappresentare gli interi dell'intervallo $] = [-32736 : +32765]$, le celle-32b gli interi compresi tra $-2\,147\,483\,648$ e $2\,147\,483\,647$, le celle-64b gli interi da $-9\,223\,372\,036\,854\,875\,808$ a $9\,223\,372\,036\,854\,875\,807$.

Questi interi li chiameremo **signed integers** e la loro adozione, in confronto con quella degli unsigned integers, rinuncia a una metà di questi interi nonnegativi, ma consente di disporre di un ugual numero, 2^{s-1}, di numeri negativi.

Questi sono determinati secondo la cosiddetta **rappresentazione in complemento a 2**. Secondo essa i numeri nonnegativi sono caratterizzati da 0 nella posizione $s - 1$ e la sequenza degli $s - 1$ bits precedenti $b_0 b_1 b_2 \cdots b_{s-2}$ fornisce l'intero $\sum_{i=0}^{s-2} b_i \, 2^i$.
Il generico intero negativo è dato dalla sequenza della forma $b_0 b_1 b_2 \cdots b_{s-2} 1$ ed il suo valore si ottiene dalla sequenza dei bits complementari $\overline{b}_0 \overline{b}_1 \overline{b}_2 \cdots \overline{b}_{s-2} 0$ con $\overline{b}_i := 1 - b_i$ e togliendo 1 dal valore di questo unsigned integer; il valore è quindi $-\sum_{i=0}^{s-2} \overline{b}_i \, 2^i - 1$.

Vediamo le rappresentazioni di alcuni interi negativi mediante halfwords:

$$-32\,766 = -2^{15} \text{ dato da } 00000000\,00000001 \quad , \quad -32\,765 \text{ dato da } 10000000\,00000001 \quad ,$$
$$-100 \text{ dato da } 00111001\,11111111 \quad , \quad -16 \text{ dato da } 00001111\,11111111 \quad ,$$
$$-7 \text{ dato da } 11101111\,11111111 \quad , \quad -1 \text{ rappresentato da } 11111111\,11111111 \quad .$$

Può servire osservare che se n denota un signed integer dato dalla sequenza binaria $b_0 b_1 \cdots b_{s-1}$ ed $\overline{n}$ il suo complemento a uno, cioè il numero espresso da $\overline{b}_0 \overline{b}_1 \cdots \overline{b}_{s-1}$, la loro somma fornisce la sequenza di s bits uguali ad 1, cioè $n + \overline{n} = -1$; Quindi vale la formula

$$\overline{n} = -n - 1 \quad .$$

Osserviamo che per la rappresentazione in complemento a 2 la sequenza crescente delle stringhe binarie dopo lo 0 vede i successivi numeri positivi e dopo l'ultimo, $2^{s-1} - 1$, vede i numeri negativi sempre in ordine crescente a partire dal minimo, -2^{s-1} fino a -1; questo è quello che in ordine ciclico precede lo 0.

Questa rappresentazione, a prima vista piuttosto bizzarra, in realtà offre dei chiari vantaggi per la realizzazione dei circuiti operativi che implementano le operazioni aritmetiche e le relazioni d'ordine e viene adottata da tutti i costruttori di apparecchiature digitali costituendo un importante standard internazionale de facto.

Segnaliamo inoltre che le words (sequenze di 32 bits) consentono di trattare gli interi-m dell'intervallo $\mathbf{[}-2\,147\,483\,648 \;:\; 2\,147\,483\,647\mathbf{]}$, che l'insieme degli interi-m contenuti nelle doublewords (64 bits) è $\mathbf{[}-9\,223\,372\,036\,854\,875\,808 \;:\; 9\,223\,372\,036\,854\,875\,807\mathbf{]}$ e che le quadwords consentono di registrare ciascuno degli interi compresi tra $-170\,141\,183\,145\,469\,231\,731\,687\,303\,715, 884\,105\,728$ e $-170\,141\,183\,145\,469\,231\,731\,687\,303\,715, 884\,105\,727$.

B70:**b.12** Per taluni scopi risulta opportuno considerare anche celle di memoria corrispondenti a quaterne di bits; a queste celle-m si da il nome di **nibbles**.
I possibili valori di un nibble sono evidentemente 16 e sono in corrispondenza biunivoca con un intero dell'intervallo $\mathbf{[}0 : 15\mathbf{]}$; essi possono essere posti in biiezione con una cosiddetta **cifra esadecimale**.
Per queste entità vale la seguente tavola di conversione

0000	1000	0100	1100	0010	1010	0110	1110	0001	1001	0101	1101	0011	1011	0111	1111
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Per esprimere i contenuti delle unità di memoria può essere vantaggioso servirsi delle notazioni esadecimali, in base 16.
Chiaramente un byte equivale a 2 nibbles, una halfword a 4 nibbles, una word ad 8 nibbles e così via. Quindi si può esprimere il contenuto di un byte con una coppia di cifre esadecimali, il contenuto di una halfword con una quaterna di cifre esadecimeli, il contenuto di una words con 8 cifre esadecimali e così via.

Le notazioni esadecimali sono spesso utilizzate per esprimere i contenuti attuali di ampie zone della memoria centrale e per esprimere gli indirizzi della memoria centrale. Per esempio se si vogliono individuare gli indirizzi dei bytes di una memoria di un gibibyte, cioè di una memoria di $2^{30} = 1\,073\,741\,824$ bytes servono 15 cifre esadecimali: il primo byte corrisponde all'indirizzo 000 000 000 000 000, l'ultimo all'indirizzo $FFF\ FFF\ FFF\ FFF\ FFF$.

B70:**b.13** Stante la finitezza delle risorse di memoria disponibili con un computer, in ogni programma ci si deve preoccupare di non superare i limiti che derivano dalla finitezza del computer in uso.
Accenniamo alle precauzioni da assumere per elaborazioni sui numeri interi.

Se si devono trattare interi piccoli si possono usare per essi delle halfwords; se si può essere sicuri che non si incontrano interi al di sopra del miliardo si possono usare le words; altrimenti bisogna servirsi di doublewords o di quadwords.
Per trattare interi di grandezza maggiore oppure interi di grandezza imprevedibile servono sottoprogrammi che operano su sequenze di interi dei tipi precedenti e che sono in grado di assegnare a nuovi valori interi molto elevati sequenze di celle-m della estensione richiesta e prevedibile solo nel corso della costruzione dei suddetti nuovi valori.
Evidentemente più si vuole essere sicuri di non superare i limiti delle memorie, più si devono impegnare risorse di memoria, di tempo di esecuzione e di onere di programmazione.

Va anche detto in linea di massima che le grandi prestazioni dei computers attualmente disponibili consigliano di scrivere programmi che non si curano di risparmiare le risorse, ma che abbiano elevata affidabilità e in particolare che non incorrano in errori di superamento dei limiti per i numeri interi utilizzati.
Va inoltre segnalato che i sistemi di programmazione disponibili sono in grado di segnalare molti degli inconvenienti accennati.
Un comportamento spesso consigliabile consiste nello scrivere programmi provvisori semplici, anche se rischiosi, di provarli attentamente e se giudicato necessario, di redigere programmi più definitivi ben accurati rispetto alle prestazioni che possono portare a risultati non corretti.

Segnaliamo anche un altro possibile comportamento di programmazione. Di fronte a problemi impegnativi e poco chiari può essere opportuno, grazie al fatto che il costo delle sperimentazioni con il computer à in linea di massima è piuttosto basso, si possono effettuare tentativi con programmi poco approfonditi al fine di chiarire gradualmente le caratteristiche del problema e di una procedura per risolverlo disponendo anche di un certo numero di risultati empirici.

Alle precedenti considerazioni sopra gli insiemi finiti di interi trattabili si aggiungeranno considerazioni riguardanti quelli che chiamiamo real numbers; queste altre considerazioni oltre a riguardare i valori massimi di questi numeri, concernono i loro possibili valori assoluti minimi (zero escluso) e la loro precisione; prevedibilmente saranno considerazioni un poco più complesse.

B70:c. informazioni simboliche

B70:**c.01** Introduciamo ora le informazioni simboliche di più ampio uso.
I singoli bytes vengono usati primariamente e ampiamente per trattare caratteri visualizzabili come cifre decimali, lettere alfabetiche e segni di interpunzione, nonché per alcuni caratteri che svolgono ruoli importanti nel controllo di dispositivi periferici come le stampanti e nel controllo della trasmissione delle informazioni tra computers e altre apparecchiature digitali.

Attualmente la massima parte dei caratteri gestiti con il computer seguono il sistema di codifica detto **codice ASCII**. ASCII è l'acronimo di American Standard Code for Information Interchange e per la precisione occorre distinguere tra le due varianti ASCII-7 ed ASCII-8 che si servono, risp. di 7 e 8 bits. La prima variante è ampiamente utilizzata dagli anni (1966-1970) nei quali si sono imposti i computers in grado di indirizzare le celle-m di 8, 16, 32, 64 e ora 128 bits ed è univocamente definita in ambito internazionale. La seconda riguarda estensioni della prima che si differenziano soprattutto per l'inclusione di caratteri visualizzabili utilizzati nelle lingue nazionali e per caratteri utilizzati per la grafica dei più semplici videogiochi.

B70:**c.02** ASCII dedica i primi 32 bytes, corrispondenti ai valori interi da 0 a 31, e l'ultimo, corrispondente a 126, ai **caratteri di controllo**.

0	00	NUL	^	\0	null	16	10	DLE	^P	\r	data link escape
1	01	SOH	^A		start of heading	17	11	DC1	^Q		device control 1
2	02	STX	^B		start of text	18	12	DC2	^R		device control 2
3	03	ETX	^C		end of text	19	13	DC3	^S		device control 3
4	04	EOT	^D		end of transmission	20	14	DC4	^T		device control 4
5	05	ENQ	^E		enquiry	21	15	NAK	^U		negative acknowledgement
6	06	ACK	^F		acknowledgement	22	16	SYN	^V		synchronous idle
7	07	BEL	^G	\a	bell	23	17	ETB	^W		end of transmission block
8	08	BS	^H	\b	backspace	24	18	CAN	^X		cancel
9	09	HT	^I	\t	horizontal tab	25	19	EM	^Y		end of medium
10	0A	LF	^J	\n	line feed	26	1A	SUB	^Z		substitute
11	0B	VT	^K	\v	vertical tab	27	1B	ESC	^[		escape
12	0C	FF	^L	\f	form feed	28	1C	FS	^\		file separator
13	0D	CR	^M	\r	carriage return	29	1D	GS	^]		group separator
14	0E	SO	^N		shift out	30	1E	RS	^^		record separator
15	0F	SI	^O		shift in	31	1F	US	^_		unit separator

127	7F	DEL	^?	delete

B70:**c.03** I 95 bytes con valori numerici compresi tra 20_{16} e $7E_{16}$ sono dedicati ai caratteri visualizzabili o stampabili, cioè alle lettere dell'alfabeto romano-inglese, alle cifre decimali, ai segni di interpunzione e alcuni altri.

La tabella che segue presenta le loro codifiche.

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F

(spazio)	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	
p	q	r	s	t	u	v	w	x	y	z	{	|	}		

B70:**c.04** Al livello dell'hardware intorno al 1970 si è imposta l'organizzazione che favorisce l'indirizzamento di unità di 8 bits chiamate **bytes** o ottetti, unità i cui valori si possono rappresentare con interi da 0 a 255 o con coppie di cifre esadecimali (da 00 a FF). Per ogni valore binario si tende a utilizzare un intero byte, per gli interi da 16 bits due bytes consecutivi, per gli interi da 32 bits si utilizzano 4 bytes consecutivi e per gli interi da 64 bits 8 bytes consecutivi.

L'importanza dei bytes è dovuta all'ampio utilizzo dei cosiddetti alfabeti ASCII. `ASCII (we)`, acronimo di American Standard Code for Information Interchange, denota un sistema di codifica diffusosi ampiamente nella elaborazione automatica dei dati negli anni intorno al 1970 e al 1980. Esso si serve di 7 bits per rappresentare 128 caratteri con i quali in quegli anni si riusciva a soddisfare gran parte delle esigenze delle elaborazioni simboliche. Il relativo alfabeto comprende 26+26 lettere minuscole e maiuscole, cifre decimali, coppie di parentesi (“`(`”, “`)`”, “`[`”, “`]`”, “`{`”, “`}`”), segni di interpunzione (“`,`”, “`.`”, “`:`”, “`;`”, “`?`”, “`!`”, “`"`”) i segni matematici di largo uso (“`+`”, “`-`”, “`=`”, “`<`”, ”`>`”, “`^`”), pochi altri caratteri visualizzabili (“`#`”, “`&`”, “`|`”, “`*`”, “`$`”, “94”, “`’`”, “`_`”, “`\`”, ...) e 32 segnali elementari utilizzati per le telecomunicazioni.

Successivamente si è avuto un primo ampliamento dei caratteri facilmente trattabili con i codici che genericamente sono richiamati dal termine `ASCII esteso (wi)`; questi mediante ottetti di bits consentono di rappresentare 256 caratteri. Esistono diverse varianti di ASCII esteso che differiscono per una parte dei 128 caratteri non ASCII originali in relazione alle esigenze dei paesi nei quali sono utilizzate o di alcune apparecchiature specifiche. Qui faremo riferimento soprattutto alla codifica chiamata Latin-9 utilizzata nell'Europa Occidentale e standardizzata dall'ISO come `ISO 8859 (we)`-15.

Secondo questi sistemi le 26 lettere dell'alfabeto romano-inglese maiuscolo sono codificate con gli ottetti contraddistinti dai valori da 65 a 90, le minuscole con i valori da 97 a 122 e le cifre decimali con i valori da 48 a 57.

B70:**c.05** Segnaliamo anche che è previsto che possono essere trattati molti altri simboli: per questi dal 1991 viene sviluppato un sistema di codifica di larga portata chiamato `Unicode (wi)` che ha come fine la definizioni di codifiche per tutti gli alfabeti dei linguaggi naturali con una apprezzabile diffusione nel presente o nel passato e di una ampia gamma di linguaggi artificiali; complessivamente nell'ottobre 2010 risultavano definiti 109 449 caratteri di alfabeti e sillabari e, per trattare anche dei segni "artificiali", erano assegnati oltre 245 000 caratteri.
Questo sistema di codifiche è uno standard internazionale de facto e trae grande importanza dal fatto di dare delle regole ampiamente condivise per la circolazione su Internet di un flusso di messaggi ormai gigantesco e in continua crescita. Questo sistema di codifiche mette a disposizione 16, 20 o 24 bits per ogni carattere secondo uno schema che consente di trattare fino a 1 114 112 simboli.

B70:d. costanti e variabili; dichiarazioni e assegnazioni

B70:**d.01** Un computer può eseguire Un programma per un computer dotato di un opportuno corredo di strumenti può vedersi come un complesso di richeste espresse secondo regole formali piuttosto precise che, ogni volta che viene presentato al computer un adeguato complesso di dati di ingresso (numerici, logici, testuali, grafici, impulsi elettrici, ...) governa l'esecuzione di una sequenza di operazioni la quale fornisce nuovi dati da considerare i risultati della suddetta esecuzione.

Un programma in ogni sua esecuzione elabora vari dati: oltre ai dati di ingresso, vi sono dati inseriti nel programma ed altri che il programma produce nel corso dell'esecuzione; tra questi una parte vengono emessi come risultato dell'esecuzione.
Tra i dati che un programma può elaborare nel corso delle sue esecuzioni vanno distinti quelli che in ciascuna delle sue possibili esecuzioni rimangono fissi da quelli che possono subire modifiche in conseguenza di qualche comando espresso nel programma; i primi sono detti **dati costanti**, i secondi **dati variabili**.

Un computer attuale per effettuare le elaborazioni che gli possono essere richieste deve essere dotato di strumenti operativi (che in effetti sono dei programmi predisposti) decisamente complessi, ma che l'utente deve conoscere soltanto in relazione alle sue necessità. Una parte primaria di questi strumenti costituisce il sistema operativo del computer (Windows, Unix, Mac OS, ...) , sistema che interagisce con l'utente e organizza le manovre richieste per le sue molteplici prestazioni.

Per effettuare esecuzioni governate da programmi scritti in un dato linguaggio il computer deve essere dotato anche di un sistema di strumenti (altri programmi predisposti) anch'essi decisamente complessi che chiamiamo **sistema di sviluppo**. A questo punto ci limitiamo a segnalare solo due dei compiti di un sistema di sviluppo: (1) occuparsi della traduzione di un programma scritto nel linguaggio di programmazione in un complesso di istruzioni comprensibili dai dispositivi fisici (o software) costituenti il computer stesso; (2) monitorare ciascuna delle esecuzioni dei programmi segnalando gli accadimenti essenziali e in particolare gli eventuali malfunzionamenti.

Anche il sistema di sviluppo di un linguaggio come C/C++ non deve essere conosciuto in dettaglio da un programmatore; tuttavia è opportuno che egli abbia idee semplificate ma chiare delle sue componenti e delle sue prestazioni in modo da potere prendere le decisioni più opportune per il soddisfacimento delle sue esigenze. La visione del sistema di sviluppo deve essere tanto più estesa quanto più è impegnativa e sistematica l'attività di programmazione e di utilizzo dei risultati. L'approfondimento di questa visione si può configurare come formazione di un proprio metodo e di un proprio stile di lavoro. Questa formazione sarà particolarmente impegnativa per i programmatori policefali.

Occorre innanzi tutto dire che per servirsi di un linguaggio di programmazione il computer da utilizzare deve disporre di un complesso piuttosto articolato di programmi che chiameremo **sistema di sviluppo** del linguaggio.
Ogni sistema di sviluppo di un linguaggio di ampia portata è un prodotto industriale decisamente impegnativo che in buona parte dipende dal computer utilizzato e dal sistema operativo in uso. Esso svolge svariati compiti: tradurre le richieste formulate in ogni programma in richieste comprensibili per la macchina, individuare e segnalare opportunamente gli errori che eventualmente compaiono nel programma, segnalare le eventuali anomalie verificatesi nel corso dell'esecuzione, mettere a disposizione ampie librerie di sottoprogrammi di utilità generale, supportare la gestione di programmi con i quali un programma scritto dall'utente può interagire e molto altro.

Dopo il precedente semplice accenno, ritorneremo su alcuni aspetti del sistema di sviluppo solo per chiarire questioni che si andranno ponendo nel corso della presentazione delle prestazioni del linguaggio.

B70:**d.02** Possiamo assumere che a ogni dato fisso o variabile che un programma elabora viene associata una cella di memoria destinata a contenere i valori che il dato viene ad assumere nelle successive fasi di una esecuzione del programma; per un dato variabile il valore registrato nella sua cella in una fase esecutiva viene detto **valore corrente** del dato.

La associazione di una cella a un dato viene effettuata dalla componente traduttore del sistema di sviluppo.

Senza entrare nelle tecniche adottate dal traduttore dei programmi in un dato linguaggio (che può essere diverso per i diversi modelli di computer, possiamo pensare che tale traduttore organizzi una tabella che associa a ogni dato che compare in un programma la corrispondente collocazione in una zona della memoria della macchina esecutrice.

La memoria del computer dal punto di vista di un programma si può considerare come un nastro nel quale si distinguono successive zone dedicate ad aggregati di dati (e quindi di celle) tendenzialmente omogenei appartenenti ai vari tipi (bytes per caratteri, celle per interi delle diverse taglie e altri che incontreremo) che sono coinvolti dal programma.

B70:**d.03** Ciascuna delle celle-m coinvolte da un programma viene individuata dall'indirizzo del primo dei bytes a essa assegnati e dal loro numero, cioè dal tipo del dato in causa.

Per taluni programmi risulta utile saper controllare anche elaborazioni che coinvolgono l'indirizzo iniziale e l'estensione della cella-m. Come vedremo questa è una possibilità di notevole portata per il linguaggio C.

Chi formulava i programmi per i primi computers utilizzati disponeva solo dei codici binari delle istruzioni della macchina e doveva servirsi degli indirizzi fisici delle celle-m nelle quali collocare i dati riguardanti il problema che doveva risolvere. Questo modo di lavorare veniva detto "programmazione nel linguaggio macchina" e risultava assai oneroso per la distanza tra codici delle istruzioni e indirizzi delle celle da una parte e operazioni da eseguire e nomi delle variabili dall'altra e di conseguenza per il gran numero di dettagli che si dovevano tenere presenti.

Un primo miglioramento si è avuto con la introduzione di linguaggi simbolici di macchina i quali hanno consentito di controllare le prestazioni dei computers di un dato modello mediante istruzioni espresse simbolicamente e mediante nomi per le celle di memoria.

Successivamente gli sviluppi tecnologici hanno reso i computers sempre più miniaturizzati, efficienti, affidabili e versatili e li hanno dotati di una varietà di dispositivi ausiliari e di collegamenti.

Contemporaneamente e sinergicamente gli elaboratori elettronici sono stati dotati di una crescente varietà di strumenti software atti a facilitarne l'utilizzo: sistemi operativi, traduttori di linguaggi, librerie di sottoprogrammi, strumenti per la diagnosi di malfunzionamenti, sistemi per l'archiviazione di dati, strumenti per lo sviluppo di programmi via via più ambiziosi, ...)

L'utilizzo dei computers, nel frattempo diventati miliardi, è passato dall'essere un fenomeno legato ad applicazioni specifiche e a singoli modelli di hardware al diventare una fenomenologia da esaminare in relazione all'andamento dell'economia e delle imprese, allo sviluppo della società e a una nuova cultura che da ampio credito alle metodologie e alle soluzioni condivise.

Senza addentrarci nei molteplici aspetti di questi sviluppi, ci porremo dal punto di vista della adozione di un linguaggio di programmazione procedurale di livello medio e di portata vasta come il C e il suo

successore C++, cercando tuttavia di segnalare i collegamenti tra i suddetti sviluppi e le metodologie di programmazione.

B70:**d.04** In un linguaggio come mC i dati costanti sono individuati mediante scritture convenzionali in grado di esprimerne precisamente il significato, mentre le variabili sono identificate da nomi che vengono scelti dal programmatore. È opportuno che la scelta degli identificatori delle variabili sia effettuata sull'intero complesso dei dati del problema preoccupandosi che essa faciliti il più possibile la individuazione ed il ricordo dei rispettivi significati e ruoli.
Questo conta tanto più quanto maggiore è la possibilità che un programma venga ripreso e ampliato o adattato in tempi successivi al suo primo sfruttamento ed eventualmente da persone diverse dal programmatore originario.

Il sistema di sviluppo del linguaggio si incarica di assegnare a tutti i dati in un programma le opportune celle-m e di inserire i valori costanti nelle celle per i dati fissi. In genere anche nelle celle per i dati variabili sono inseriti dei valori iniziali prestabiliti, ma questo non avviene in modo univoco su tutti i computers e potrebbe non essere controllabile senza rischi; è quindi consigliabile che il programmatore si preoccupi del valore portato da ciascuna variabile prima del suo utilizzo iniziale.

Non è utile entrare nei dettagli dei collegamenti tra gli identificatori delle variabili e le celle di memoria loro assegnato, dettagli che dipendono dal comportamento interno del sistema di sviluppo: è sufficiente osservare che esso nella fase di traduzione del programma si organizza una tabella che gli permette di conoscere anche in fase esecutiva questa relazione.

B70:**d.05** Veniamo alle regole per il controllo dei dati nel caso di macchina programmabile con il linguaggio mC, strumento per il quale useremo l'abbreviazione **MSPmC**.

Diciamo **alfabeto degli identificatori** l'insieme costituito dai caratteri alfabetici, dalle cifre decimali e dal segno "_".
Un identificatore è dato da una stringa di caratteri dell'alfabeto degli identificatori la cui iniziale non può essere una cifra ma deve essere una lettera o il segno "_".

Gli identificatori dei dati da elaborare sono introdotti in un programma da frasi dichiarative con le quali si stabilisce anche il tipo del dato identificato. Esempi:

```
boolean accettabile, presenza, daEsaminare;
char iniz, finale, categ;
integer j, k2r, lunInCar, valtot, maxValue, numPoints;
```

La scelta degli identificatori per un programma che prevedibilmente sarà ritoccato con una certa frequenza va esaminata con cura. In linea di massima il programmatore deve scegliere tra due esigenze che possono entrare in conflitto: da un lato scegliere identificatori concisi, dall'altro decidere identificatori leggibili e mnemonici, cioè in grado di suggerire e/o ricordare il significato dei dati che saranno richiamati. Negli esempi sono accennate soluzioni intermedie con stringhe relativamente concise e abbastanza leggibili, in alcuni casi grazie al ricorso di "stringhe a dorso di cammello", stringhe scandibili in sottostringhe grazie alla comparsa di poche maiuscole entro le molte minuscole. Vediamo come possono essere introdotte le costanti dei tipi che ora ci limitiamo a considerare, cioè degli interi, dei valori booleani e dei caratteri visualizzabili. Queste scritture sono chiamate anche "literals".

B70:**d.06** Le costanti intere possono essere espresse con notazioni relative a diverse basi.
Con notazioni decimali decimali: 35 23009 -16 -1000000
Con notazioni ottali: O14 sta per l'intero 12
Con notazioni esadecimali: 0xc sta per 12, come l'equivalente 0XC.

A ciascuna di queste costanti potrebbero essere assegnate celle-m di estensioni diverse in dipendenza della grandezza del valore assoluto.

Se si vuole introdurre una costante da assegnare a una cella-64b, del tipo chiamato `long integer` è necessario usare una notazione literal, scrittura decimale seguita dalla lettera L o dalla l: per esempio si introduce una cella-64b per il numero 12 scrivendo `12L`.

Sono disponibili lostanti booleane TRUE e FALSE, equivalenti rispettivamente a 1 e 0.

Anche i contenuti dei singoli bytes possono essere introdotti con notazioni diverse costituenti i cosiddetti literals di carattere. Un tale literal è costituito da una rappresentazione del carattere da esprimere delimitata da da due segni di apostrofo (segno chiamato anche single quote e accento grave).
Tutti i caratteri sono rappresentabili con notazioni ottali di una delle forme `\`*o* `\`*oo* `\`*ooo* , dove *o* rappresenta una cifra ottale e il numero espresso dopo il segno `\` rappresenta la posizione del carattere nella sequenza dei caratteri ASCII-8.
I literals più semplici ed evidenti sono disponibili per i caratteri visualizzabili e si sono ottenuti dal carattere in causa delimitato da due segni di apostrofo : `'a'` `'!'` `'$'`.
Otto caratteri ASCII non visualizzabili importanti per la programmazione sono espressi mediante sequenze escape come segnalato dalla seguente tabella:

newline	hor.tab	vert.tab	backspace	carriage return	form feed	backslash	single quote
\n	\t	\v	\b	\r	\f	\\	\’
’\n	\t	\v	\b	\r	\f	\\	\”

B70:**d.07** Vediamo ora alcune frasi di dichiarazione e di assegnazione per variabili dei tipi fondamentali dei numeri interi e dei bytes.

```
int step,stepNumMax;
short int matrIscritti,numIscritti,indScolari,numScolari;
long int numIntntAddr;
stepNumMax=200000000; numIscritti=102;
numScolari=28; numIntntAddr=2000000000;
```

La prima frase stabilisce che i primi due nomi identificano due variabili intere che occupano due celle-32b, cioè 2 bytes; la seconda che i 4 nomi che seguono `short int` sono gli identificatori di altrettante variabili intere che richiedono ciascuna una cella-16b; la terza richiede che `numIntntAddr` rinvii a una cella-64b in grado di contenere interi i cui valori possono variare nell'intervallo $\mathbf{[}-2^{31} : 2^{31}-1\mathbf{]}$. Queste frasi sono dette dichiarative ed hanno anche l'effetto di individuare precise posizioni (indirizzi) della memoria centrale per le suddette celle-m e di predisporre che i rispettivi contenuti in esecuzione siano trattati come numeri interi con segno.

Le due ultime frasi assegnano i numeri scritti a destra del segno “`=`” come valori attuali delle variabili scritte a sinistra di questo carattere. Le frasi di questo tipo, quando compaiono dopo le frasi dichiarative delle variabili esibite (in genere all'inizio di un programma, prima di ogni altra frase che coinvolga la rispettiva variabile) sono dette frasi di inizializzazione.

```
char lettScrlt,carDL,carrReturn;
lettScrlt='A';
carDL='\044';
carrReturn='\r';
```

Il primo dei tre enunciati è una frase dichiarativa e stabilisce che ciascuno dei tre nomi che seguono la parola riservata `char` nel programma che segue è destinato a controllare una cella-8b; viene anche fissato l'indirizzo di tale cella, ma il programmatore può evitare di conoscerlo in quanto potrà servirsene usando semplicemente il corrispondente identificatore.
I tre enunciati che seguono sono frasi di assegnazione e stabiliscono quale ottetto di bits verrà inserito come valore attuale alla corrispondente cella. Al posto di queste frasi avrebbero potuto comparire le seguenti tre equivalenti:

```
lettScrlt='\100'; carDL='$'; carrReturn='\15';
```

B70:**d.08** Una frase dichiarativa e la frasi di inizializzazione della variabile dichiarate si possono unificare. Invece delle cinque frasi riguardanti variabili intere di B70d07, supposto che le frasi di assegnazione siano frasi di inizializzazione si possono sostituire con le seguenti:

```
int step, stepNumMax = 200000000;
```

```
short matrIscritti,numIscritti=102,indScolari,numScolari=28;
```

```
long numIntntAddr=2000000000;
```

Si osservi che le parole riservate `short int` e `long int` si possono sostituire con le equivalenti `short` e `long`.

B70:e. arrays e stringhe

B70:**e.01** Per affrontare moltissimi problemi si devono utilizzare e costruire sequenze di dati omogenei, in genere di dati dello stesso tipo.
Una sequenza di numeri naturali può servire a registrare misurazioni di grandezze ottenute in tempi successivi come cardinali di insiemi, lunghezze, pesi, durate, velocità, quantità di denaro, temperature di un paziente, concentrazioni, percentuali, ... Altre sequenze possono fornire i numeri degli abitanti di una sequenza di città o nazioni, le altitudini di località incontrate in un percorso,
Assieme alle sequenze numeriche si possono considerare sequenze di caratteri; esempi di queste sequenze sono dati dai caratteri che si incontrano in una parola di una lingua naturale, nella denominazione di una località, nel nome di una persona, nei simboli di un composto chimico, in un acronimo, ...

Il modo canonico per registrare in una memoria una sequenza di dati consiste nel collocarli in una sequenza di celle consecutive della memoria centrale.
Una sequenza di celle è localizzata dall'indirizzo della prima cella e dal numero delle celle, cioè dal numero delle sue componenti. Ciascuna delle sue celle viene individuata dall'indirizzo della cella iniziale e dal numero di celle sulle quali si deve avanzare per giungere a quella richiesta.

B70:**e.02** Nel linguaggio mC, come sostanzialmente in tutti i linguaggi di programmazione procedurali, le sequenze in memoria sono controllate dagli **arrays**. Questi sono variabili collettive che consentono al programmatore di accedere facilmente alle componenti delle sequenze che rappresentano senza preoccuparsi della loro precisa collocazione in memoria (di questa si fa carico il sistema di sviluppo).
Con C/C++ si possono trattare arrays di una, due o più dimensioni; qui ci occupiamo solo degli arrays monodimensionali con componenti intere o costituite da bytes.

Per disporre di arrays di interi e di bytes servono dichiarazioni come le seguenti.

```
char denom[25];
short denonL,incassiN;
int incassi[50];
```

La prima frase dispone che servendosi dell'identificatore tt denom si intendono trattare denominazioni di al più 25 caratteri il cui numero attuale sia determinato dal valore attuale della variabile intera `denomL`. La seconda chiede di controllare attraverso l'identificatore `incassi` sequenze di al più 50 interi il cui numero attuale sia contenuto nella variabile `incassiN`. Inoltre resta inteso che i valori attuali dei caratteri siano elementi dell'alfabeto ASCII (verosimilmente visualizzabili) e che gli interi siano rappresentati in 32 bits e quindi possano appartenere all'intervallo $\mathbf{[}-2^{31}:2^{31}-1\mathbf{]}$.

B70:**e.03** Il programma in cui compaiono le dichiarazioni precedenti potrebbe presentare anche frasi di assegnazione come le seguenti:

```
denomL=9;
denom[0]='c; denom[1]='e'; denom[2]='l'; denom[3]='l';
denom[4]='u'; denom[5]='l'; denom[6]='a'; denom[7]='r'; denom[8]='i';
```

Queste frasi esprimono richieste operative simili, ciascuna delle quali avente l'effetto di determinare un contenuto di byte a partire da una **scrittura di costante**, quella che compare a destra del segno "`=`", e di assegnarlo come valore attuale a una cella di memoria determinata dalla scrittura alla sinistra dello stesso segno "`=`".

Va osservato che questo segno non serve a esprimere una relazione di uguaglianza o a proporre una equazione; esso infatti richiede una azione consistente nella assegnazione di un nuovo valore ad una variabile, cioè nella modifica del contenuto di una cella-m. Possiamo dire che "`=`" esprime una operazione di assegnazione.

La prima frase di assegnazione comporta la determinazione della rappresentazione dell'intero 9, costituita dalla sequenza di 32 bits `00000000000000000000000000001001`, e l'inserimento di tale valore nei 4 bytes di memoria associati all'identificatore di variabile intera `denomL`.
Veniamo alla frase `denom[4]='u';` : essa comporta la determinazione della rappresentazione ASCII del carattere "`u`", costituita dall'ottetto `10101110`, e l'inserimento di tale ottetto nel byte della memoria associato alla quinta delle celle per caratteri delle 25 associate all'array di caratteri `denom`, cioè alla cella che si ottiene avanzando di 4 posizioni rispetto alla prima assegnata all'array (questa accessibile attraverso l'espressione `denom[0]` .

Si osserva che gli attributi ordinali che si usano discorsivamente per individuare le componenti di una sequenza (prima, seconda, terza, ...n-sima, ...) sono sfasati rispetto agli indici degli arrays ([0], [1], [2], ..., $n-1$, ...).

Va anche rilevato che anche l'indicazione che segue l'identificatore di un array per individuarne un componente va interpretata come azione piuttosto che come precisazione statica come per le entrate delle matrici. Una scrittura come `denom[4]` comporta la valutazione di in valore numerico (in questo esempio 4, ma tra le parentesi quadre si potrebbe avere una espressione aritmetica) seguita da un incremento per tale valore dell'indirizzo iniziale dell'array. Questo indirizzo è misurato in bytes per `denom`, in quaterne di bytes per un array di tipo `int` e similmente per le rimanenti celle-m.

B70:**e.04** Se vogliamo disporre dei primi 10 numeri della `successione di Fibonacci (wi)` nelle prime 10 componenti dell'array che ha come identificatore `val` , si possono utilizzare le frasi di assegnazione che seguono.

```
FibLun=10;
Fib[0]=0; Fib[1]=1; Fib[2]=1; Fib[3]=2; Fib[4]=3; Fib[5]=5; Fib[6]=8;
Fib[7]=13; Fib[8]=21; Fib[9]=34;
```

Ciascuna di queste 11 frasi, che viene chiusa dal segno "`;`" con funzione di separatore di frasi, comporta l'assegnazione del valore attuale di una variabile numerica intera alla cella associata alla variabile stessa; la prima riguarda la variabile `valN`, le successive coinvolgono le prime 10 celle associate all'array **val**.

Se dell'array `Fib` servono solo 10 componenti si può utilizzare la seguente dichiarazione più sintetica e sicura (ma più rigida)

```
int val[10] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34 } ;
```

Questo è un esempio di dichiarazione e inizializzazione di un intero array di interi. In generale una frase di questo tipo presenta nell'ordine:
la parola chiave del tipo di array da introdurre;
l'identificatore dell'array seguito dal numero delle sue componenti tra parentesi quadre;
il segno "`=`";
l' lista dei valori da assegnare alle successive componenti separati da virgole e delimitata da parentesi graffe.

B70:**e.05** **(1) Eserc.** Predisporre nelle prime 20 componenti di un array i primi numeri primi.

(2) Eserc. Predisporre nelle 12 componenti di un array i numeri dei giorni relativi ai successivi mesi di un anno bisestile.

B70:**e.06** Le sequenze di caratteri, cioè le stringhe, rivestono grande importanza nella elaborazione dei dati. Innanzi tutto le stringhe di caratteri visualizzabili consentono di trattare tutte le informazioni esprimibili nei linguaggi naturali: nomi e descrizioni di persone, oggetti di ogni genere, prodotti, norme, idee e quant'altro. Ogni testo può essere ridotto a un complesso di stringhe.
Vi sono poi le informazioni che vengono espresse mediante linguaggi convenzionali e artificiali: espressioni matematiche, enunciati della logica, formule chimiche, ...
Un vasto sottocampo di quest'ultino riguarda i testi trattabili con strumenti informatici e in particolare i testi sorgente dei linguaggi di programmazione e di gestione delle basi dati, i testi in grado di governare il tracciamento di figure e i testi dei linguaggi per la stampa, per la grafica e per la comunicazione come TEX `(we)`, `HTML (we)`, `XML (we)` ed `SVG (we)`.

Riconosciuta l'importanza delle stringhe, il linguaggio C ha reso disponibili utili strumenti per il trattamento delle singole stringhe.
Le prime componenti di C che consideriamo sono gli **string literals**, le scritture che consentono di definire concisamente delle stringhe costanti. Queste scritture sono costituite da sequenze di caratteri ASCII racchiuse tra doppi apici come

```
"stringa"  ,  "Avvertimento"  ,  "testo costituito da 112 caratteri"
```

Con string literals si possono trattare anche stringhe molto lunghe che conviene scrivere utilizzando più linee del file che fa da supporto fisico del testo sorgente: basta per questo far comparire come ultimo carattere di una linea di testo il carattere backslash "\" .

```
"Questa frase piuttosto, lunga e verbosa come potete constatare direttamente,\
viene scritta su due linee del supporto del testo sorgente."
```

In uno string literal possono comparire anche caratteri ASCII non visualizzabili. In particolare si possono avere caratteri con effetti tipografici come new line, carriage return ed horizontal tab. Per esempio uno string literal contenente il carattere new line rappresentato da "\n" se inviato a una stampante comporta la emissione di due linee.
String literals nei quali compaiono molti caratteri di controllo possono avere effetti di stampa o di emissione su video piuttosto elaborati.

Ogni string literal viene collocato in una sequenza di bytes occupati dai successivi caratteri tra i doppi apici seguiti da un ottetto contenente il carattere null, il primo dell'alfabeto ASCII costituito da otto bit uguali a 0.

B70:**e.07** Osserviamo che se avessimo voluto trattare il nome del matematico Izrail Gelfand nella sua versione cirillica o nella yiddish non sarebbe sufficiente l'alfabeto ASCII e si dovrebbe fare ricorso al più generale sistema `Unicode (wi)`.
Similmente se avessimo voluto trattare i primi 10 `numeri di Mersenne (wi)` non sarebbero stati sufficienti 10 celle di memoria dedicate a interi dell'intervallo $[2^{31} : 2^{31} - 1]$, in quanto solo i primi 8 numeri sono trattabili con queste celle (l'ottavo è $M_{31} = 2^{31} - 1$, mentre il nono $M_{61} = 2305843009213693951$ e il decimo $M_{89} = 618970019642690137449562111$ superano la capacità di tali celle.

Se si vogliono trattare numeri interi molto elevati, come vedremo, si devono adottare modalità di codifica più complesse oppure rinunciare alla precisione ed accontentarsi di valutazioni approssimate le quali non sono in grado di rispondere a certe esigenze e a risolvere certi tipi di problemi; ad esempio sono del tutto inutilizzabili per trattare problemi di natura crittografica.

Osserviamo che i limiti finiti, ovviamente, degli strumenti effettivi devono essere tenuti ben presenti nella pratica computazionale. Tali limitazioni nello sviluppo di considerazioni generali, viceversa,

possono essere considerate quasi un intralcio alla formulazione rapidamente comprensibile delle argomentazioni e dei risultati.
In effetti gli atteggiamenti di chi porta avanti studi matematici specialistici e di chi si occupa di calcoli specifici possono essere piuttosto diversi e in certi aspetti contrastanti.

B70:f. operazioni di lettura e scrittura [1]

B70:**f.01** Anche i più semplici programmi prevedono la possibilità di leggere i dati concernenti una istanza del problema da risolvere da un'apparecchiatura di ingresso e la possibilità di scrivere su un dispositivo di uscita i risultati.
Spesso è opportuno far precedere l'immissione dei dati con richieste esplicite. Inoltre in genere è opportuno accompagnare i risultati destinati a un operatore/utente umano con segnalazioni sufficientemente chiare dei significati; per certe applicazioni servono anche emissioni che chiariscano il contesto nel quale si sono avuti dei risultati intermedi ritenuti critici.

Dei molteplici dispositivi di ingresso e uscita oggi disponibili nelle presenti sezioni prendiamo in considerazione solo pochi tipi.

Per la sola lettura e per la sola scrittura si considerano solo files sequenziali simbolici, descrivibili come nastri di bytes sui quali i dati sono registrati flussi di stringhe ASCII leggibili in chiaro. Taluni di questi sono usati solo per letture e altri solo per scritture; in questa seconda categoria collochiamo anche le stampanti.

Possono tuttavia essere utilizzati anche files sequenziali simbolici da utilizzare in fasi di scrittura alternate a fasi di scrrittura. In particolare un tale file in un primo momento viene letto e successivamente viene esteso con nuovi dati prodotti dal programma. Oppure un tale file viene prima scritto con dati che si possono descrivere come risultati intermedi, e in una seconda fase viene riletto dall'inizio per altri calcoli che operano anche sui dati intermedi.

Si considero anche dispositivi costituiti da terminale video e tastiera. Questi possono essere utilizzati solo in lettura, solo in scrittura, oppure in modo interattivo per scambi bidirezionali di stringhe leggibili.

Sulle operazioni di entrata/uscita non entreremo in molti dettagli ma ci limiteremo a descrivere il comportamento di pochi sottoprogrammi ai quali demanderemo sistematicamente le manovre di immissione ed emissione.

Secondo questo primo modo di presentare le operazioni di immissione nella ed emissione dal computer programmabile possono essere trasferite solo stringhe ASCII; queste in genere sono costituite per la maggior parte da caratteri leggibili ma contengono anche caratteri incaricati di organizzare in record successivi i testi da leggere e in linee e pagine i testi da emettere.

B70:**f.02** In questa e nelle prossime sezioni prenderemo in considerazione solo programmi molto semplici che presentano richieste di lettura iniziali, successive elaborazioni su informazioni numeriche e simboliche e richieste di scrittura finali.

Per le richieste di lettura si dovrà quindi ricorrere sistematicamente a trasformazioni di stringhe ASCII immesse in dati interni (ora solo numeri interi e per le richieste di scrittura a trasformazioni di dati interni interi e stringhe ASCII in stringhe abbastanza complesse organizzate in linee e pagine. Le prime di queste trasformazioni le diciamo **operazioni di codifica**, le seconde **operazioni di decodifica**.

I files simbolici sequenziali prima dell'immissione possono essere preparati con comuni source editors, mentre dopo l'emissione possono essere visionati come pagine stampate o con un source editor che gestisce schermate e permette di modificare i files stessi.
Questi files possono dunque servire per passare informazioni da un programma ad uno successivo. Evidentemente si possono utilizzare in modo coordinato anche gruppi estesi di programmi; per denotare questi sistemi di programmi si usa il termine **programmi da utilizzare in cascata**.

B70:**f.03** I programmi più semplici non fanno altro che emettere un messaggio sul dispositivo di uscita standard; con un tale messaggio il programma non può che segnalare il fatto di essere operativo.

Uno di questi programmi ha il seguente testo sorgente.

```
#include <iostream.h>
int main()
{
    cout << "Il programma con una sola scrittura e‘ operativo" << endl ;
    return(0);
}
```

Commentiamo rapidamente il testo. La prima linea rende disponibile una libreria di sottoprogrammi che consente di servirsi del comando `cout` e della costante `endl`. con i quali si organizza l'emissione.
La seconda linea costituisce l'intestazione del programma e il suo corpo viene presentato tra le due parentesi graffe che ne costituiscono i delimitatori accoppiati.
La prima linea del corpo con il comando `cout` richiede l'emissione della stringa che segue racchiusa tra due doppi apici e un carattere di fine linea rappresentato da `endl`. La seconda costituisce una richiesta di conclusione delle operazioni.

B70:**f.04** Programmi lievemente più articolati richiedono solo la lettura di taluni valori e la loro successiva riemissione. Il programma che segue prevede la lettura di pochi numeri seguita dalla loro ripresentazione.

```
#include <iostream.h>
int main()
{
    int k, m, n ;
    cin >> k >> m >> n ;
    cout << "k = " << k << ", m = " << m << ", n = " << n ;
    getch(); // frase (1)
    return(0);
}
```

La frase che inizia con `cin` prevede la immissione di tre scritture decimali di interi, la loro codifica e l'inserimento delle corrispondenti sequenze di 32 bits nelle celle assegnate alle variabili `k`, `m` ed `n`; si ha poi l'emissione di questi tre valori preceduti dalle variabili con le quali vengono gestiti nel programma stesso.
La linea `getch();` richiama un sottoprogramma che fa richiedere all'utente del programma di immettere un carattere; prima di questa azione l'utente ha la possibilità di osservare l'emissione e quindi di porre fine all'esecuzione con la battuta di un carattere qualsivoglia; la sua lettura sarà seguita dall'esecuzione della frase `return(0);` conclusiva. Qui viene programmata una semplicissima interazione

B70:**f.05** Il programma precedente prevede un semplicissimo dialogo interattivo, ossia uno scambio di informazioni tra utente del programma e processo esecutivo; qui l'utente può semplicemente scegliere un tempo di osservazione.
Vedremo in seguito come si possono predisporre delle sessioni interattive più articolate, a cominciare da quelle che permettono di controllare l'adeguatezza, la coerenza e la completezza dei dati immessi da tastiera per evitare che qualche errore o qualche mancanza sui dati immessi comporti la esecuzione di

elaborazioni con dati intermedi e risultati finali non voluti, dati che potrebbero avviare elaborazioni del tutto inutili o, peggio, elaborazioni che illudono sul significato dei risultati ottenuti, in quanto basate su dati erronei.
In effetti le azioni che si devono effettuare per garantire la bontà dei dati immessi richiedono controlli e validazioni che possono effettuarsi solo utilizzando i comandi di selezione ed iterazione che introdurremo in B70g. Inoltre la logica di molte operazioni di lettura controllata è piuttosto articolata e in genere conviene che tali controlli siano organizzati in appositi sottoprogrammi. Queste manovre saranno riprese nella sezione B70d.

B70:**f.06** Le scritture delimitate da doppi apici consentono di esprimere tutte le stringhe trattabili con sequenze di bytes. In precedenza abbiamo visto solo scritture di stringhe costituite da caratteri semplicemente visualizzabili, lettere, ",", "'", spazi bianchi. Vi sono però vari bytes che devono essere rappresentati da sequenze di altri caratteri chiamate **sequenze di escape**. Essi sono precisati dalla seguente tabella.

`\n` new line, inizia nuova linea
`\h` horizontal tab, avanzamento a posizione orizzontale predefinita
`\v` vertical tab, avanzamento a posizione verticale predefinita
`\b` backspace, arretramento
`\r` carriage return, ritorno a inizio linea
`\f` form feed, inizia nuova pagina
`\a` alert, suono
`\\` backslash
`\?` question mark, punto interrogativo
`\‘` singolo apice
`\"` doppio apice
`\0` NUL, byte di 8 bits nulli
`\`*ooo* rappresentazione di un byte mediante 1, 2 o 3 cifre ottali
`\x`*hhhh* rappresentazione di informazione mediante 1, 2, 3 o 4 cifre esadecimali

B70:**f.07** Il linguaggio mC consente di formulare una ampia gamma di espressioni servendosi di operandi (costanti, variabili e componenti di arrays), di svariati operatori e di parentesi tonde aventi innanzi tutto lo scopo di delimitare sottoespressioni.
Tra gli operatori si distinguono gli operatori aritmetici, gli operatori relazionali e gli operatori logici.
Gli operatori aritmetici riguardano operandi numerici, cioè operandi interi o reali-C, e forniscono un risultato numerico.

`+` operatore binario di addizione di operandi numerici, usato anche come operatore unario con il solo effetto di evidenziare un valore positivo o un non cambiamento di segno;
`-` operatore binario di sottrazione tra due operandi numerici e operatore unario prefisso che riguarda un valore negattivo o il cambiamento di segno di un operando numerico;
`*` operatore binario di moltiplicazione di operandi numerici;
`/` operatore binario di divisione tra operandi numerici;
`%` operatore binario di resto di divisione tra due operandi interi;
`++` operatore unario di incremento per un operando intero che può essere usato come prefisso e come suffisso [B70f09];
`--` operatore unario di decremento per un operando intero che può essere usato come prefisso e come suffisso [B70f09].

B70:**f.08** Una espressione aritmetica costituisce la richiesta di un complesso di operazioni, ciascuna richiesta da un operatore e coinvolgente uno o due operandi; questi possono essere indicati nell'espressione stessa oppure essere forniti come risultato di una operazione eseguita in precedenza. Il succedersi delle operazioni richieste da un'espressione viene regolato da regole di precedenza per l'esecuzione tra i due operatori che nel corso del calcolo di un'espressione vengano a essere separati da un solo operando.

Tra gli operatori aritmetici la precedenza maggiore riguarda `++`, `--` e `-` unario; seguono gli operatori `*`, `/` e `%`; infine la precedenza minore è quella di `+` e `-`.

Tra due operatori successivi della stessa precedenza viene eseguito per primo il più a sinistra. La precedenza può essere modificata dalla presenza di coppie di parentesi tonde che vengono a delimitare sottoespressioni che devono essere valutate prima e indipendentemente da quanto sta loro intorno.

B70:**f.09** Vediamo alcuni esempi di espressioni numeriche molto semplici.

L'espressione `5+7*3` implica per prima cosa il calcolo di 21 (`+` precede `+`) e quindi il calcolo di `5+21` e la messa a disposizione del contesto, cioè dell'enunciato in cui si trova l'espressione stessa del valore numerico 26.

Se si vuole invece il calcolo della somma `5+7` seguito dalla moltiplicazione del risultato per 3 va usata l'espressione `(5+7)*3` in modo da ottenere 24.

`-31*4-9` fornisce `-133`; l'espressione `a-b-c` equivale alla `(a-b)-c` ed è diversa dalla `a-(b-c)`, equivalente alla `a-b+c` e alla `(a-b)+c`; a sua volta questa fornisce valori diversi da quelli calcolati dalla `a-(b+c)`.

La `(a-b)*(a+b)` equivale alla `a*a-b*b`; `-7*8` fornisce `-56`, come `7*(-8)`, mentre `7*8` e `(-7)*(-8)` forniscono 56.

Per l'operatore divisione applicato a due operandi interi, il dividendo ed il divisore, è opportuno distinguere il caso in cui il primo è multiplo del secondo e quando non vale questa proprietà.

`144/8` fornisce 12, numero chiamato quoziente; `144/8/2` vale 9, come `(144/8)/2`, mentre `144/(8/2)` conduce a 36;

`144/8` fornisce 12; `144/8/2` vale 9, mentre `144/(8/2)` produce 36. `-6+44/11` rende disponibile `-2`.

Quando i il dividendo non è multiplo del secondo la divisione conduce a un quoziente intero ottenuto per troncamento: `18/7` porta a 2, mentre `(-18)/7` e `18/(-7)` producono `-3`.

Se il dividendo è multiplo del divisore il quoziente moltiplicato per il divisore riporta al dividendo; questa "ricostruzione" non vale nel caso in cui il divisore non divide il dividendo; in questo caso serve conoscere anche il resto della divisione.

Nel linguaggio C il resto della divisione tra gli interi `h` e `k` si ottiene con l'operatore %: quindi `7%3` vale 1 e `44%13` fornisce 5.

Chiaramente per ogni coppia di interi trattabili `h` e `k` vale l'uguaglianza tra `k` e l'espressione $(\texttt{k}/\texttt{h}) * \texttt{h} + (\texttt{k}\%\texttt{h})$.

L'operatore resto si può usare liberamente nelle espressioni numeriche del linguaggio.

$\texttt{12}\%\texttt{5} + \texttt{40}\%\texttt{7} * \texttt{23}\%\texttt{8}$
fornisce $\texttt{2} + \texttt{5} * \texttt{7}$, cioè 37.

B70:**f.10** Presentiamo alcuni esempi di enunciati per la valutazione di espressioni.

I due enunciati

```
int k = 6;  cout << "k*(k+1)/2 fornisce ",k*(k+1)/2," .";
```

comportano l'emissione di

```
k*(k+1)/2 fornisce 21 .
```

Il frammento di programma che segue mostra gli effetti dell'operatore `++` usato per il preincremento e per il postincremento e dell'operatore `--` usato per il predecremento e per il postdecremento.

```
int n=5; int m=3;
cout << n << ", " << (n++) ", " << (--m) " ;"; // emette   5, 6, 2;
cout << ++n << ", " << m++ ", " << m*n " ;"; // emette   7, 2, 21;
cout << (++m)++ << ", " << (n++)*(++n) ", " << m*(++n) ; // emette   4, 63, 50
```

B70:**f.11** **(1) Eserc.** Esprimere le richieste per il calcolo dell'area di un particolare rettangolo con i vertici aventi coordinate intere.

(2) Eserc. Esprimere le richieste per il calcolo del volume di un parallelepipedo retto rettangolo i cui vertici sono esprimibili con coordinate intere.

(3) Eserc. Esprimere le richieste per la emissione delle 4 permutazioni circolari della parola `nove` e in genere di una stringa di 4 caratteri diversi.

(4) Eserc. Esprimere le richieste per la individuazione e l'emissione di tutti i prefissi di una particolare sequenza di 5 caratteri.

B70:**f.12** Come si è detto, nei linguaggi C e C++ si trattano valori booleani, cioè valori interpretabili come true e false, in particolare valori che determinano le scelte esecutive controllate dai costrutti che vedremo in B70d.
I due valori di verità true e false sono implementati attraverso valori interi e questo consente di effettuare operazioni numeriche sopra valori ottenuti valutando espressioni booleane e viceversa prendere decisioni riguardanti scelte basandosi su valori forniti da espressioni numeriche.

Una espressione valutata false fornisce l'intero 0, mentre una espressione che porta al valore true fornisce il valore intero 1. Se invece si vuole ricavare un valore di verità da un numero intero, si ottiene false dal numero 0 e true da ogni altro numero intero.

B70:**f.13** Gli operatori relazionali sono operatori binari che prevedono due operandi numerici e forniscono un valore che in linea di principio andrebbe usato come valore booleano.

- `<` operatore "minore di";
- `<=` operatore "minore o uguale di";
- `>` operatore "maggiore di";
- `>=` operatore "maggiore o uguale di";
- `==` operatore "uguale a";
- `!=` operatore "non uguale a".

Gli operatori logici sono operatori binari o unari che, in linea di principio, prevedono operandi logici e risultato logico.

- `&&` operatore binario AND; fornisce true sse entrambi gli operandi valgono true;
- `||` operatore binario OR; fornisce false sse entrambi gli operandi assumono il valore false;
- `!` operatore unario prefisso di negazione, NOT; scambia i valori true e false.

Per quanto riguarda le precedenze esecutive, prevale l'operatore unario !, seguono gli operatori relazionali e l'operatore `&&` e per ultimo viene `||`; inoltre gli operatori numerici hanno la precedenza sui relazionali e questi sui logici.

Per avere espressioni più leggibili tuttavia è consigliabile nelle espressioni con operatori relazionali e logici di far uso di coppie di parentesi anche non strettamente necessarie.

B70:**f.14** Vediamo alcuni esempi.

L'espressione `35<= i && i <= 73` fornisce il valore `true`, ossia l'intero 1, sse il valore attuale della variabile `i`, che supponiamo intera, appartiene all'intervallo $\mathbf{[}35 : 73\mathbf{]}$; essa quindi implementa la funzione indicatrice di questo intervallo entro l'insieme degli interi standard per il compilatore.

L'espressione `i < 0 || 10 <= i` vale `true` sse il valore attuale della `i` è inferiore a 0 oppure è maggiore o uguale a 10; essa quindi implementa la funzione indicatrice $\mathcal{I}_{\mathbb{Z}}[\mathbf{(} : 0\mathbf{)} \cup \mathbf{[}10 : \mathbf{)}]$.

L'espressione `1 <= i&&i <= 6&& - 1 <= j&&j <= 7` fornisce 1 sse il punto-ZZ $\langle i, j\rangle$ appartiene al rettangolo-ZZ caratterizzato dai vertici opposti $\langle 1, -1\rangle$ e $\langle 6, 7\rangle$ e produce 0 in caso contrario; essa quindi implementa la funzione indicatrice del suddetto rettangolo entro $\mathbb{Z} \times \mathbb{Z}$.

L'espressione `1 <= i && i <= 10 && 1 <= j && j <= i` implementa la funzione indicatrice del triangolo rettangolo-ZZ avente come vertici $\langle 1, 1\rangle$, $\langle 1, 10\rangle$ e $\langle 10, 10\rangle$, entro il piano $\mathbb{Z} \times \mathbb{Z}$.

L'espressione `(i <=4) + (j == 6) + (k != 15 && h>i*3)` fornisce il numero delle espressioni relazionali tra le coppie di parentesi tonde che risultano `true`.

B70:**f.15** Nei linguaggi C e C++ le variabili di tipo `char` consentono di operare sui caratteri ASCII, visualizzabili o meno, per leggerli, scriverli, confrontarli con altri caratteri, usarli per comporre o per analizzare parole, frasi ed espressioni artificiali (formule, codifiche, ...).
Inoltre le costanti e le variabili `char` variabili possono fornire valori interi a variabili intere e viceversa possono ricevere i loro valori da variabili e costanti intere.
Infatti gli ottetti di bits che forniscono i valori di carattere ASCII possono essere interpretati anche come rappresentazioni di valori interi, in particolare di interi dell'intervallo $\mathbf{[}0 : 127\mathbf{]}$.

La tabella che segue presenta le codifiche intere di alcuni caratteri visualizzabili.

(1)
$$\left\downarrow \begin{array}{cccccccccccccccc} \texttt{\~{}} & \cdots & \texttt{0} & \texttt{1} & \cdots & \texttt{9} & \cdots & \texttt{A} & \cdots & \texttt{Z} & \cdots & \texttt{a} & \cdots & \texttt{z} & \cdots \\ 32 & \cdots & 48 & 49 & \cdots & 57 & \cdots & 65 & \cdots & 90 & \cdots & 97 & \cdots & 122 & \cdots \end{array} \right\downarrow .$$

Una presentazione più completa si trova in `B70c03` e in `ASCII character set (we)`.

Consideriamo i seguenti esempi riguardanti le due interpretazioni dei contenuti delle variabili e delle costanti char.

```
cout >> 'a'+'b' " ;"// comporta l'emissione di 195

char carmin, carmai;
cin << carmin ;  // legge un carattere che supponiamo minuscolo
carmai = carmin - 'a' + 'A' ;  // ottiene il corrispondente carattere maiuscolo
cout >> carmai ;   // lo emette
```

B70:g. strutture di controllo selettive

B70:**g.01** Nei frammenti di programmi mC considerati finora sono intervenuti solo pochi e ben definiti elementi: dati, dichiarazioni, variabili, espressioni e semplici sequenze di assegnazioni e semplici frasi di I/O. Ora dobbiamo iniziare a esporre l'organizzazione di programmi un po' più elaborati.

Cominciamo con il precisare che con il termine **controllo del programma** intendiamo un dispositivo descrivibile un po' antropomorficamente come colui che dirige, controlla, ogni esecuzione del programma muovendosi sulle sue frasi e ordinando l'esecuzione di ciascuna frase toccata.

Per dirigere i frammenti di programmi precedenti il controllo si muoveva semplicemente in progressione. Dato che le operazioni da eseguire per risolvere ogni istanza di problema risolvibile in tempi finiti sono in numero finito, in linea di principio ciascuno di questi problemi potrebbe essere risolto con un controllo che si muove solo progressivamente.

Però per tutti i problemi da affrontare con automatismi questo richiederebbe programmi lunghissimi, anzi programmi da estendere ogni volta che si vuole risolvere una istanza di problema più esigente di quelle che possono essere affrontate con un precedente programma (dal testo finito).

Una progressiva crescita delle esigenze dei problemi che si vogliono affrontare risulta evidente in ogni società e in ogni periodo normale, cioè non funestato da qualche catastrofe.

È quindi normale che si imponga la adozione di programmi (è superfluo dire di testo finito) le cui frasi possano essere toccate più volte (anche molti miliardi di volte) dal controllo. Dunque è necessario che il controllo possa muoversi non solo progressivamente, cioè possa saltare, anche all'indietro.

B70:**g.02** Per organizzare procedure a esecuzione non sequenziale potrebbe bastare due tipi di istruzioni estremamente semplici adottate anche nei più primitivi linguaggi di macchina: l'istruzione di salto condizionato e l'istruzione di salto incondizionato. La prima è schematizzata dalla scrittura

`se(`*clausola*`) allora salta alla frase` *etichetta*

Qui *clausola* consiste in un'espressione logica, in particolare un'espressione relazionale o da una ancor più semplice variabile booleana. Essa però potrebbe anche consistere in un'espressione numerica il cui effetto equivale a quello del valore `true` se il suo valore è diverso da 0, mentre equivale a `false` nel caso opposto.

L'entità *etichetta*, *label*, rappresenta una scrittura che caratterizza una e una sola frase esecutiva. Questa la chiamiamo **frase bersaglio** dell'istruzione di salto.

Se il valore attuale della clausola è `true`, il controllo passa alla frase contrassegnata dall'etichetta indicata; in caso contrario il controllo procede a esaminare ed eseguire la frase che segue l'attuale nel testo del programma.

L'istruzione di salto incondizionato può considerarsi un caso particolare del precedente, ha la forma

`salta alla frase` *etichetta*

e semplicemente provoca il salto del controllo alla frase contraddistinta da *etichetta*.

Nel linguaggio mC queste frasi assumono le forme

`if(`*clausola*`) goto` *etichetta*

`goto` *etichetta*

Va detto anche che nel linguaggio C una etichetta è un identificatore e va posta prima della corrispondente frase bersaglio seguita da un segno "`:`". Vedremo anche che , se si programma "come si deve", nei programmi C si usano pochissime frasi bersaglio ed etichette.

B70:**g.03** Stanti le ben note prestazioni della tecnologia elettronica (ma anche elettromeccanica) è ragionevole accettare il fatto che i suddetti dispositivi di programmazione sono effettivamente realizzabili con opportuni circuiti operativi elettronici (ed anche elettromeccanici).

In effetti quando si disponeva solo dei linguaggi di macchina (dal 1945 al 1955 all'incirca) e dei primi linguaggi procedurali (il primo Fortran di Backus) avvalendosi dei suddetti dispositivi sono stati scritti parecchi efficaci programmi anche complessi.

Si può anche ricordare che prima dei computer pionieristici erano stati costruiti automatismi puramente meccanici ed elettromeccanici con notevoli prestazioni.

Un'altra idea che qui ci limitiamo a proporre, ma che verrà ampiamente approfondita, riguarda il fatto che questi dispositivi di salto consentono di organizzare 'tutte" le elaborazioni deterministiche concepibili.

Va tuttavia segnalato che l'adozione di programmi a esecuzione non sequenziale, oltre ad ampliare enormemente la portata della programmazione, ha introdotto anche il rischio di programmi che potrebbero condurre ad esecuzioni che potrebbero non concludersi con un numero finito di passi esecutivi. Questo comporta la necessità di affrontare un nuovo problema: quello del garantire che un programma non incorra nella possibilità di non arrestarsi mai. questo viene detto "problema dell'arresto di un autoatismo".

Va anche considerato che con programmi a esecuzione non solo sequenziale si giunge a utilizzare strumenti che hanno a che fare con un infinito potenziale; si vogliono controllare attività concrete e quindi che operano finitamente, ma servendosi di strumenti con prestazioni che possano estendere la loro portata sempre di più, diciamo con strumenti di portata potenzialmente infinita. Questo comporta nuovi problemi, a cominciare da quello dell'arresto. Evidentemente sarebbe stato ingenuo, se non dissennato, sperare di attuare questo importante avanzamento senza nuovi problemi.

B70:**g.04** Negli anno dal 1955 al 1965 si sono sviluppati i primi computers (allora si chiamavano calcolatori elettronici o anche, suggestivamente, ordinatori) e contemporaneamente si sono avuti, sinergeticamente, progressi tecnologici, diversificazione dei dispositivi, ampliamenti delle applicazioni, crescita della diffusione delle macchine, crescita della programmazione e crescita dell'importanza sociale del mondo dell'informatica (oltre all'aumento dei neologismi).

La crescita della programmazione ha riguardato sia il numero degli addetti, sia il numero dei programmi messi a punto, sia le metodologie della, sia le aspettative riposte in essa dal mondo amministrativo e produttivo; in questi anni la programmazione quindi ha cominciato a rivestire notevole importanza culturale.

Tuttavia negli anni 1960, in seguito all'esigenza di disporre di programmi sempre più estesi, complessi e incisivi, è emersa l'esigenza di attività di programmazione più disciplinate.

Sempre più spesso era necessario riprendere programmi preesistenti per modificarli vuoi per ampliarne la portata e le prestazioni, vuoi per aggiornare e generalizzare i loro obiettivi al fine di usarli per affrontare insiemi di istanze più eterogenei, vuoi per aumentare la precisione e la attendibilità dei risultati numerici.
A questo punto, poco dopo l'introduzione del termine "software", si è iniziato a prendere in considerazione un intero "ciclo di vita" per il software stesso.

Mentre in precedenza i pregi richiesti ai programmi si limitavano alla attendibilità, alla velocità ed alla precisione dei risultati numerici, sono venuti ad assumere importanza crescente altri pregi: adattabilità, versatilità, estendibilità, esauriente documentazione e quindi leggibilità dei relativi testi sorgente.

B70:**g.05** Si è allora cominciato a osservare che i programmatori erano pochi, che erano portati a seguire abitudini personali e spesso estemporanee, che poco si preoccupavano della documentazione dei programmi e della leggibilità dei loro sorgenti e che poco discutevano delle scelte di programmazione con colleghi che avrebbero potuto riprendere i loro programmi.

Più specificamente si è osservato che i programmi nei quali comparivano numerosi `goto` spesso risultavano difficilmente comprensibili anche dagli stessi autori che dovevano riprenderli qualche tempo dopo la loro prima utilizzazione; inoltre la loro correttezza era onerosa da controllare.

In effetti accadeva spesso di dover modificare e ampliare affannosamente le prestazioni di programmi in uso con l'aggiunta o l'adattamento di porzioni servendosi di gruppi di `goto`.

In genere questi `goto` avevano bersagli in posizioni difficili da individuare e i loro effetti complessivi non venivano analizzati con sufficiente completezza. Inoltre quando venivano attuate successive modifiche il controllo e la strategia delle logiche delle elaborazioni procedevano a deteriorarsi.

B70:**g.06** In quel periodo, in conseguenza delle critiche formulate da vari teorici della programmazione, in particolare da Edsger Dijkstra nel 1968, e grazie a un teorema formulato nel 1966 da Boehm e Jacopini sulla possibilità di evitare istruzioni equivalenti al `goto` nelle macchine di Turing, si è imposta l'opportunità di evitare le istruzioni `goto` rimpiazzandole con le strutture di selezione e con le strutture di iterazione che vedremo tra breve.

Si sono dunque imposte strutture di controllo e modalità per la loro organizzazione di una certa rigidità ma che, se adottate come criteri per una disciplina della programmazione, portano alla disponibilità di programmi più leggibili, più controllabili, piĕstendibili e più riutilizzabili nella prospettiva di attività di programmazione sempre più sistematiche, con obiettivi sempre più ampi e in progressiva evoluzione.

Questo modo di organizzare la programmazione è stato chiamato **programmazione strutturata** e in seguito lo seguiremo diligentemente. Introdurremo dunque le strutture di controllo che la facilitano e trascureremo le istruzioni `goto` (a questo proposito si parla di *goto-less programming*, considerando che gli enunciati `goto` solo in rare situazioni consentono soluzioni chiare e poco rischiose.

B70:**g.07** Nel corso di un'elaborazione può accadere di dover scegliere di affrontare diverse azioni in dipendenza dei valori attuali di alcuni parametri variabili. In queste circostanze si dice che si devono organizzare **selezioni** tra diverse azioni.
La situazione più semplice riguarda la possibilità, nell'ambito di una sequenza di azioni, di effettuare o meno una manovra più o meno complessa. Con i dispositivi del salto incondizionato e condizionato questa selezione si organizza con frammenti di programma schematizzabili come segue.

azioni precedenti
`if(`*clausola*`) goto Lsegue;`
azioni da eseguire sse non vale *clausola*
`Lsegue :` azioni successive

Leggermente più elaborata è l'organizzazione della scelta tra due manovre alternative

azioni precedenti
`if(`*clausola*`) goto Laltern;`
azioni da eseguire sse non vale *clausola*
`goto Lsegue;`
`Laltern:`
azioni da eseguire sse vale *clausola*

`Lsegue :` *azioni successive*

Si possono inoltre prevedere selezioni tra tre o più possibilità trattabili con costrutti che estendono il precedente.

B70:**g.08** Nell'ambito della programmazione strutturata si adottano invece, risp., i seguenti costrutti

azioni precedenti
`if(`*clausola*`) {`
  azioni da eseguire sse vale clausola `}`
azioni successive

azioni precedenti
`if(` *clausola*`) {`
  azioni da eseguire sse vale clausola `}`
`else {`
  azioni da eseguire sse non vale clausola `}`
azioni successive

In questi costrutti si individuano chiaramente i cosiddetti **blocchi di istruzioni** gruppi di frasi consecutive delimitati da parentesi graffe coniugate, ciascuno dei quali chiaramente condizionato da una clausola che lo precede.

B70:**g.09** La nozione di blocco è centrale nella programmazione strutturata e che si possono avere sia blocchi semplici costituiti da una sola frase esecutiva o da una sequenza di queste frasi, sia blocchi elaborati contenenti altri blocchi dipendenti dai costrutti selettivi che stiamo descrivendo, dipendenti dai costrutti iterativi descritti nella sezione che segue e contenenti richiami a functions.

Accenniamo anche al fatto che i blocchi possono contenere anche dichiarazioni (ed inizializzazioni) di variabili che hanno uno **scope**, cioè un ambito di visibilità e validità, limitato al blocco stesso.

Va detto anche che per un blocco ridotto a una sola frase le parentesi che lo delimitano possono essere tralasciate, in modo da alleggerire il testo sorgente.

Nei precedenti costrutti e in gran parte di quelli che stiamo per introdurre viene meno la necessità di introdurre etichette per le frasi alle quali rinviavano le frasi `goto`. La stesura nel testo sorgente dei costrutti strutturati è opportuno sia realizzata seguendo sistematicamente dei criteri di collocazione delle parole chiave e delle parentesi graffe.

Adottando diligentemente questi accorgimenti si possono avere programmi di buona leggibilità e che risultano più facili da progettare, da redigere, da adattare al mutare delle esigenze, evento che in molte attività si verifica di frequente. Conviene sottolineare che le necessità di aggiornare i programmi, soprattutto quelli di grandi dimensioni, nel tempo sono andate crescendo e che molti problemi legati alle attività concrete richiedono programmi sviluppati da folti gruppi di programmatori e con aggiornamenti preventivati.

Infine va aggiunto che i testi ben strutturati si possono presentare abbastanza agevolmente mediante diagrammi di flusso o mediante altre tecniche di visualizzazione.

B70:**g.10** Nel corso di una elaborazione accade spesso che al controllo si pone la scelta tra la successiva esecuzione di diverse manovre alternative. Per esempio si presenti una prima manovra da eseguire sse si verifica una *clausola 1*, una seconda da effettuare sse non si verifica *clausola 1* ma si verifica una *clausola 2* e una terza da eseguire sse non si verifica nessuna del due clausole precedenti. Il meccanismo

per questa scelta viene implementato dal costrutto schematizzato come segue (negli schemi che seguono trascuriamo di indicare azioni precedenti e successive).

`if(`*clausola 1*`) {`
azioni da eseguire sse vale *clausola 1* `}`
`else if(`*clausola 2*`) {`
azioni da eseguire sse non vale *clausola 1* ma vale *clausola 2* `}`
`else {`
azioni da eseguire sse non valgono né *clausola 2* né *clausola 2* `}`

Soluzioni analoghe si adottano per 2, 4 o piǒssibili scelte.

Per queste selezioni si individua l'insieme delle situazioni sulle quali scegliere e lo si ripartisce in parti disgiunte, l'ultima delle quali essendo la complementare delle precedenti ed essendo preceduta dalla parola chiave `else`, altrimenti.

Si possono anche organizzare costrutti privi di azioni concernenti l'insieme complementare delle situazioni che soddisfano clausole esplicite. Gli schemi di questi costrutti riguardanti 2 e 3 possibilità (in B70g04 è stato presentato quello con una unica possibilità) sono i seguenti:

`if(`*clausola 1*`) {`
azioni da eseguire sse vale *clausola 1* `}`
`else if(`*clausola 2*`) {`
azioni da eseguire sse non vale clausola 1 e vale *clausola 2* `}`

`if(`*clausola 1*`) {`
azioni da eseguire sse vale *clausola 1* `}`
`else if(`*clausola 2*`) {`
azioni da eseguire sse non vale clausola 1 e vale *clausola 2* `}`
`else if(`*clausola 3*`) {`
azioni da eseguire sse non vale *clausola 1*, non vale *clausola 2*, ma vale *clausola 3* `}`

B70:**g.11** Presentiamo alcuni esempi, come al solito sotto forma di frammenti di programma.

```
// Dal progressivo del mese il numero dei suoi giorni in un anno non bisestile
if(xmese == 2) ngiorni = 28 ;
else if(xmese==4 || xmese==6 || xmese==9 || xmese==11) ngiorni = 30 ;
else ngiorni = 31;
```

```
// Segnalazioni conseguenti al voto vps ottenuto
in una prova universitaria scritta
if(vps<=12) {cout << "ripetere prova scritta" << endl ; }
else if(vps<18) {
   cout << "consigliato ripetere prova scritta" << endl ;
   cout << "il superamento prova orale porta al massimo 24/30" << endl ; }
else if(vps<24) {
   cout << "presentarsi alla prova orale << endl ; }
else {cout << "la prova scritta comporterebbe il voto 25/30";
cout << "per migliorare voto presentarsi alla prova orale" << endl ; }
```

B70:**g.12** Come si è detto, in un blocco selettivo possono essere inseriti altri costrutti di selezione e in questi altri ancora; in altre parole si possono programmare selezioni a più livelli.

Un primo esempio è dato dalla generalizzazione di una selezione in B70g11 la quale non vale per gli anni bisestili.

```
    // Dai progressivi dell'anno e del mese ricavare il numero dei giorni del mese
    if(xmese == 2) {
       if (xanno%400==0) ngiorni = 29 ;
       else if(xanno%100==0) ngiorni=28 ;
       else if(xanno%4==0) ngiorni=29 ;
       else ngiorni=28 ;
       }
else if(xmese==4 || xmese==6 || xmese==9 || xmese==11) ngiorni = 30 ;
    else ngiorni = 31;
```

Un esempio ancor più chiaramente definito di selezione a due livelli riguarda la assegnazione di una coordinata $\langle x, y\rangle$ del piano sugli interi ad una di 9 parti di questo insieme.

```
    if(x<0) {
       if(y<0) cout << "III quadrante" << endl ;
       else if(y==0) cout << "semiasse orizzontale negativo" << endl ;
       else cout << "II quadrante" << endl ;
       }
    else if(x==0) {
       if(y<0) cout << "semiasse verticale negativo" << endl ;
       else if(y==0) cout << "origine" << endl ;
       else cout << "semiasse verticale positivo" << endl ;
       }
    else {
       if(y<0) cout << "IV quadrante" << endl ;
       else if(y==0) cout << "semiasse orizzontale positivo" << endl ;
       else cout << "I quadrante" << endl ;
       }
```

B70:**g.13** **(1) Eserc.** Redigere un frammento di programma nel quale si controlla che vengano immessi tre numeri interi e si decide se il secondo esprime la posizione sulla retta dei numeri interi di un punto compreso tra i punti aventi come ascisse il primo e il terzo numero.

(2) Eserc. Si chiede un frammento di programma nel quale si decide se tre numeri dati possono esprimere le lunghezze dei lati di un triangolo, distinguendo il caso di tre punti allineati.

(3) Eserc. Si rediga un frammento di programma nel quale si pongono in ordine crescente prima due numeri interi letti e successivamente tre interi.

(4) Eserc. Si scriva un frammento di programma nel quale si pongono in ordine alfabetico tre sigle automobilistiche, o più in generale tre stringhe di due lettere.

(5) Eserc. Redigere un frammento di programma nel quale si esaminano due sigle di tre lettere registrate, risp., nell'array `a[0:2]` e in `b[0:2]` per stabilire se la prima precede la seconda, oppure se coincidono, oppure se la seconda precede la prima.

B70:h. strutture di controllo iterative

B70:**h.01** Consideriamo il problema di sommare quattro numeri interi; evidentemente lo si può risolvere con un programma come il seguente:

```
#include <iostream.h>
int main() {
int h,k,m,n,sum;
cin >> h >> k >> m >> n ;
sum = h ;
sum = sum + k ;
sum = sum + m ;
sum = sum + n ;
cout << sum << endl;
return 0 ;
}
```

Un tale programma può dirsi "meramente sequenziale" e in buona sostanza imita il calcolo che si può effettuare manualmente con una semplice calcolatrice numerica meccanica, elettromeccanica o elettronica (v.a. Schickard, Pascal, Leibniz, Thomas de Colmar, ...).

B70:**h.02** Se si devono sommare 100 o 1000 numeri questo modo di fare richiede un programma molto lungo, assurdamente prolisso. Inoltre la somma di un numero degli addendi non predeterminato programmata nel precedente modo puramente sequenziale dovrebbe contenere anche frasi `if` per il controllo della conclusione della somma attualmente richiesta e sarebbe ancor meno ragionevole.

Per avere programmi gestibili si impongono due nuovi modi di fare. Innanzi tutto è necessario servirsi di gruppi di frasi, che nei casi più semplici si riducono a frasi singole, che nel corso di una esecuzione del programma possano essere eseguite più volte, ossia possano essere toccate dal controllo più volte.

Queste gruppi di frasi, come quelli che sono oggetti di scelta nei costrutti selettivi, sono detti blocchi di frasi e sono caratterizzati dall'essere delimitati da coppie di parentesi graffe coniugate; questi delimitatori però possono essere evitati per i blocchi costituiti da una singola frase.

Questi blocchi più specificamente li chiamiamo **blocchi iterativi**; Chiamiamo poi **ciclo [esecutivo]** ogni loro esecuzione.

Nei blocchi iterativi deve essere possibile richiedere azioni diverse nelle loro diverse esecuzioni, cioè nei diversi cicli iterativi. In altre parole devono avere una sufficiente versatilità e non possono essere pedissequamente ripetitive, ma devono contenere elementi variabili nei diversi cicli.

Per questa versatilità si possono adottare vari dispositivi.

Innanzi tutto in esse devono comparire operandi che cambiano nelle successive esecuzioni. Un primo modo per ottenere questo richiede di porre nel blocco operazioni di lettura che forniscono nuovi dati a ogni nuova esecuzione del blocco stesso; ciascuna esecuzione viene detta **esecuzione ciclica** o **ciclo [esecutivo]** del blocco; talora si usa il termine azione da reiterare.

Un secondo dispositivo consiste nel servirsi di sequenze di gruppi di dati che possono essere: componenti di arrays disponibili prima del blocco, dati forniti da frasi di lettura appartenenti nel blocco risultati di elaborazioni interne al blocco, risultati di richiami di functions che dipendono da parametri fatti variare opportunamente da stadio a stadio.

Tra le elaborazioni interne al blocco conviene segnalare le conseguenze di costrutti selettivi interni al blocco che dipendono da qualche dato variabile e quindi nei diversi cicli consentono di scegliere comportamenti diversi, anche molto.

B70:**h.03** Si impone quindi la necessità di poter inserire in un programma dei **costrutti iterativi** o **iterazioni**, cioè dei complessi di istruzioni costituiti da un blocco iterativo e da parole chiave ed espressioni con il compito di controllare il complesso dei suoi cicli esecutivi.

I blocchi iterativi strutturalmente più semplici sono costituiti da sequenze di una o più frasi esecutive. si possono però organizzare blocchi iterativi molto articolati contenenti altri costrutti iterativi e altri costrutti selettivi i quali a loro volta possono essere articolati quanto si vuole.

Nel linguaggio mC, come in tutti i linguaggi evoluti, si possono organizzare svariati tipi di costrutti iterativi seguendo regole sintattiche piuttosto diverse.
Una prima distinzione riguarda: (I) iterazioni la cui sequenza di cicli risulta definita prima dell'inizio della sua esecuzione e dipende relativamente poco dalle circostanze delle diverse esecuzioni; (II) iterazioni la cui sequenza di cicli viene a definirsi nel corso dell'esecuzione delle manovre stesse e dipende in modo determinante dalle circostanze delle singole esecuzioni.

Le prime vengono rette da un indice che corre su una sequenza di valori predefinita. La corsa dell'indice tipicamente viene individuata dall'assegnazione all'indice di un valore iniziale, da un incremento (o un decremento) da eseguire prima di un eventuale nuova manovra ciclica e da una relazione che stabilisce se si avrà una manovra ciclica successiva o se viceversa l'iterazione è conclusa.

Le iterazioni (II) vengono governate da una condizione che può dipendere da vari parametri i quali possono cambiare nel corso dell'esecuzione di ogni nuovo ciclo e che determina se si deve proseguire la manovra corrente, oppure interromperla per passare alla (eventuale) successiva, oppure concludere senza ulteriori controlli l'intera iterazione.

Un'altra classificazione delle iterazioni distingue quelle rette da una clausola esaminata prima di eseguire una eventuale nuova manovra ciclica, quelle governate da una clausola esaminata alla fine di ogni manovra e quelle rette da una clausola esaminata a un certo punto dell'esecuzione delle manovre da reiterare.

B70:**h.04** Una semplice tipica iterazione del tipo (I) si trova nel frammento seguente, che si suppone preceduto dalla costruzione dell'array `val`.

```
// Sommare gli interi forniti dalle prime 10 componenti dell'array val[]
int somma=0;
for (int i=0; i<10; i++) somma = somma + val[i] ;
```

Qui abbiamo un costrutto f̂or con un blocco iterativo ridotto ad una semplice frase di accumulo, un indice di reiterazione, `i`, definito nell'argomento della parola chiave `for` il quale corre dal valore iniziale 0 fino al valore finale 9 (l'intero che precede 10) con incrementi di 1 a ogni nuovo ciclo, in seguito alla richiesta di incremento `i++`.

L'indice di un costrutto `for` potrebbe anche subire decrementi come nel frammento seguente finalizzato all'emissione dei mesi di un anno procedendo all'indietro, tornando al passato.

```
for(int mese=12; mese>0;mese--)
    cout << "mese numero " << mese << val[12mese] << endl ;
```

L'indice di un `for` a ogni nuova manovra da reiterare potrebbe subire variazioni diverse dall'aumento o dalla diminuzione di 1 come accade in questo frammento che riguarda l'essere bisestili o meno gli anni dal 2000 al 2099.

```
for (int anno=2000; anno<2100; anno+=4) {
    ngior[anno-2000] = 366;
    ngior[anno-1999] = ngior[anno-1998] = ngior[anno-1997] = 366 ;
    }
```

L'indice di un `for` potrebbe comunque essere modificato da altre frasi che fanno parte del blocco iterativo; una situazione di questo genere tuttavia non è da incoraggiare in quanto di comprensione non immediata e quindi portatrice di possibili rischi.

Lo svolgersi delle successive manovre cicliche di una iterazione (I), e in particolare delle iterazioni organizzate dai più semplici costrutti `for`, talora può essere conveniente presentarle come visite di una sequenza di posizioni visualizzabili; questa sequenza potrebbe essere un intervallo numerico, una progressione aritmetica, una progressione geometrica o la successione dei valori contenuti nei componenti di un qualche array, o anche la successione dei valori assunti da una qualche function avente dominio discreto monodimensionale.

B70:**h.05** In generale un costrutto `for` presenta una assegnazione iniziale ad una variabile intera (ma non solo) che assume il ruolo di **indice del costrutto**, una clausola da testare prima di effettuare un primo o nuovo ciclo e una richiesta di modifica da effettuare dopo ogni esecuzione di ciclo.

L'assegnazione iniziale può contenere anche la dichiarazione dell'indice che avrà visibilità limitata al costrutto; viceversa essa può mancare: questo accade quando si è provveduto a una inizializzazione dell'indice prima del costrutto `for` oppure in un costrutto `for` che non si serve di un indice nel modo che vedremo.

La clausola che condiziona la possibilità di effettuare un nuovo ciclo spesso riguarda il confronto dell'indice con un parametro che viene modificato a ogni nuovo ciclo, ma può anche riguardare altri parametri nessuno dei quali si propone con il ruolo di indice (unico) dell'iterazione. Questa clausola può essere molto complessa e/o essere incapsulata in una function [B70e]; in quest'ultimo caso la clausola può non essere agevolmente rilevabile dal testo del programma e questo comporta un certo rischio di poca leggibilità.

A loro volta le azioni che vengono eseguite dopo l'esecuzione di un ciclo possono mancare, oppure consistere in semplici incrementi o decrementi, oppure essere rette da enunciati complessi per esempio venire incapsulate in una function.

B70:**h.06** Presentiamo alcuni frammenti di costrutti `for`.

(1)
```
// dato un array int d[100] individuare i suoi valori massimo e minimo
dMIN = dMAX = d[0] ;
for(int i = 1 ; i++ ; i<100) {
    if(d[i] < dMIN) dMIN = d[i] ;
    if(d[i] < dMAX) dMAX = d[i] ;
    }
```

(2)
```
// Dato un array int d[100] e due soglie dTHMI e dTHMA,
// porre in dacc[<daccL] la sequenza dei suoi valori compresi tra le soglie
// e porre in daaccp[<daccL] la sequenza delle rispettive posizioni
daccL=0;
for(int i = 1 ; i++ ; i<100) {
if(dTHM <= d[i] &&d[i] <= dTHMA) {
    dacc[daccL] = d[i]; daccp[daccL++] = i;
```

```
    }
```

B70:**h.07** (1)
```
// Dato un array int d[<dL] di interi che forniscono delle percentuali,
// costruire l'array decil[<10] dei numeri di percentuali
// che cadono nei 10 intervalli di decili
int id,
for(id=0 ; id++ ; id <10) decil[id] = 0;
for(int i = 1 ; i++ ; i<dL) {
    datt=d[i] ;
    for(id=0 ; id++ ; id <10) {
        if(datt < 10*id) {
            decil[id]++; break;
            }
        }
    }
```

In questo frammento compare la frase `break;` avente questa unica forma. Essa ha l'effetto di trasferire il controllo alla prima frase esecutiva che segue la parentesi graffa che conclude il blocco della struttura iterativa nella quale si trova.
Questa frase può trovarsi anche nelle altri strutture iterativi che vedremo nelle prossime sezioni caratterizzate dalle parole chiave `while`, e `do` e nella struttura selettiva caratterizzata da `switch`.

B70:**h.08** (1)
```
// Lettura di un array monodimensionale di interi
    }
```

(2)
```
// Lettura di una stringa
    }
```

(3)
```
Lettura di un elenco di nomi
    }
```

B70:**h.09** 1 // Scrittura

B70:**h.10** **(1) Eserc.** Scrivere un frammento di programma che genera le prime 30 potenze di 2.

(2) Eserc. Scrivere un frammento di programma che genera i primi 10 numeri fattoriali a partire dalla loro definizione ricorsiva.

(3) Eserc. Scrivere un frammento di programma che genera le prime componenti della `successione di Fibonacci (wi)` a partire da una sua definizione costruttiva.

B70:**h.11** Il comando `while` si può considerare una semplificazione del comando `for` in quanto come argomento presenta solo la clausola per la esecuzione di una nuova manovra ciclica. Esso presenta la forma seguente

azioni di inizializzazione
`while(`*clausola di reiterazione*`) {`
blocco iterativo `}`

In alcune esecuzioni dell'iterazione potrebbe non essere eseguita alcuna manovra ciclica: questo accade sse preliminarmente alla esecuzione della prima manovra l'espressione condizionale vale false.

B70:**h.12** Anche il costrutto `do - while` si può considerare una semplificazione del comando `for`, ovvero come una variante del comando `while`. Esso presenta la forma seguente leggermente più articolata

azioni di inizializzazione
`do {`
`    `*blocco iterativo* `}`
`while(`*clausola di reiterazione*`) ;`

Esso provoca l'esecuzione di una prima manovra ciclica e successivamente, come dopo l'eventuale esecuzione di ogni nuova manovra ciclica, la valutazione della clausola di reiterazione per stabilire se si deve eseguire una manovra ciclica successiva.

esempi

B70:**h.13** Consideriamo ora i due comandi `continue` e `break` che possono essere utilizzati all'interno dei blocchi iterativi per precisare i loro effetti.
Il comando `continue` compare successivamente a un comando selettivo `if`, `else if` o `else` oppure come comando conclusivo di un blocco subordinato a un comando selettivo. La sua esecuzione comporta che il controllo salti alla conclusione della manovra ciclica attuale e quindi alla valutazione della clausola di reiterazione, evitando tutte le manovre intermedie. Un frammento schematico che mostra il suo effetto è il seguente.

```
bool completo = false;
while(!completo) {
   bool finissaggio=true;
   azioni che possono modificare finissaggio e completo
   if(!finissaggio) continue;
   azioni di finissaggio su quanto prodotto nella manovra ciclica
   }
```

B70:**h.14** Anche il comando `break` può comparire come comando conclusivo di un blocco subordinato a un comando selettivo all'interno di un blocco iterativo; esso potrebbe anche trovarsi, come vedremo in B70d18, in relazione a blocchi `case` in costrutti `switch`. La sua esecuzione comporta l'interruzione dell'esecuzione del blocco iterativo; esso quindi va invocato quando si verificano condizioni che comportano questa interruzione. Tipicamente tale comando compare in una posizione intermedia del blocco iterativo in modo da consentire l'esecuzione di una prima parte di una manovra ciclica che sarà l'ultima, ma non la seconda parte.
L'effetto di conclusione della iterazione di un `break` può accompagnare l'effetto di una clausola di reiterazione, oppure essere efficace per una iterazione priva di una sua clausola esplicita. Un frammento schematico che mostra il suo effetto nella prima situazione di `break` in presenza di clausola reiterativa è il seguente.

```
bool completo = false;
while(!completo) {
   bool stopiteraz=false;
   azioni che possono modificare stopiteraz e completo
   if(stopiteraz) break;
   azioni finali della manovra ciclica
```

```
}
```

(1) Eserc. Precisare un frammento di programma che inizia con il commento: `// In seq[0..49] si trova una sequenza crescente di interi;`
`// individuare la posizione del massimo valore inferiore a 100.`

B70:**h.15** Un'altra situazione che vede una iterazione contenente un `break` e mancante di clausola reiterativa può vedersi come possibilità di servirsi di costrutti iterativi che in apparenza avviano una iterazione illimitata. Un frammento schematico per questa situazione è il seguente.

```
while(true) {
   bool stopiteraz=false;
   azioni che possono modificare stopiteraz
   if(stopiteraz) break;
   azioni finali della manovra ciclica
   }
```

L'argomento del `while` è sempre vero e si procede a successive esecuzioni della manovra ciclica fino a che si verificano le condizioni che implicano una esecuzione del comando `break`. Va osservato che la logica del blocco iterativo deve essere studiata attentamente per garantire che in tempi ragionevoli si abbia effettivamente l'esecuzione di un `break`. In caso contrario si avrebbe una ripetizione illimitata della manovra ciclica e il computer resterebbe illimitatamente silenzioso e inutilizzabile. In questa situazione si usa dire che "il computer è in loop" e la cosa è del tutto negativa; l'esecuzione va interrotta e il programma modificato e questo è gravemente negativo se risulta difficile capire quando si deve arrestare la macchina e quali sono le correzioni da apportare al programma.

B70:**h.16** Vediamo ora il costrutto `switch`, costrutto selettivo che consente di organizzare scelte tra svariate manovre in dipendenza dei valori assunti da una variabile sugli interi o sui caratteri.

Esso si presenta come terna costituita dalla parola chiave `switch`, da un argomento consistente di una variabile o i un'altra espressione valutabile e da un blocco di istruzioni che si articola in una sequenza di sottoblocchi; ciascuno di questi inizia con una o più coppie costituite dalla parola chiave `case` e da un valore presentato come possibile valore attuale della variabile di switch.

Nelle soluzioni più semplice da leggere ciascuno dei successivi sottoblocchi esprime azioni da eseguire se per la variabile di switch si trova uno dei valori presentati all'inizio e si conclude con un comando `break` che implica l'uscita dal costrutto switch. L'ultimo dei sottoblocchi di un costrutto può iniziare con la semplice parola chiave `default` collocata prima della formulazione delle azioni da eseguire quando il valore della variabile di switch è risultato diverso da tutti quelli previsti per i sottoblocchi precedenti.

Vediamo un esempio piuttosto autoesplicativo concernente i 5 `solidi platonici (wi)`

```
// Precisazione dei dati caratteristici dei solidi platonici
switch(numvertici) {
  case 4 :  {    // tetraedro
    numspig = 6; numfacce = 4; n1Schl = 3; n2Schl = 3; break; }
  case 6 :  {    // ottaedro
    numspig = 12; numfacce = 8; n1Schl = 3; n2Schl = 4; break; }
  case 8 :  {    // cubo
    numspig = 12; numfacce = 6; n1Schl = 4; n2Schl = 3; break; }
  case 12 :  {    // icosaedro
```

```
  numspig = 30; numfacce = 20; n1Schl = 3; n2Schl = 5; break; }
case 20 :  {    // dodecaedro
  numspig = 30; numfacce = 12; n1Schl = 5; n2Schl = 3; break; }
}
```

L'ultimo sottoblocco può mancare del `break` finale senza che si abbiano differenze.
In un costrutto `switch` si possono avere sottoblocchi diversi dall'ultimo mancanti del `break` finale; in un tal caso il controllo, dopo l'esecuzione del sottoblocco, invece di passare al comando che segue il blocco switch, "percola" nel sottoblocco successivo. Se più sottoblocchi mancano del `break` finale i possibili flussi del controllo possono essere un po' complicati da seguire; questa possibilità consente però di risparmiare molte ripetizioni di comandi.

B70:**h.17** Negli schemi dei costrutti precedentemente introdotti si fa uso sistematico di blocchi di programma esprimenti complessi di azioni.
In un programma che vuole risolvere un problema ben definito attraverso un procedimento ben organizzato possiamo aspettarci di incontrare blocchi ciascuno dei quali con una finalità operativa chiaramente presentabile agli operatori che hanno il compito di controllare la qualità del programma stesso. Questi controlli di qualità possono riguardare diverse scelte per i molteplici parametri valutativi ai quali si possono dare diversi pesi; in effetti questi parametri possono riguardare aspetti quali adeguatezza, correttezza, efficienza esecutiva, facilità d'uso, versatilità, riutilizzabilità e altro ancora.

I blocchi che si possono isolare in un programma possono avere finalità e taglie molto diverse, dalle più minute, alle più elaborate.
I blocchi più minuti sono costituiti da un solo enunciato da eseguire o da una sola espressione da valutare; i blocchi di un solo enunciato, contrariamente ai più elaborati, non è necessario delimitarli con una coppia di parentesi graffe.
Altri blocchi sono costituiti da sequenze di enunciati e da sottoblocchi, cioè da blocchi interamente contenuti nel blocco dal quale dipendono; di un sottoblocco si dice che è interamente annidato nel blocco dal quale dipende.
Vi sono poi blocchi costituiti da uno dei costrutti selettivi o iterativi visti in precedenza, costrutti nei quali si individuano dei sottoblocchi.

Analizzando un programma ben strutturato in termini di blocchi e sottoblocchi dipendenti si giunge a individuare per il testo del programma una struttura chiamata arborescenza distesa e descritta in D30. In una tale struttura si possono avere blocchi con livelli di dipendenza anche molto diversi.

B70:**h.18** La programmazione strutturata consente di esprimere tutte le procedure pensabili. Qui non cerchiamo di dimostrare questa affermazione, ma ci limitiamo a proporre intuitivamente questa "illimitata" potenzialità della programmazione strutturata.
Trovandoci di fronte a un problema da affrontare con una procedura può accadere di individuare un procedimento che richiede di effettuare una dopo l'altra una sequenza di manovre ciascuna delle quali risolve un sottoproblema corrispondente a una porzione del problema complessivo.
Alternativamente si può individuare un insieme di possibilità per i dati che possano essere distinte mediante opportune operazioni e che ciascuna possibilità possa considerarsi un sottoproblema di quello originario.
In una terza alternativa si può individuare una manovra da eseguire con varianti controllabili algoritmicamente e da eseguire secondo una successione determinata, (in particolare in dipendenza di una variabile che corre in modo controllabile) e ciascuna delle varianti da affrontare possa considerarsi un sottoproblema di quello posto all'inizio.

Si hanno quindi tre modi di ridurre un problema a sottoproblemi e riesce difficile concepire modi diversi da questi per ridurre un problema dato a sottoproblemi.

I precedenti modi per ridurre un problema portano a delineare un programma principale che possa presentarsi come una bozza di programma che si riduce a una struttura sequenziale, selettiva o iterativa di blocchi che andranno singolarmente analizzati e trasformati in strutture da trattare più avanti.
In tal modo si può proseguire fini a che, o si hanno solo blocchi facilmente esprimibili, o si incontrano sottoproblemi che non si sanno esprimere in termini di soluzione operativa. Il secondo caso, se si era proceduto correttamente, porta a concludere che non è stato proposto un problema effettivamente risolubile. Nel primo caso si ottiene un programma ben strutturato in grado di risolvere il problema dato.

B70:i. programmi su interi e stringhe [1]

B70:**i.01** Il seguente programma calcola un coefficiente binomiale.

B70:**i.02** Programma per la fattorizzazione di un intero positivo.

B70:**i.03** Calcolo del minimo comune multiplo e del massimo comun denominatore.

B70:**i.04** Elenco di numeri primi.

B70:**i.05** Precisazione della posizione settimanale di un giorno

B70:**i.06** Taglio sillabico di parola italiana.

B70:**i.07** Conversioni di una data.

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