1

Capitolo B70
linguaggio di programmazione c++ [1]

Contenuti delle sezioni

57 pagine

B700.01 Questo è il primo capitolo dedicato al linguaggio di programmazione procedurale che identifichiamo con `c++`, sigla per C++ ridotto.

Esso consiste in un sottoinsieme piuttosto circoscritto del `linguaggio C++ (wi)` che intende costituire uno strumento di programmazione utilizzabile concretamente e piuttosto facilmente con qualcuno dei molti sistemi software per lo sviluppo del linguaggio C++ attualmente disponibili.

La presenza di questo linguaggio nella *esposizione* ha tre motivazioni.

La prima è pratica; esso consente di implementare concretamente algoritmi in grado di sostenere la comprensione di svariate nozioni della matematica discreta e della analisi numerica; alcune di queste implementazioni sono già nella *esposizione*.

La seconda è dovuta al fatto che si può vedere come corrispondente limitato ma effettivamente sperimentabile del linguaggio di una macchina sequenziale programmabile, ossia di un genere di strumenti che qui abbiamo introdotto alla base della computabilità, cioè della nozione che qui è stata trattata per fornire motivazione iniziale della matematica.

Inoltre questo linguaggio è sufficiente per concretizzare le procedure per sostenere la redazione di questa stessa *esposizione* che sono presentate in 05 e la sua descrizione può contribuire alla comprensione delle motivazioni e delle caratteristiche di tali procedure.

B700.02 Conviene ricordare che C++ è una evoluzione del linguaggio C che amplia tutta la sua portata, ma presenta alcune differenze sintattiche rese opportune dal fatto che si serve di librerie standard molto più ampie e dall'essere in progressiva evoluzione.

Il linguaggio c++ denota un sottoinsieme ridotto del linguaggio C++ che grosso modo, è un sottoinsieme del linguaggio C; più precisamente la sua portata è contenuta nella portata di C, ma adotta alcune regole formali di C++ assenti dal linguaggio C.

Consideriamo c++ un sottoinsieme di C++ invece che sottoinsieme del più vicino linguaggio C per vari motivi pratici.

C++ è più sicuro grazie alla più forte tipicizzazione; è dotato di librerie più ricche e in continua evoluzione; rispetto a C il linguaggio C++ è più vicino ai problemi e meno legato alle esigenze della macchina, ma consente ancora di essere consapevole dei dettagli delle operazioni sulle quali si basano le esecuzioni e che sono in stretta corrispondenza con i dettagli dei ruoli delle entità elementari, delle operazioni e delle strutture della matematica discreta presentati nei primi capitoli del tomo B.

C++ consente anche commenti più elastici da collocare a destra della coppia di 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 i meccanismi delle classi di oggetti apre la possibilità della programmazione per oggetti, tecnica impegnativa e non cotemplata dal c++, anche se possiede notevoli potenzialità per la implementazione di strutture matematiche e in particolare per elaborazioni su di figure geometriche.

B700.03 In questo capitolo c++ 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-fp**, cioè i numeri rappresentabili con le codifiche del genere **floating point** esprimibili con 32, 64, 128 o 256 bits; queste informazioni nell'ambito della programmazione vengono chiamati "real numbers".

Le pagine che seguono vogliono anche introdurre i primi concetti sulla programmazione e sulle prestazioni del 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 fattivamente l'esecuzione di calcoli concernenti costruzioni matematiche riguardanti configurazioni discrete.

La possibilità di sperimentare, oltre ad avere evidente valenza pratica, può essere un robusto supporto alla comprensione di molte nozioni matematiche, in particolare negli ambiti delle strutture discrete, dei meccanismi inferenziali, dei calcoli approssimati e dei procedimenti per le applicazioni della disciplina.

Dopo questa introduzione e alcuni paragrafi che seguono, mediante piccoli programmi o mediante frammenti di programma, i cosiddetti **snippets**, risulterà possibile presentare le implementazioni di molti dei procedimenti costruttivi che costituiscono componenti dignificativi della *esposizione*.

B70 a. programmazione dei computers, generalità

B70a.01 La programmazione degli elaboratori elettronici è oggi una delle attività più importanti da tanti punti di vista, fatto evidenziato dalla onnipresenza di questi strumenti.

L'elaborazione automatica ha suscitato interessi economici, culturali e politici che toccano, più o meno direttamente, tutti gli aspetti della vita dei contemporanei.

L'efficienzae l'efficacia nelle attività di programmazione condizionano l'efficienza e la efficacia di interi settori amministrativi, commerciali e industriali e risultano determinanti per la capacità di interi sistemi paese di riuscire ad evolversi nell'odierno panorama di cambiamenti continui e generali.

B70a.02 Esaminiamo brevemente gli aspetti del mondo della programmazione.

I dispositivi per il calcolo elettronico più importanti, in breve i computers, hanno la possibilità di adattarsi ai cambiamenti degli ambienti nei quali sono inseriti prendendo delle decisioni a partire dagli stimoli che giungono dal loro intorno o da ordini che provengono da un loro controllore presente o remoto.

Queste decisioni sono effettuate mediante elaborazioni che dall'esterno sono viste come trasformazioni di informazioni di ingresso in informazioni di uscita, trasformazioni effettuate da operazioni realizzate da loro circuiti organizzate da programmi per i computers predisposti al loro interno.

Una elaborazione è costituita da una sequenza di passi che vedono successivi cambiamenti della configurazione del computer e che consistono in piccoli cambiamenti delle informazioni contenute nei dispositivi di memoria del computer stesso e che nel complesso portano alla derivazione dalle informazioni in ingresso, i dati della elaborazione attuale, delle informazioni in uscita, i risultati della elaborazione.

Ogni elaborazione viene controllata da un programma che consideriamo una implementazione, ossia un adattamento al computer, di un algoritmo che può essere anche molto composito.
Anche un programma è costituito da un complesso di informazioni che può essere registrato nel computer e trasmesso atraverso canali di comunicazione tra dispositivi digitali anche molto distanti tra di loro.

Ogni programma viene trattato da persone, che chiamiamo programmatori, e dispositivi di calcolo, i computers e apparecchiature per la trasmissione.
Un programma è formulato da un testo scritto in un determinato linguaggio di programmazione che in qualche miodo deve essere letto e interpretato da programmatori e computer.

Nel corso della storia del calcolo automatico, dagli anni 1940 a oggi, l'organizzazione dei computer e di quanto le circonda si è evoluto continuamente in misura rilevante e in molte direzioni.
Per semplicità ci limitiamo a discutere la situazione che più qui interessa e consideriamo programma un testo scritto in un linguaggio sorgente comprensibile direttamente dai programmatori e solo indirettamente dai computers. Si deve quindi avere un processo di traduzione di ogni programma sorgente in un programma più comprensibile al computer che chiamiamo programma eseguibile.

B70a.03 Cominciamo a descrivere un primo schema, che denotiamo con P1, secondo il quale si ottiene la comprensione da parte del computer di un programma scritto da un programmatore e leggibile da altri.
Un tale programma, che chiamiamo programma sorgente, è costituito da stringhe di caratteri leggibili che si presentano come linee successive; ora ci limitiamo a pensare ad un programma piuttosto semplice formato da poche decine di linee, ciascuna con meno di 80 caratteri.

Questo programma sorgente viene trasformato da un programma chiamato compilatore in un cosiddetto programma compilato scritto in un cosiddetto linguaggio oggetto più comprensibile al computer del sorgente.
Su questo linguaggio diciamo solo che esprime un complesso di comandi per il computer che essenzialmente richiedono l'entrata in funzione delle istruzioni a disposizione del computer, ossia che provocano l'attivazione dei suoi circuiti operativi.
Una situazione di questo tipo si riscontra nelle usuali calcolatrici numeriche più semplici, macchine in grado di eseguire le operazioni aritmetiche e alcune altre chiamate scientifiche in grado di manipolare uno o due dati numerici espressi finitamente per ottenere un solo risultato dello stesso genere.

I semplici programmi che vogliamo considerare però controllano azioni più complesse, in quanto possono eseguire, oltre alle sequenze di operazioni numeriche, operazioni di lettura dall'esterno, di scrittura verso l'esterno, manipolazioni di caratteri e testi leggibili e, soprattutto decisioni basate su dati precedentemente resi disponibili che comportano di scegliere tra successive possibili sequenze di operazioni.

Inoltre si deve tener conto che il programma sorgente sottoposto al compilatore può contenere errori di tanti tipi, dai refusi ad errori di imostazione, che in qualche modo devono essere segnalati e seguiti da qualche tipo di correzione o dall'arresto della elaborazione; in ogni caso serve una segnalazione, il più possibile chiara, del tipo dell'errore e, auspicabilmente, serve qualche suggerimento di una correzione rivolto al programmatore.

Aggiungiamo che con un programma di poche decine di linee non si riescono a risolvere molti problemi; sicuramente si rendono necessari complessi di comandi e di scelte decisamente più lunghi e complessi e questo richiede attività di programmazione impegnativi che devono essere molto articolati e quindi richiedere organizzazioni ben più complesse nelle quali entrano attori che non si riducono a un programmatore e a un computer che esegue tutte le richieste che ha fornito preima di una elaborazione.

Nei paragrafi che seguono discutiamo brevemente la prima tecnica da adottare per organizzare i programmi di una certa complessità, la loro cosiddetta "organizzazione modulare"; infatti alla nozione do modulo di programma dovremo fare riferimento anche prima di presentare in dettaglio organizzazioni modulari specifiche.

In seguito dobbiamo occuparci di vari dettagli delle unità informative usate nei programmi e delle nozioni riguardanti le operazioni di immissione e di emissione di dati e solo successivamente [:g] potremo presentare i primi programmi concreti che seguono il semplice schema P1 sopra descritto.

B70a.04 Accade spesso di dover risolvere un problema impegnativo **P** con un procedimento automatico definito solo a grandi linee che si prevede richieda di eseguire molteplici manovre che coinvolgono tanti tipi di dati disponibili all'interno della organizzazione interessata, oppure ottenibili da fonti esterne attraverso la lettura di più unità periferiche e che si prevede debba affrontare diversi percorsi operativi che dipendono dalle caratteristiche delle diverse possibili istanze del problema.

In queste circostanze risulta conveniente sforzarsi di individuare problemi più ridotti relativamente semplici da risolvere che si possono considerare sottoproblemi di **P** in quanto tali che, componendo opportunamente le corrispondenti soluzioni si possa ottenere la soluzione dell'intero **P**.

Situazioni di questo genere si sono presentate spesso anche quando i problemi potevano essere risolti solo con interventi umani; in questi casi si era cercato di organizzare il lavoro di più persone che fossero in grado di risolvere i sottoproblemi e il lavoro di un coordinatore capace di controllare il procedere dei lavori parziali e di far utilizzare le diverse soluzioni parziali fino ad assemblare un risultato finale.

Queste attività, che chiamiamo organizzazioni modularizzate, si sono riscontrate in tutti i periodi storici e in tutti i paesi, fin dalle prime civiltà dedite alla caccia e all'agricoltura.

L'organizzazione modulare si riconosce anche nella matematica, in tutti i campi della tecnica, dell'ingegneria, della medicina, nelle amministrazioni, nelle biblioteche, nei trasporti, nelle forze armate e in tutte le altre attività impegnative condotte con razionalità.

L'organizzazione modulare è stata adottata e studiata con particolare attenzione e sistematicità nelle attività computazionali basate sui dispositivi elettronici e digitali.
In questa direzione si è sviluppata la nozione di sottoprogrammi e si è giunti a disporre di collezioni di sottoprogrammi molto numerose e variegate, le cosiddette librerie di sottoprogrammi.
Questi oggetti da tempo sono considerati importanti prodotti industriali dotati di una determinante valenza economica e strategica.

B70a.05 Un programma per computer odierno, che si propone obiettivi anche solo modestamente ambiziosi presenta una struttura modulare fatta di tante componenti, i moduli di programma, in numeri che vanno dalle decine alle molte migliaia.
Vediamo che tipi di compiti svolgono i moduli che si trovano in quasi tutti i programmi con obiettivi di rilievo.

Vi sono moduli che sono incaricati di leggere dati di ingresso per ciascuna delle istanze del problema trattato.
Specularmente sono presenti sottoprogrammi che si occupano della presentazione dai risultati ottenuti da ogni esecuzione del programma.
Possono essere presenti sottoprogrammi dedicati al calcolo di valori di funzioni matematiche, dalle basilari funzioni esponenziali, logaritmiche e trigonometriche alle più complicate funzioni speciali, in conseguenza dalle possibili esigenze computazionali.
Altre esigenze applicative portano a richiedere sottoprogrammi per calcoli su matrici, sistemi di equazioni, integrali, soluzione di equazioni differenziali, calcoli di trasformate e tanto altro.
Molti sottoprogrammi disponibili riguardano le elaborazioni di ampie quantità di dati numerici per esigenze statistiche, di simulazione o di previsione.
Quando un problema richiede di servirsi dei dati disponibile in una base di dati servono i sottoprogrammi resi disponibili da un DBMS, ossia da un DataBase Management System, alcuni dei quali per l'accesso di dati, altri per la registrazione di nuovi dati nella base di dati.
I problemi riguardanti logistica, organizzazione di attività o presa di decisione possono richiedere sottoprogrammi messi a punto negli ambiti della programmazione lineare, della quadratica e della ricerca operativa.
Per molti problemi di organizzazione dei dati servono sottoprogrammi studiati nell'ambito delle teorie combinatorie; ricordiamo in particolare i programmi di ordinamento di dati numerici, di nomi, di termini o di collezioni di strutture di dati (ad esempio di dati geografici o astronomici).
Molti programmi si concludono con l'emissione di grafici, mappe e altri tipi di immagini costruite artificialmente: questi risultati richiedono sottoprogrammi dell'area della computer graphics.
Per trattare problemi riguardanti sistemi e reti di comunicazione servono sottoprogrammi riguardanti grafi.

B70a.06 Senza insistere nella tipologia dei sottoprogrammi, in particolare non entrando nelle librerie di sottoprogrammi per fini specialistici, vediamo a grandi linee come vengono strutturati i sottoprogrammi in un linguaggio procedurale come C e C++.

Innanzi tutto un sottoprogramma si può considerare come un programma con finalità circoscritte che si serve di dati di ingresso e produce risultati in uscita da e verso un unico ambiente, il programma che lo ha invocato in un certa fase della sua elaborazione.

Quindi il programma chiamante deve fare in modo di fornire i dati attuali al sottoprogramma e di ricevere i risultati ottenuti, come accade allo stesso programma, con la restrizione che sia la fornitura che la ricezione di dati vengono eseguite una sola volta per ciascun richiamo del sottoprogramma.
Come vedremo questi passaggi di informazioni si possono fare in modi diversi e quindi vi sono da distinguere diverse organizzazioni per le chiamate dei sottoprogrammi.

Risulta evidente che nei programmi molto impegnativi si devono utilizzare sottoprogrammi anch'essi piutosto impegnativi che risulta opportuno si servano di loro sottoprogrammi incaricati di lavori più semplici.
Si puoò quindi prospettare una organizzazione di un programma che si serve, direttamente o indirettamente di sottoprogrammi organizzati gerarchicamente, un modulo di tale gerarchia potendo essere richiamato da più moduli di livelli superiori diversi.
Evidentemente questi complessi di moduli di programma devono essere organizzati con grande accuratezza e seguendo criteri piuttosto elaborati che sono stati ampiamente studiati.
Tra i vantaggi di questo modo di prganizzare il software vi sono quello consistente nel riutilizzare molti sottoprogrammi per molti programmi diversi e quello di sistematizzare grandi attività di programmazione per riuscire a sviluppare una grande varietà di strumenti per la risoluzione di problemi.
Tutto questo esige la definizione di metodi organizzativi del software molto accurati e la adozione di strategie di portata molto ampia. Queste questioni vengono poste nella disciplina chiamata ingegneria del software, parte molto importante dell'informatica che costituisce anche un settorecon rilevanza economica e industriale.
L'ingegneria del software ha a che fare con tutti i maggiori linguaggi di programmazione, rispetto ai quali ha una prevedible dipendenza, ma che anche contribuisce a far evolvere per quanto riguarda lo sviluppo di nuovi linguaggi e di nuove versioni di linguaggi consolidati che devono tenere conto delle indicazioni della ingegneria del software soprattutto per quanto riguarda la produttività, la adeguabilità rispetto alla evoluzione complessiva (del software, dell'hardware e delle esigenze dei settori applicative) e della sicurezza dei sistemi computazionali.

B70 b. linguaggio c++, generalità

B70b.01 Prima di introdurre le prime nozioni sul linguaggio c++, riprendiamo le due sostanziali motivazioni della sua scelta.

Il linguaggio C, predecessore del linguaggio C++, da decenni è ampiamente conosciuto e utilizzato e i suoi costrutti sono stati ampiamente ripresi in molti altri linguaggi di programmazione, sia da linguaggi specialistici, che da linguaggi a un livello superiore del C.
Quindi c++ consente di presentare algoritmi che possono servire a molti ambienti della programmazione e viceversa è in grado 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 ai più avanzati livelli della progettazione dei programmi e della analisi delle prestazioni.

In altre parole C++ occupa una posizione importante nella ingegneria del software e riesce a convivere insieme a vari altri linguaggi di programmazione.
Sono numerose le pubblicazioni che rendono disponibile una vasta gamma di esempi con i quali i contenuti presentati nelle pagine che seguono possono trovare varianti, ampliamenti di portata e soluzioni complementari.

Fatto non secondario, sia C che C++ sono linguaggi standardizzati da organismi internazionali e questo ha contribuito a renderli facilmente portabili tra diversi computers e tra diversi sistemi operativi.

La versione standard più recente di C++ è sostenuta da due importanti organismi, `ISO (we)`, International Organization for Standardization, e `IEC (we)`, International Electrotechnical Commission ed è ufficialmente nota come C++23; essa è stata pubblicata nel 2023.
Anche la versione standard più recente del linguaggio C vien sostenuta da ISO e da IEC ed è stata resa pubblica nel 2023; essa è ufficialmente nota come C23.

B70b.02 La seconda motivazione di c++ 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 un linguaggio meno praticato e non sarebbe direttamente attuabile se si fosse adottato qualche genere di `pseudocodice (wi)`, ossia qualche presentazione espressa in termini più discorsivi e più facilmente accostabili.

La disponibilità di manuali sui linguaggi C e C++, tra l'altro, ci consente di introdurre le prestazioni di c++ che ci sembrano più rilevanti per l'*esposizione*con discorsi concisi che evitano di soffermarsi su molti dettagli che riteniamo di interesse secondario.

La scelta di c++, oltre che rendere disponibili algoritmi effettivamente verificabili, intende favorire la costituzione di librerie di programmi che portino anche a considerare come problema culturale e di ampia influenza quello della disponibilità di raccolte di procedure.

Inoltre i programmi che presenteremo pensiamo servano a evidenziare varie considerazioni alle quali diamo molto peso come le seguenti.
Tutti i procedimenti costruttivi riguardano prevalentemente sequenze finite, a partire da quelle su interi e stringhe.
È significativo e utile esaminare le differenze tra alcune formule matematiche e le loro possibili implementazioni.

B70b.03 È opportuno anche segnalare che le attività di implementazione di algoritmi di interesse per la matematica conducono naturalmente a porsi una vasta gamma di problemi che riguardano nozioni presenti da tempo in questa disciplina.

Un prima tema che emerge dalle implementazioni riguarda le relazioni che intercorrono tra molte nozioni presenti nelle argomentazioni della matematica e nozioni di uso comune nelle dicussioni sopra la programmazione, il trattamento dei dati e la soluzione effettiva di problemi di vasto interesse.

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 espressa 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.

B70b.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 trattate distintamente sia da esecutori umani che da esecutori artificiali, soprattutto quelli attribuibili all'elettronica digitale.

B70 c. informazioni booleane e numeri interi

B70c.01 Ogni dispositivo per elaborazioni automatiche e ogni macchina in grado di implementare operazioni matematiche, deve essere in grado, innanzi tutto, di trattare le entità informative più semplici ed essenziali, le cifre binarie, ossia i bits.
Questi oggetti formali servono primariamente a rappresentare i due valori di verità, il vero e il falso, valori che in genere e in particolare nel linguaggio C++ vengono associati, rispettivamente, ai numeri interi 1 e 0.
L'importanza 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 oggi è possibile immagazzinare, elaborare e trasmettere grandissime quantità di informazioni binarie con dispositivi minuscoli e con costi operativi conservazione molto contenuti.

B70c.02 Come viene detto in particolare in B60, ai valori di verità si possono applicare le operazioni booleane concernenti le sentenze, ossia gli enunciati ai quali si è convenuto possibile assegnare uno di tali valori.
Un bit, considerato portatore 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 in genere riguarda la effettuazione o meno di una data azione, oppure l'esecuzione di una di due manovre diverse; gli effetti ottenibili servendosi di queste celte basilari rende i valori di verità fondamentali nello studio r nelle applicazioni delle procedure.

B70c.03 Un bit può essere usato anche per esprimere la presenza o l'assenza di un particolare oggetto in una prefissata collezione. Per trattare questioni concernenti l'appartenenza o meno ad un insieme finito E presentato con una sequenza, ovvero con una lista, di n componenti risulta naturale utile servirsi di sequenze di n cifre binarie: infatti ciascuna di tali sequenze consente di individuare un sottoinsieme di E, in quanto 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 $b_i \in \{0, 1\}$.

Bisogna aggiungere due altri ruoli fondmentali dei bits.
Tutti i circuiti digitali sono costituiti da componenti che eseguono una delle operazioni binarie: gli operandi o l'unico operando di una di queste operazioni sono forniti da impulsi elettrici chiaramente

attribuibili a due classi, quella rappresentante il bit 0 e quella corrispondente al bit 1; un impulso simile rappresenta il risultato dell'operazione.
Le memorie a stato solido, attualmente di largo uso, sono costituite da microcircuiti NAND in grado di conservare e restituire segnali di due tipi, ciascuno dei quali atto a rappresentare un valore binario.

B70c.04 La tecnologia odierna consente di registrare, trasmettere ed elaborare con grande efficienza e velocità enormi quantità di cifre binarie.
Sono disponibili dischi magnetici e banchi di memorie a stato solido in grado di registrare molti triliardi di bits, ovvero molti multipli di 10^{12} di bits e banchi di dispositivi circuitali statici (memorie a stato solido) con capacità paragonabili.
I circuiti operativi dei processori attuali sono in grado di elaborare informazioni binarie a velocità di miliardi di operazioni al secondo.
Come si ricava da una serie di situazioni, tutte le informazioni trattabili con le attuali apparecchiature informatiche possono essere rappresentate mediante sequenze binarie.

Le precedenti considerazioni rendono ben comprensibile la attuale tendenza generale di servirsi di sequenze di bits per la registrazione, la trasmissione e l'elaborazione di tutte le informazioni che si vogliono gestire mediante 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 con il computer e da gestire con linguaggi di programmazione.

B70c.05 Per rappresentare le informazioni delle diverse specie si rende necessario che i dispositivi hardware e i linguaggi di programmazione consentano di controllare unità di registrazione di diverse capacità.
Occupiamoci per ora dei numeri interi.

Mentre per esporre proprietà matematiche generali degli interi in genere non ci si deve preoccupare di quanto siano grandi, per decidere come implementarli si deve essere consapevoli che si possono trattare con semplicità e uniformità solo interi che variano in intervalli limitati.

Si osserva che si possono trattare con maggiore efficienza interi che possono assumere solo valori assoluti limitati, mentre si possono trattare con maggiore versatilità interi che possono assumere valori massimi maggiori, ma che necessariamente ciascuno di essi richiede più bits richiede dispositivi di memoria più costosi e circuiti operativi più estesi e che richiedono più energia o maggiori tempi esecutivi.

In talune circostanze conviene operare con una collezione di numeri di valori limitati ma che si possono elaborare efficientemente; in altre è preferibile una collezione numerica più estesa che si possa trattare con meno preoccupazioni sui massimi valori che possono assumere, ma più costosa in termini di dispositivi da impiegare, di tempi esecutivi, di energia consumata e di riscaldamento da tenere contenuto.

Prevalentemente si trattano interi rappresentabili con 16, 32, 64 o 128 bits e si può stabilire se si vogliono solo numeri nonnegativi oppure numeri che possono essere sia nonnegativi che negativi.

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 possono assumere i valori nell'intervallo $\mathbf{[} -32\,768 : 32\,767\mathbf{]}$.

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, dato che $2^{64} = 18\,446\,744\,073\,709\,551\,616$, si possono trattare gli interi naturali dell'intervallo $\mathbf{[}0 : 18\,446\,744\,073\,709\,551\,616\mathbf{]}$, oppure gli interi relativi dell'intervallo $\mathbf{[} -9\,223\,372\,036\,854\,775\,808 : 9\,223\,372\,036\,854\,775\,807\mathbf{]}$.

B70c.06 Nel seguito faremo riferimento quasi esclusivamente a personal computers di uso comune, le macchine più 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 vedremo i computers e le altre apparecchiature con intelligenza digitale come contenitori di grandi sequenze di bits.

I dispositivi per la registrazione delle informazioni binarie, sostanzialmente i circuiti elettronici degli odierni computers, li chiamiamo **dispositivi di memoria**.
I dispositivi, che consentono di trattare (immettere, trasformare ed emettere) le informazioni digitali per la massima parte riguardano sottosequenze di lunghezza ben definita di registri binari con ben definiti insiemi di valori.
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-mm**.

B70c.07 Nelle odierne apparecchiature elettroniche vengono trattate soprattutto celle-mm riguardanti sottosequenze di 8, 16, 32, 64 e 128 bits. Le celle-mm di s registri binari le diremo in breve celle di s bits o anche celle-sb.

Chiameremo **bytes** o **ottetti** i contenuti delle celle-mm 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, boxes, disposte orizzontalmente e caratterizzando le loro posizioni 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 essere modificati nel corso di una elaborazione e gli interi che forniscono le loro posizioni si possono chiamare i loro indirizzi interni.

Abbiamo quindi raffigurazioni come le seguenti:

//input pB70b05

B70c.08 Nelle considerazioni che seguono presentiamo le risorse di memoria seguendo lo schema che viene proposto ai programmatori in un linguaggio di alto livello come C e C++.
Questo schema costituisce una semplificazione standardizzata di quanto si riscontra fisicamente, ossia al livello operativo dei circuiti hardware dei computres attuali. Le memorie presentate ai programmatori sono vicine alle memorie materiali di cui erano dotati i computers strettamente sequenziali degli anni 1960-1970, nei quali si aveva un unico tipo di memoria centrale.
I sistemi successivi, oltre a differenziarsi notevolmente, hanno adottato tecniche via via più articolate e complesse volte ad aumentare le prestazioni complessive utilizzando i dispositivi hardware dei diversi generi che si rendevano disponibili, dispositivi a diversi livelli che in genere di dispongono in successivi

strati dai più vicini ai circuiti operativi ai più lontani, strati nei quqli si hanno velocità operative decrescenti, costi unitari decrescenti ed estensioni crescenti; con questo genere di organizzazione si devono adottare strategie organizzative e procedure di supporto necessariamente più elaborate.

Le schema semplificato che seguiremo ha due importanti pregi: consente ai i programmatori di produrre programmi che si adattano a una grande varietà di sistemi harware e mantengano la loro efficacia nel tempo; evita che i programmatori si debbano occupare dei complesssi meccanismi dietro le quinte sopra accennati.
delle memorie (virtuali) su dischi e banchi a stato solido.

B70c.09 Secondo lo schema delle memorie semplifcato per il programmatore, il programma dispone di una memoria centrale nella quale si trovano le celle-mm di diverse estensioni.
Ciascuna di esse può essere raggiunta dai circuiti operativi a della unit a di controllo per essere letta o scritta attraverso il suo **indirizzo**, l'intero naturale che rappresenta la sua posizione nella sequenza (virtuale) che costituisce la memoria centrale.

Il programmatore non deve occuparsi della disposizione nella memoria centrale, di questa si incarica il sistema di gestione dei programmi.
Il programmatore gestisce singole celle-mm, sequenze di celle-mm o schieramenti di celle-mm di due o più dimensioni riconducibili a sequenze attraverso i loro identificatori la cui scelta è a sua completa disposizione.
In tal modo egli riesce a controllare pienamente le informazioni che lo interessano e nient'altro; l'unica sua preoccupazione riguarda la estensione complessiva di memoria centrale a sua disposizione.

Occorre anche segnalare che tra i diversi ruoli delle celle-mm vi è anche quello di individuare le posizioni di altre celle-mm; l'estensione delle celle-mm dedicate agli indirizzi evidentemente dipende dalla ampiezza della memoria disponibile, una memoria più ampia richiede indirizzi più lunghi.

B70c.10 Coerentemente con quanto visto per le celle elementai binarie di una cella-mm, conviene raffigurare le sequenze di celle-mm con file di rettangoli disposti orizzontalmente caratterizzati da indirizzi che crescono procedendo da sinistra verso destra.
Può servire visulizzare il contenuto di una sequenza di celle-mm indicando in questi rettangoli i valori dei rispettivi contenuti correnti e segnalando qualche indirizzo . Il valore variabile nel tempo del contenuto di una cella-mm può essere rappresentato diversamente a seconda del suo ruolo, ovvero dell'utilizzo al quale è destinato, o a seconda del significato che gli si attribuisce in una particolare presentazione di alcune clle-mm.
Torneremo sull'argomento parlando di variabili e operazioni relative richieste nel linguaggio.

Occorre precisare che vanno distinti gli indirizzi assegnati alle diverse taglie di celle-mm, cioè ai bytes, alle halfwords, alle 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.
Le capacità delle memorie più comodamente utilizzabili da un computer attuale può superare il terabyte.

B70c.11 Per trattare gli indirizzi e i contenuti della memoria centrale conviene avere presenti le potenze di 2 [W10a01].
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 GB e 8 GB, anche se notazioni per maggiore precisione si dovrebbero usare le notazioni 1 Gibiby e 8 Gibiby.

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 binaria 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
```

B70c.12 Come si è detto, i contenuti di ciascuna delle celle-mm possono rappresentare oggetti diversi in dipendenza del ruolo che alla cella il programmatore sta attribuendo.
Il ruolo di una cella-mm viene stabilito a livello software da una dichiarazione nel linguaggio di programmazione, mentre a livello hardware viene preso in considerazione dai circuiti operativi che operano sulla cella.

Una cella-mm 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 0000000 e 10100000 00000000.
Gli interi 10, 100, 1000 e 10000 sono forniti, rispettivamente, 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.

B70c.13 Una cella-mm di s bits la chiameremo **cella-sb** 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 $011100001111_2 = 3854_{10}$.

B70c.14 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 $b_0b_1b_2\cdots b_{s-2}$ fornisce l'intero $\sum_{i=0}^{s-2} b_i\,2^i$.
Il generico intero negativo è dato dalla sequenza della forma $b_0b_1b_2\cdots b_{s-2}1$ 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 questa sequenza considerate 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:

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

B70c.15 Può servire osservare che se n denota un signed integer dato dalla sequenza binaria $b_0b_1\cdots b_{s-1}$ 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 pratica offre precisi 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-mm dell'intervallo $\mathbf{[}-2\,147\,483\,648 \;:\; 2\,147\,483\,647\mathbf{]}$, che l'insieme degli interi-mm 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 $-170\,141\,183\,145\,469\,231\,731\,687\,303\,715,884\,105\,727$.

B70c.16 Per taluni scopi risulta opportuno considerare anche celle di memoria corrispondenti a quaterne di bits; a queste celle-mm si da il nome di **nibbles**, ossia di bocconi.
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 $FFF\ FFF\ FFF\ FFF\ FFF$.

B70c.17 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 rappresentare valori interi molto elevati attraverso sequenze di celle-mm della estensione opportuna che potrebbe essere 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 adattabilità, ovvero elevata affidabilità e in particolare che non incorrano in errori di superamento dei limiti per i numeri interi incontrati.
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 condizionati da superamento dei valori massimi garantiti.

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 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 con l'osservazione di un certo numero di risultati empirici preliminari.

Alle precedenti considerazioni sopra gli insiemi finiti di interi trattabili si dovranno aggiungere 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 d. informazioni simboliche

B70d.01 Vediamo ora come vengono trattate comunemente nel linguaggio C++ i caratteri leggibili e le **informazioni simboliche**, in pratica le stringhe di caratteri appartenenti a un preciso alfabeto.
Questo alfabeto comprende le cifre decimali, le lettere dell'alfabeto inglese, 26 maiuscole e 26 minuscole, segni di interpunzione e alcuni segni che denotano operazioni matematiche.
Oltre a questi segni visualizzabili sono trattabili alcuni caratteri che svolgono ruoli importanti nel controllo dei dispositivi periferici come stampanti e schermi video e nel controllo della trasmissione delle informazioni con altri computers e con apparecchiature digitali che seguono precise regole operative concordate.

le stringhe dei caratteri trattabili consentono di elaborare gli usuali testi leggibili che possono essere stampati, presentati su uno schermo, registrati su un dispositivo di memoria portatile (disco o chiavetta flash), o emessi da un altoparlante e che possono essere immessi nel sistema digitati su tastiera, scritti a mano su uno schermo adeguato, catturati da un microfono o presi da un dispositivo di memoria trasferibile.

Con il linguaggio C++ e con molti altri linguaggi, i dati simbolici si trattano seguendo soprattutto il sistema di codifica detto **codice ASCII** memorizzando ogni stringa in una sequenze di bytes consecutivi, un carattere per byte.

Altre modalità consentono di eleborare alfabeti più ampi con tecniche più elaborate; tra queste sono prevalenti quelle del cosiddetto **codice Unicode**, sistema di codifica che si è proposto di rendere trattabili con modalità standard la massima parte delle lingue naturali e artificiali effettivamente praticate nel presente e nel passato che presentiamo in :d04-d05.

B70d.02 ASCII è l'acronimo di American Standard Code for Information Interchange e costituisce uno standard internazionale ampiamente riconosciuto.
Per la precisione occorre distinguere tra le due varianti ASCII-7 ed ASCII-8 che registrano un carattere, rispettivamente con 7 e con 8 bits.
La prima variante è stata ampiamente utilizzata a partire dagli anni 1965-1970, periodo nel quale si sono imposti i sistemi in grado di gestire le celle-mm di 8, 16, 32, 64 e 128 bits.
La seconda è un ampliamento della prima e include un buon numero di caratteri visualizzabili delle maggiori lingue occidentali e caratteri utilizzati per la più semplice grafica e per i più semplici videogiochi.

ASCII dedica i primi 32 bytes, corrispondenti ai valori decimali da 0 a 31, ai valori binari da 0000000_2 a 00011111_2 e ai valori esadecimali da 00_{16} a $1F_{16}$ e il byte $127 = 01111111_2 = 7f_{16}$ ai cosiddetti **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

B70d.03 I 95 bytes con valori numerici compresi tra 20_{16} e $7E_{16}$ sono dedicati ai caratteri visualizzabili o stampabili, cioè alle 26 lettere dell'alfabeto romano-inglese, alle cifre decimali, ai segni di interpunzione e alcuni segni per operazioni matematiche.

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	{	|	}		

B70d.04 Tra i produttori di hardware intorno al 1970 ha cominciato ad imporsi 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_{16} a ff_{16}).
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, per gli interi da 64 bits 8 bytes consecutivi e per 18 bita 16 bytes consecutivi.

L'importanza dei bytes è dovuta all'ampio utilizzo di collezioni di caratteri con un numero di segni tra una e due centinaia. 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 1965-1970. 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 segni 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 delle lingue di paesi, sostanzialmente occidentali, 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 con il nome ufficiale `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**d.05** Il termine **Unicode** denota un importante e ambizioso progetto internazionale avviato nel 1991 che si è proposto di standardizzare l'implementazione digitale della maggior parte dei caratteri e degli altri segni utilizzati nei testi prodotti in tutte le parti del mondo, nel presente e nel passato, comprendendo testi letterari, matematici, musicali e testi dedicati alle svariate tecniche.

Il suo scopo è quello di agevolare il più possibile la circolazione digitale e telematica dei dati, delle informazioni, delle conoscenze e conseguentemente delle idee.

Il progetto Unicode viene portato avanti dal consorzio Unicode Consortium sostenuto dalle importanti organizzazioni ISO e IEC e si occupa della codifica digitale dei caratteri e dei segni che organizza e classifica nel cosiddetto **Universal Coded Character Set**, abbreviato da **UCS**.

Nel suddetto character set sono contenuti tutti gli alfabeti dei linguaggi naturali del presente e del passato della cui utilizzazione, scritta od orale, si riesce a trovare una accettabile documentazione.
In UCS vengono collocati anche i caratteri e i segni usati in una ampia gamma di linguaggi artificiali e di sistemi di scrittura.
Nella versione di Unicode 16.0.0 del 2024-09-10 risultano definiti 154 998 caratteri suddivisi in blocchi di simboli [https://en.wikipedia.org/wiki/Unicode_symbol] e 168 cosiddetti "scripts", collezioni di segni utilizzati negli scritti prodotti da attività comunicative, accademiche, industriali e tecniche in senso lato [https://www.unicode.org/charts/PDF/Unicode-16.0/U160-1CC00.pdf].

Esempi di blocchi di caratteri: simboli delle divise monetarie, segni a deponente e a esponente, numeri enumeri Maya, lettere incorniciati in cerchi, simboli dei sistemi fonemici e fonetici (IPA), emoji ed emoticon, frecce, operatori matematici, forme geometriche, simboli pittografici, simboli musicali, simboli per carte da gioco, scacchi, domino, mahjong, ...

Esempi di scripts : geroglifi egizi, simboli pittografici, simboli di sistemi di scrittura sillabici, simboli delle lingue asiatiche, africane e di tutte le altre lingue esotiche, simboli utilizzati da prodotti informatici

anche tra quelli ora in disuso, simboli per circuiti elettrici ed elettronici, frecce, simboli stradali, simboli alchemici ed ermetici, ...

B70d.06 Il sistema di codifiche Unicode 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.
In effetti Unicode viene supportato da un gran numero di prodotti software di vasta adozione: linguaggi di programmazione, compilatori, IDE, linguaggi per la comunicazione come HTML, browsers, piattaforme per posta elettronica, DBMS, sistemi operativi, traduttori automatici, generatori di immagini e suoni, ...

Questo sistema di codifiche mette a disposizione 16, 20 o 24 bits per ogni codepoint, sequenza di bits che costituisce la codifica di un carattere o script facente parte di UCS secondo uno schema un po' elaborato.
Lo schema prevede 17 cosiddetti planes, ciascuno costituito da $65\,536 = 2^{16}$ codepoints contigui e quindi consente di trattare fino a 1 114 112 simboli.
Per ottimizzare statisticamente memorizzazione, trasmissione e eleborazione dei testi si assegnano codepoints più corti ai segni per i quali si prevede un utilizzo più frequente.

B70 e. costanti e variabili; dichiarazioni e assegnazioni

B70e.01 Per quanto detto, un programma per un computer dotato di un comune semplice corredo di unità periferiche (tastiera, stampante, monitor video, porta USB per dispositivi flash), può vedersi come un complesso di richieste 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, se non si verificano situazioni da considerare patologiche, fornisce nuovi dati con il ruolo dei risultati della elaborazione effettuata.

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; in genere solo una parte di essi questi solo una parte viene emessa per costituire il risultato dell'esecuzione.

Tra i dati che un programma può elaborare nelle sue possibili esecuzioni vanno distinti quelli che in ciascuna esecuzione 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 ci si aspetta possano essergli richieste, deve essere dotato di norme operative che hanno la forma di programmi predisposti e che attualmente possono essere molto articolati, ma che l'utenteinteressato ai risultati deve conoscere soltanto in relazione alle sue motivazioni.
Una parte primaria degli strumenti in dotazione del computer costituisce il sistema operativo del computer (Windows, Unix, Mac OS, ...) , sistema di programmi che interagiscono con l'utente e danno tutti i supporti necessari alle manovre richieste per le molteplici prestazioni esecutive.

Per condurre efficacemente le esecuzioni governate dai programmi applicativi degli utenti 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**.

Qui 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 hardware e software costituenti il sistema di sviluppo stesso; (2) monitorare ciascuna delle esecuzioni dei programmi segnalando gli accadimenti che possono interessare l'utente e in particolare le situazioni che giudica erronee e le situazioni che fanno sospettare che l'elaborazione in corso non stia fornendo risultati corretti.

B70e.02 Anche il sistema di sviluppo di un linguaggio come C++ non deve necessariamente 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 che possa prender facilmente le decisioni più opportuneper il raggiungimento dei suoi obiettivi.

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 da parte di un utente sistematico del computer si può configurare come formazione di un proprio metodo e di un proprio stile di lavoro.
Questa formazione sarà particolarmente impegnativa quando il computer viene utilizzato da intere squadre di programmatori che devono affrontare problemi computazionali impegnativi.

Ogni sistema di sviluppo di un linguaggio di ampia portata è un prodotto industriale decisamente evoluto che in buona parte presenta dipendenze dal computer utilizzato e dal sistema operativo istallato.

Esso svolge svariati compiti, soprattutto:
- tradurre le richieste formulate in ogni programma in richieste comprensibili per la macchina;
- individuare e segnalare opportunamente gli errori lessicali e sintattici che trova nel testo del programma e gli errori semantici potrebbe rilevare nella struttura del programma;
- segnalare le eventuali anomalie si verificano nel corso dell'esecuzione;
- mettere a disposizione ampie librerie di sottoprogrammi di utilità generale;
- supportare la gestione di programmi di sistema con i quali un programma scritto dall'utente può interagire

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.

B70e.03 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 sistemi operativi, possiamo pensare che il 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 vengono coinvolti dal programma.

B70e.04 Ciascuna delle celle-mm coinvolte da un programma viene individuata dall'indirizzo del primo dei bytes a essa assegnati e dal loro numero, corrispondente al suo tipo di dato.

In molte circostanze occorre saper controllare di una cella-mm sia l'indirizzo iniziale che la sua estensione; come vedremo il linguaggio C per questo offre diverse possibilità.

Chi formulava i programmi per i primi computers utilizzati poteva servirsi solo dei codici binari delle istruzioni della macchina e doveva servirsi degli indirizzi fisici delle celle-mm nelle quali collocare i dati da elaborare. 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, a causa del gran numero di dettagli che il programmatore doveva avere presenti.

Un primo miglioramento si è avuto negli anni 1950 con la introduzione dei cosiddetti linguaggi simbolici di macchina; ciascuno di essi consentiva di controllare le prestazioni dei computers di un dato modello mediante simboli esprimenti istruzioni e mediante nomi per le celle-mm che il programmatore poteva scegliere in modo da poterli ricordare con facilità.

Successivamente gli sviluppi delle relative tecnologie hanno reso i computers sempre più miniaturizzati, efficienti, affidabili, versatili e diffusi e hanno indotto i produttori a dotarli di una varietà di dispositivi ausiliari rivolti a migliorare le interazioni uomo-macchina e i collegamenti tra la macchina e altre apparecchiature esterne, anche remote; tra queste ultime terminali per utilizzatori distanti, sensori, attuatori e altri computers impegnati in elaborazioni con finalità in comune.

B70e.05 Contemporaneamente e sinergicamente gli elaboratori elettronici sono stati dotati di una crescente varietà di strumenti software rivolti a facilitarne l'utilizzo: sistemi operativi, traduttori di linguaggi, librerie di sottoprogrammi, strumenti per la diagnosi di errori di programmazione e di malfunzionamenti dell'hardware, sistemi per la gestione di archivi di dati, strumenti per il sostegno allo sviluppo di programmi via via più ambiziosi, ...

L'utilizzo dei computers, le cui popolazioni sono passate dalle decine ai miliardi di esemplari, si è trasformato dall'essere una pratica guidata dalle esigenze di applicazioni specifiche e dalle caratteristiche di singole realizzazioni hardware al costituire una fenomenologia complessa da esaminare in relazione all'andamento dell'economia e delle imprese, allo sviluppo della società e a una nuova cultura che dà ampio credito alle metodologie e alle soluzioni condivise e alle esigenze di ampie categorie di utilizzatori più o meno diretti.

Tra queste ultime non si possono trascurare le categorie dei detentori di poteri riguardanti imprese finanziare, industriali, della logistica e dei commerci, dei media e dello spettacolo, i servizi sanitari e dell'istruzione, dei poteri militari e politici complessivi, nonché, indirettamente ma massicciamente la categoria dei consumatori di intrattenimento e videogiochi.

Naturalmente non ci addentreremo nei molteplici aspetti di questi della società odierna e ci poniamo dal punto di vista della adozione di un linguaggio di programmazione procedurale medio-alto e di portata vasta come il C++, cercando tuttavia di segnalare i collegamenti più influenti tra i suddetti sviluppi e le metodologie della programmazione.

B70e.06 In un linguaggio come C++ i dati costanti sono individuati mediante scritture che seguono precise convenzioni volte a esprimere accuratamente il loro significato, mentre le variabili sono identificate da nomi che dovrebbero essere scelti dal programmatore esaminando con accuratezza le possibili conseguenze sulle sue previsioni di lavoro.

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 il programma che sta scrivendo venga ripreso e ampliato o adattato in tempi successivi al suo primo utilizzp, soprattutto quando si prevede che su di esso intervengano nuove persone.

Il sistema di sviluppo del linguaggio si incarica di assegnare a tutti i dati in un programma le opportune celle-mm 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 è dubbio che questo sia fatto in modo univoco sui tutti i computers e potrebbe essere fonte di rischi.
È quindi consigliabile che il programmatore si preoccupi del valore portato da ciascuna variabile prima di ogni suo utilizzo.

Serve adentrarsi nei dettagli dei collegamenti tra gli identificatori delle variabili e le celle-mm loro assegnate, dettagli che dipendono dal comportamento interno del sistema di sviluppo, solo per programmi per esigenze molto specifiche.
Per la massima parte dei programmi è sufficiente osservare che il sistema di sviluppo nella fase di traduzione del programma organizza una tabella che gli permette di conoscere questa relazione nelle fasi esecutive.
Una elaborazione che gestisce i dati servendosi di indirizzi per un nastro o per la memoria centrale, grazie alle tabelle di collegamento identificatori-indirizzi, risulta equivalente a una elaborazione che per i dati si serve di identificatori.

B70e.07 Veniamo alle regole per il controllo dei dati con il linguaggio c++.

Diciamo **alfabeto degli identificatori** l'insieme costituito dai caratteri alfabetici, dalle cifre decimali e dal segno "_".
Un identificatore si ottiene con 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;
```

Veniamo alla scelta degli identificatori per un programma che prevedibilmente dovrà essere riadattato a varie nuove richieste applicative.
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 l'identificatore va assumendo.
Negli esempi precedenti sono suggerite soluzioni intermedie con stringhe relativamente concise e abbastanza leggibili, in alcuni casi grazie al ricorso alle cosiddette "stringhe a dorso di cammello", stringhe scandibili in sottostringhe grazie alla comparsa di poche maiuscole entro le più numerose minuscole.

B70e.08 Vediamo come possono essere introdotte le scritture 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**.

Le costanti intere possono essere espresse con notazioni posizionali relative a diverse basi.
Con notazioni decimali: 35 23009 -16 -1000000
Con notazioni ottali: `o14` sta per l'intero decimale 12 , `o1000` sta per 512.
Con notazioni esadecimali: `0xc` sta per 12, come l'equivalente scrittura `0XC`.

A ciascuna di queste costanti potrebbero essere assegnate celle-mm di estensioni diverse che ovviamente devono essere più estese per valori assoluti maggiori.

Un intero grande esige una cella-mm estesa, ma si potrebbe chiedere di inserire un intero piccolo in una cella-mm estesa.
Se ad esempio si vuole introdurre una costante intera da assegnare a una cella-64b, cioè se si vuole una costante del tipo chiamato `long integer`, è necessario usare una scrittura literal, ossia una notazione decimale seguita dalla lettera "`L`" o dalla "`l`": per esempio si predispone una cella-64b per il numero 12 scrivendo `12L`.

B70e.09 Sono disponibili le costanti booleane `TRUE` e `FALSE`, equivalenti rispettivamente ai bits 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 (una delle cifre da 0 a 7) e il numero espresso dopo il segno \ rappresenta la posizione del carattere nella sequenza dei caratteri ASCII-8 [:d01].

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	\\	\”

B70e.10 Vediamo alcune frasi di dichiarazione e di assegnazione per variabili dei due 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 dice che i 4 nomi che seguono la occorrenza di `short int`, stringa costituente una cosiddetta **scrittura chiave** o **scrittura riservata**, sono gli identificatori di altrettante variabili intere, ciascuna delle quali richiede 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 $[-2^{31} : 2^{31} - 1]$.

Le frasi precedenti sono dette **frasi dichiarative** ed hanno anche l'effetto di determinare gli indirizzi, ossia le posizioni nella memoria centrale, assegnati alle corrispondenti celle-mm e di predisporre che i rispettivi contenuti nel corso della esecuzion 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 alla sua sinistra.
Le frasi di questo tipo devono comparire dopo le frasi dichiarative delle variabili in causa (queste in genere collocate all'inizio di un programma) e devono comparire prima di ogni altra frase che coinvolga la rispettiva variabile.
Esse sono dette **frasi di inizializzazione**.

B70e.14 Consideriamo il frammento di programma seguente.

```
char lettScrlt, carDL, carrReturn;

lettScrlt=’A’;

carDL=’\044’;

carrReturn=’\r’;
```

Il primo dei quattro 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, che il programmatore è ben lieto di evitare di conoscere, in quanto potrà servirsi semplicemente del 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.

Si possono usare scritture più concise: le precedenti frasi si possono sostituire con la sola seguente:

```
char lettScrlt=’\100’; carDL=’$’; carrReturn=’\15’;
```

B70e.15 In generale si possono unificare le frasi dichiarative ele frasi i inizializzazione che riguardano le stesse variabili.
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 segnala che `short int` e `long int`, le scritture riservate usate più frequentemente, si possono sostituire con le equivalenti ridotte parole chiave r`short` e `long`.

B70 f. arrays e stringhe

B70f.01 Per affrontare moltissimi problemi si rende necessario costruire ed elaborare sequenze di dati omogenei, soprattutto sequenze di datiomogeneri.

Una sequenza di numeri naturali può servire a registrare misurazioni di grandezze ottenute in osservazioni successive come cardinali di insiemi, lunghezze, pesi, durate, velocità, quantità di denaro, temperature di un paziente, concentrazioni, percentuali, ...
Sequenze di interi consentono di fornire i numeri degli abitanti di una sequenza di città o di nazioni, le altitudini in metri di località incontrate in un dato percorso, le quantità di un prodotto fabbricato in anni successivi, i punteggi delle squadre partecipanti a un campionato, ...

Oltre alle sequenze numeriche risultano utili le 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.
Questo tipo di posizionamento di informazioni si può naturalmente accostare alla registrazione di dati omogenei sopra un segmento di un nastro di carta, di un nastro magnetico o di una traccia di un disco magneto-ottico.

Come si è detto in ??, ogni sequenza di celle-mm è individuata dall'indirizzo della prima e dal numero delle celle, cioè dal numero delle componenti della sequenza e un modo canonico per individuare una di queste celle si serve dell'indirizzo della cella iniziale sommato del numero di celle sulle quali si deve avanzare partendo dalla iniziale per giungere a quella richiesta.

B70f.02 Nel linguaggio C++, come sostanzialmente in tutti i linguaggi di programmazione procedurali, le sequenze in memoria sono controllate dagli **arrays**, le più semplici tra le strutture informative.
Si tratta di variabili collettive, strutture di dati per la registrazione in memoria di sequenze e altri schieramenti di dati che consentono al programmatore di accedere facilmente alle loro componenti, ossia senza preoccuparsi della loro precisa collocazione nella memoria fisica, della quale, come si è detto, si fa carico il sistema di sviluppo).
Con C++ si possono trattare arrays di una, due o più dimensioni; ora 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 possono trattare denominazioni di al più 25 caratteri il cui numero attuale è 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 (visualizzabili nella gran parte delle applicazioni pratiche) e che ciascuno degli interi sia registrato in 32 bits consecutivi e quindi possano assumere tutti i valori appartenenti all'intervallo $\mathbf{[}-2^{31} : 2^{31}-1\mathbf{]}$.

B70f.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 con conseguenze simili: ciascuna di esse ha l'effetto di determinare il contenuto di un byte fornito da una **scrittura di costante**, quella che compare a destra del segno "`=`", e di assegnarlo come valore attuale a una cella-mm determinata dalla scrittura alla sua sinistra.

Va osservato che il segno "`=`" non esprime una relazione di uguaglianza e non propone una equazione, come fa quando compare in una formula matematica; esso infatti richiede una azione consistente nella assegnazione di un nuovo valore ad una variabile, cioè nella modifica del contenuto di una cella-mm.
Possiamo dire che "`=`" esprime una operazione di assegnazione e va assimilato al connettivo "`:=`" usato spesso quando si definisce una espressione matematica e talora anche per introdurre in un discorso un termine che si vuole usare specificamente.

La prima delle precedenti frasi 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`.

La frase `denom[4] = 'u';` comporta invece 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 raggiunge avanzando di 4 posizioni rispetto alla prima assegnata all'array, la quale è 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 nel linguaggio C++ ([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.
Una scrittura come `denom[4]` comporta la valutazione di un valore numerico (in questo esempio 4, ma tra le parentesi quadre si potrebbe avere anche una espressione aritmetica piuttosto elaborata), 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 gli altri tipi di celle-m.

B70f.04 Se vogliamo disporre dei primi 10 numeri della `successione di Fibonacci (wi)` nelle prime 10 componenti dell'array che ha come identificatore `Fib` , 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 "`;`" il quale ha il ruolo di separatore di frasi, comporta l'assegnazione del valore fornito dalla scrittura decimale al secondo membro alla cella associata alla data componente di array; le successive frasi le successive coinvolgono le prime 10 celle associate all'array `Fib`.

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 } ;
```

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

B70f.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.

(3) Eserc. Porre nelle 20 componenti di un array i pesi atomici arrotondati dei 20 elementi chimici più leggeri.

B70f.06 Le sequenze di caratteri, cioè le stringhe, rivestono grande importanza nella elaborazione dei dati.
Innanzi tutto le stringhe di caratteri visualizzabili sono necessarie per affrontare le analisi computazionali delle informazioni esprimibili nei linguaggi naturali: nomi e qualificazioni di persone, oggetti di ogni genere, prodotti, slogan, norme, ricette, tabelle di dati, sonetti, trascrizioni di registrazioni, testi di ogni genere.
Ogni testo scritto o comunque registrato può essere ridotto a una stringa (eventualmente ricorrendo ad Unicode [:d05, :d06].
Vi sono poi le informazioni che vengono espresse mediante linguaggi convenzionali e artificiali: espressioni matematiche, enunciati della logica, formule chimiche, sequenze biologiche, ricette, norme d'uso di appareecchiature, ...

Un vasto campo applicativo delle stringhe riguarda la elaborazione automatica dei testi di ogni genere e in particolare dei testi che sono utilizzati nei prodotti software di più evidente utilità: i testi sorgente dei linguaggi di programmazione e di gestione delle basi dati, i testi in grado di governare il tracciamento di figure i testi dei linguaggi per la tipografia, per la grafica e per la comunicazione; alcuni esempi: TEX (we), HTML (we), XML (we) ed SVG (we).

B70f.07 Riconosciuta l'importanza delle stringhe, i linguaggi C e C++ hanno reso disponibili vari strumenti per il trattamento delle 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 32 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 lorsignori possono \
```

```
constatare direttamente, viene scritta su tre linee del file .txt che fa \
da supporto per il testo davanti ai vostri occhi."
```

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 occorrenze del carattere new line rappresentato da "\n" se inviato a una stampante comporta la emissione di più linee.

String literals nei quali compaiono molti caratteri di controllo possono avere effetti di stampa o di emissione su video piuttosto elaborati. Con questi literals vengono generati i disegni che costituiscono la cosiddetta `ASCII art (we)`.

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. Le stringhe che presentano un null finale qualche vantaggio pratico e sono chiamate **null teminated strings**.

B70**f.08** 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, ottenibili i cosiddetti numeri real che vedremo più avanti, le quali non sono in grado di soddisfare certe esigenze e di risolvere certi tipi di problemi; ad esempio sono del tutto inutilizzabili per trattare problemi di natura crittografica.

Ribadiamo che i limiti finiti degli strumenti effettivi, ovviamente, devono essere tenuti ben presenti nella pratica computazionale.
Tali limitazioni nello sviluppo di considerazioni generali, viceversa, possono essere considerate quasi un intralcio alle formulazioni rapidamente comprensibili di molti enunciati e di molte argomentazioni. 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 g. operazioni di lettura e scrittura [1]

B70g.01 La massima parte dei programmi prevede la possibilità di leggere i dati concernenti una istanza del problema da risolvere da un'apparecchiatura di ingresso e la possibilità di scrivere i risultati su un dispositivo di uscita.

Nel caso di dati immessi da un operatore che si serve di una tastiera è opportuno far precedere il suo intervento da richieste esplicite.
Inoltre in genere è opportuno accompagnare i risultati inviati a un destinatario umano o artificiale con segnalazioni sufficientemente chiare dei loro significati.
Per certe applicazioni elaborate in risultano opportune anche segnalazioni emissioni che chiariscano il contesto nel quale si sono ottenuti i risultati; a tale chiarimento possono servire tutti i dati di ingresso, i dati critici contenuti nella macchina, alcuni dati intermedi che possono far capire lo svolgimento delle manovre eseguite talora alcuni dati acquisiti dall'esterno nel corso dell'elaborazione.

Dei molteplici dispositivi di ingresso e uscita oggi utilizzabili dai sistemi di calcolo qui ne prendiamo in considerazione solo pochi tipi.

Innanzi tutto consideriamo letture da nastri, dischi e memorie flash preregistrate e scritture su nastri, dischi o memorie flash che verranno elaborate in un secondo tempo.
Per le operazioni di ingresso e uscita, ossia per le **operazioni I/O**, ci limitiamo a trattare quelle che riguardano solo files di dati sequenziali descrivibili come stringhe di bytes; per un trasferimento di dati verso e dal computer parliamo di flusso di bytes e più tecnicamente di **input stream** in lettura e di **output stream** in scrittura.

Nelle prossime pagine ci occuperemo principalmente di letture e scritture di files sequenziali simbolici contenenti stringhe ASCII leggibili in chiaro.

Un tale file può servire solo per la lettura o solo per la scrittura e questa operazione può essere effettuata con una sola manovra oppure in fasi successive.

B70g.02 In molti programmi semplici si prevede una sola lettura iniziale dei dati, un cmplesso di elaborazioni di informazioni in memoria e una sola scrittura finale dei risultati.
In generale invece in un programma letture, elaborazioni e scritture possono essere alternate e dipendere dai dati letti e dai risultati intermedi.

In una elaborazione può accadere che un file sequenziale venga letto inizialmemnte e successivamente venga esteso con un flusso di dati prodotti dalle operazioni programmate.
Un file sequenziale a disposizione del programma potrebbe essere utilizzato per ospitare dati intermedi, oppure potrebbe essere letto e riletto più volte.
Queste evenienze si verificano nel caso di penuria di spazio di memoria; erano comuni nei sistemi del passato ma ora con la disponibilità di memorie molto più estese e con la possibilità di ricorrere a sistemi remoti come depositi di dati interessano solo per programmi destinati a dispositivi molto piccoli contenuti in apparecchiature con compiti come il controllo di veicoli o di sonde nello svolgimento di missioni impegnative.
Qui invece ci occuperemo primariamente di elaborazioni che non incontrano vincoli fisici rilevanti, ma riguardano solo manovre che richiedono operazioni ben definite a priori.

B70g.03 Qui prestiamo una precisa attenzione verso apparecchiature costituite da terminale video e tastiera e che consideriamo servano ad immettere o emettere flussi leggibili e più precisamente stringhe ASCII.

Tastiera e video in genere vengono utilizzate insieme e spesso in modo interattivo per scambi bidirezionali di stringhe leggibili, di solito di lunghezza contenuta, in deteminati momenti di una esecuzione.

Va segnalato che di solito lo schermo del video serve anche per mostrare tutto quello che viene digitato sulla tastiera.

Anzi, quando si opera in quella che viene detta **modalità assistita** spesso l'immissione dei dati viene sollecitata da richieste auspicabilmente chiare e spesso viene aiutata con la presentazione dopo ogni rchiesta di liste di alternative possibili, con il completamento di stringhe prevedibili e con proposta di stringhe giudicate correzioni di quelle immesse e considerate imprecise.

Inoltre il video terminale può dare accesso a testi che possono provenire dall'universo Internet, testi che possono essere utilizzati più o meno direttamente per una elaborazione che l'opratore alla tastiera controlla in tempo reale, ossia mentre si sta svolgendo.

Delle operazioni di entrata/uscita prenderemo in esame solo i tipi più semplici e non entreremo in molti dettagli tecnici; ci dedicheremo primariamente a descrivere il funzionamento e l'utilizzo di pochi sottoprogrammi ai quali demanderemo tutte le manovre di immissione ed emissione.

Occorre precisare che non prevediamo di avere immissioni ed emissioni di sequenze di caratteri leggibili; che nelle stringhe elaborate possono essere presenti anche caratteri ASCII non leggibili incaricati in lettura di presentare le stringhe come costituite da records successivi e in uscita incaricati di organizzare le stringhe come linee successive costituenti pagine successive.

B70**g.04** In questa e nelle prossime sezioni dunque prenderemo in considerazione solo programmi piuttosto semplici ciascuno dei quali, in buona sostanza, presenta richieste di lettura di dati iniziali, un successivo complesso di elaborazioni su informazioni numeriche e simboliche e richieste di scrittura dei risultati ottenuti.

Le informazioni lette saranno costituite da stringhe ASCII che risulta necessario trasformare in corrispondenti dati interni che saranno solo numeri interi da registrare canonicamente in celle-32b.
Per ottenere scritture finali saranno necessarie trasformazioni di numeri interi e stringhe ASCII in stringhe che saranno visualizzate come linee e pagine nell quali si hanno i risultati corredati di qualche commento esplicativo.

Le prime trasformazioni annunciate, in relazione al programma le diciamo **operazioni di codifica dei dati**, le ultime **operazioni di decodifica dei risultati**.

Ci proponiamo di fornire ai programmi files simbolici sequenziali che possono essere preparati agevolmente con uno dei comuni source editors, e files da emettere che possono essere visionati come pagine stampate oppure che possono essere eleborate con un source editor interattivo tramite video che possa essere usato agevolmente per ritoccare i files emessi prima di una loro archiviazione o per adattarli, eventualmente ricorrendo ad altri files in qualche modo compatibili, ai fini di loro successivi utilizzi.

In effetti i files sequenziali simbolici costituiscono un mezzo comodo e facilmente controllabile per passare informazioni da un programma ad uno che si può vedere collocato in una posizione successiva in uno schema di un progetto che si serve significativamente di testi leggibili riguardanti dati interessanti per le finalità applicative del progetto, ad esempio dati che servono ad organizzare e governare una attività articolata, immaginabile come una delle molte attività che traggono vantaggio dall'essere supportate sistematicamente da adeguati programmi.

Per denotare questi sistemi di programmi operativamente collegati si usa il termine **programmi da utilizzare in cascata**.

B70g.05 I programmi massimamente semplici non fanno altro che emettere un messaggio su un dispositivo di uscita scelto come strumento 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 questa 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`, gli unici elementi del linguaggio necessari per formulare l'emissione.
La seconda linea costituisce l'intestazione del programma e il suo testo, detto anche corpo del modulo di programma, segue presentato tra una parentesi graffa aperta e una chiusa; queste costituiscono la coppia dei delimitatori coniugati dl testo.
Nella prima linea del testo il comando `cout` richiede l'emissione della stringa prefissata che segue racchiusa tra due doppi apici a sua volta seguita dal carattere `endl` che determina la fine della linea stessa.
La seconda linea del programma è la semplice richiesta di conclusione delle operazioni e si configura come richiamo di un sottoprogramma, un importante tipo di protagonista della programmazione che esamineremo più avanti.

B70g.06 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 interi 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(); // chiede approvazione di fine run
    return(0);
}
```

La frase che inizia con `cin` prevede la immissione da un dispositivo scelto come dispositivo di lettura standard 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 dagli identificatori dalle variabili con le quali vengono individuati nel programma stesso; questi identificatori hanno lo scopo di chiarire il significato di ciascuno dei numeri.
Queste scritture esplicative qui appaiono banali, ma nei programmi che forniscono numerosi risultati hanno grande utilità, rendono il programma uno strumento agevole all'uso e meritano attenzione.

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 conclusiva `return(0);` .

Occorre osservare che la scritta che segue l'enunciato `getch();` costituisce un commento al programma.

B70g.07 Il programma precedente prevede un semplicissimo dialogo interattivo, ossia uno scambio di informazioni tra utente del programma e processo esecutivo; qui l'utente può solo l'istante nel quale effettuare l'esecuzione.

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.

Dei dati erronei, ossia incoerenti con gli obiettivi del programmatore, potrebbero avviare elaborazioni del tutto inutili o, peggio, elaborazioni che illudono sul significato dei risultati ottenuti.
In effetti le azioni che si devono effettuare per garantire la bontà dei dati immessi richiedono controlli e validazioni che si possono effettuare solo utilizzando i comandi di selezione ed iterazione che vedremo, rispettivamente, in B70h e in B70i.
Inoltre la lettura di dati articolati e soggetti a vincoli di coerenza richiede validazioni che seguono logiche che vanno precisate con attenzione alle caratteristiche dei significati applicativi dei dati e in genere conviene che tali controlli siano organizzati con il richiamo di appositi sottoprogrammi.
Queste questioni saranno riprese nella sezione B70j.

B70g.08 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, cifre, ",", "'", spazi bianchi.
Vi sono però vari caratteri ASCII, visualizzabili e non, 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 di vvertimento
`\\` backslash
`\?` question mark, punto interrogativo
`\'` singolo apice
`\"` doppio apice
`\0` NUL, byte di 8 bits nulli
`\`*o*, `\`*oo*, `\`*ooo* rappresentazione di un byte mediante 1, 2 o 3 cifre ottali
`\x`*h*, ... , `\x`*hhhh* rappresentazione di informazione mediante 1, 2, 3 o 4 cifre esadecimali, ossia mediante 2, 4, 6 o 8 bits

B70g.09 Il linguaggio C++ 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 [:h10];
- `--` operatore unario di decremento per un operando intero che può essere usato come prefisso e come suffisso [:h10].

B70g.10 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 retto 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.

B70g.11 Vediamo alcuni esempi di espressioni numeriche molto semplici.

L'espressione `5+7*3` implica per prima cosa il calcolo di 21 (`*` ha la precedenza su `+`) e quindi il calcolo di `5+21` e la messa a disposizione del contesto, cioè della posizione 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` che richiede di sommare 5 e 7 e successivamente di moltiplicare 12 per 3 ottenendo il valore 36 da rendere disponibile al contesto.

`-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.

B70g.12 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)` rende disponibile 36;

`144/8` fornisce 18; `144/8/2` vale 9, mentre `144/(8/2)` produce 36.

`-6+44/11` rende disponibile `-2`, come l'eqivalente `-6+(44/11)`.

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

Solo 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++ (e nel 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 `(k/h)*h+(k%h)`.

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

Ad esempio `12%5+40%7*23%8` cacola `2+5*7` e in conclusione fornisce 37.

B70**g.13** 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," ." << endl ;
```

comportano l'emissione della linea

```
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**g.14** **(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 il calcolo dell'area di un triangolo con i vertici caratterizzati da coordinate intere pari.

(5) Eserc. Esprimere le richieste per il calcolo dell'area di un quadrilatero con i vertici caratterizzati da coordinate intere pari.

(6) Eserc. Esprimere le richieste per il calcolo dell'area di un poligono con 5 lati con i vertici caratterizzati da coordinate intere pari.

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

B70**g.15** Come si è detto, nei linguaggi C e C++ si trattano valori booleani, cioè valori interpretabili come true e false, valori la cui importanza è data dalla possibilità di utilizzarli, vedremo in :h e in :i, per decidere le scelte esecutive sulle quali si basano la versatilità, la adattabilità e la portata applicativa dei programmi.

I due valori di verità true e false sono implementati attraverso valori interi e questo consente di programmare operazioni numeriche sopra valori ottenuti valutando espressioni booleane e viceversa di prendere decisioni di ampia portata basandosi su valori forniti da espressioni numerich e più in generale, su valutazioni quantitative riguardanti dati che possono essere numerosi ed etereogeneri.

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.

B70g.16 C, C++ e tutti i linguaggi procedurali dispongono di operatori relazionali su numeri, operatori binari che prevedono due operandi numerici e forniscono un valore che può essere usato sia come valore booleano, che come valore intero binario.

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

Sono anche disponibili gli operatori logici, operatori binari o unari che prevedono operandi logici e risultato binario.

- `&&` operatore binario AND; fornisce true, 1, sse entrambi gli operandi valgono true;
- `||` operatore binario OR; fornisce false, 0, sse entrambi gli operandi assumono il valore false;
- `!` operatore unario prefisso esprimente negazione, NOT; scambia i valori true e false, o equivalentemente lo scambio tra 0 e 1.

Per quanto riguarda le precedenze esecutive, prevale l'operatore unario !, seguono gli operatori relazionali su numeri e l'operatore `&&`, per ultimo venendo `||`; occorre aggiungere che gli operatori numerici hanno la precedenza sui relazionali come questi precedono i 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.

B70g.17 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 dei valori interi ai quali il compilatore assegna una cella-4B.

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 $\mathcal{I}_{\mathbb{Z}}[\mathbf{(} : 0\mathbf{)} \cup \mathbf{[}10 : \mathbf{)}]$ (consideriamo scontata la limitatezza della portata delle celle-4B).

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.

B70g.18 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 $[0 : 127]$.

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

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

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 h. strutture di controllo selettive

B70h.01 Nelle linee di programma considerate finora sono intervenuti solo pochi e ben definiti componenti elementari del linguaggio: dati, dichiarazioni, variabili, espressioni, sequenze di pochi comandi: assegnazioni, semplici frasi di I/O e richiami di sottoprogrammi con un solo compito ben preciso.
Ora dobbiamo occuparci della redazione di programmi un po' più elaborati e con compiti un po' più compositi.

Cominciamo con il precisare che con il termine **controllo del programma**, in breve **controllo-p**, intendiamo un dispositivo che risulta lecito descrivere in termini antropomorfi come colui che dirige o controlla l'esecuzione di ogni comando espresso nel programma muovendosi sulle sue frasi per far eseguire dai meccanismi interni del sistema hw-sw computer le operazioni che sono implicate dalle frasi che raggiunge nei successivi passi esecutivi.

Per far eseguire le frasi che compaiono nei programmi visti in precedenza precedenti al controllo bastava muoversi procedendo dalla prima all'ultima.
Dato che le operazioni da eseguire per risolvere ogni istanza di problema risolvibile in tempi finiti sono in numero finito si potrebbe pensare che ogni problema potrebbe essere risolto con un controllo che si muove avanzando.
Si obietta che questi programmi possono affrontare solo problemi con istanze che richiedono elaborazioni strettamente simili e quindi hanno una portata molto ridotta, in quanto risulta evidente che tutti i problemi impegnativi riguardano situazioni che possono presentare forti differenze.
In particolare la maggior parte dei problemi impegnativi riguarda insiemi di istanze che richiedono svolgimenti differenti e anche a priori difficilmente prevedibili, sui quali un automatismo non può nutrire incertezze.

Procedimenti e programmi che intendono essere ampiamente applicabili devono essere in grado di distinguere le istanze dissimili e di applicare ad essi manovre diverse; questa capacità di distinzione delle situazioni da affrontare e di scegliere fra successive percorsi operativi diversi costituisce anche un requisito che devono avere i procedimenti e i programmi che aspirano a essere efficaci e a potersi adattare a istanze nuove attraverso semplici ritocchi.
In altre parole quest'ultima qualità riguarda la possibilità di avere programmi in grado di evolversi.

B70h.02 La distinzione delle situazioni che si incontrano deve essere determinata da parametri facenti parte dei dati iniziali o calcolabili a partire da essi; un linguaggio di programmazione capace della suddetta distinzione quindi deve disporre di frasi che consentano di scegliere tra due successive linee di comportamento dall'esame dei parametri disponibili.

Una progressiva crescita delle esigenze dei problemi computazionali che si vogliono risolvere risulta evidente anche dalla storia di tante società in tanti periodi, soprattutto nei tempi più recenti, i più sollecitati ai cambiamenti indotti dal crescere delle tecnologie e dai conseguenti cambiamenti di esigenze e di prospettive.

La crescita delle esigenze computazionali riguarda in particolare la possibilità di sottoporre a elaborazioni automatiche grandi quantità di dati simili: questo è del tutto evidente per le problematiche che richiedono analisi statistiche su popolazioni numerose e sulle problematiche che richiedono valutazioni numeriche ottenibili procedendo per approssimazioni successive.

Per i programmi si tratta della possibilità di far ripetere più volte l'esecuzione di un'intera sequenza di frasi e, come vedremo più in generale, di far ripetere più volte quelli che definiremo "blocchio di frasi".

In altri termini diciamo che è necessario che il controllo possa muoversi, oltre che progressivamente, anche in seguito a proprie scelte e che possa saltare verso le sequenze di frasi (e verso i blocchi esecutivi) che ha stabilito essere gli unici adatti (o i più adatti) alla situazione corrente cheè stato in grado di analizzare quantitativamente e/o qualitativamente.

Si vuole dunque che il controllo possa effettuare anche salti all'indietro per far ripetere azioni e scelte che tengono conto degli effetti delle azioni più recenti.

Si vuole in particolare la possibilità di organizzare iterazioni di sequenze di frasi esecutive, manovre che in particolare consentono l'esplorazione, l'utilizzo e il cambiamento di situazioni che talora si possono raffigurare distribuite come punti di una figura geometrica dotata di regolarità.

Va subito segnalato che si possono avere iterazioni con numeri di ripetizioni definiti in partenza e iterazioni la cui ripetizione solo in conseguenza di dati raccolti dopo ciascuna ripetizione; questo accade in molti calcoli per approssimazioni successive.

Conviene anche far notare che la possibilità del controllo di saltare verso frasi qualsiasi gli consente di ottenere esecuzioni con movimenti ben più articolati delle ripetizioni di sequenze esecutive.

B70**h.03** Per organizzare procedure per esecuzioni non solo sequenziali potrebbero bastare due tipi di istruzioni estremamente semplici (sono quelle adottate nei più elementari linguaggi di macchina): l'istruzione di salto condizionato e l'istruzione di salto incondizionato.

La prima segue il seguente schema:

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

Qui **clausola** rappresenta 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**, o in inglese label, è una scrittura che serve a identificare una e una sola frase esecutiva del programma come possibile meta una o più istruzioni di salto. Ciascuna di queste la chiamiamo **frase bersaglio di salti**

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, segue il semplice schema

`salta alla frase` *etichetta*

e palesemente provoca il salto del controllo alla frase contraddistinta da *etichetta* invece che alla frase successiva.

Nel linguaggio C++ queste frasi assumono, rispettivamente, le forme

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

`goto` *etichetta*

Occorre aggiungere che nel linguaggio C++ una etichetta è un identificatore e va posta prima della corrispondente frase bersaglio seguita da un segno "`:`".

In seguito presenteremo le ragioni per le quali è buona norma, sia nei programmi C++ che e negli altri linguaggi procedurali, utilizzare solo in pochi casi ben motivati le frasi bersaglio e le etichette.

B70h.04 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 potevano programmare gli elaboratori elettronici disponendo solo dei linguaggi di macchina (dal 1945 al 1955 all'incirca) e dei primi linguaggi procedurali (il Fortran associato al nome di Backus e il COBOL della McMurray) avvalendosi degli accennati suddetti dispositivi di salto sono stati scritti parecchi programmi efficaci e di ragguardevoli dimensioni che sono riusciti a far capire a non pochi le prospettive del calcolo automatico.

Si può anche ricordare che prima dei computer pionieristici erano stati costruiti automatismi puramente meccanici ed elettromeccanici con notevoli prestazioni. Ci limitiamo ad citare gli automi a controllo idrico e pneumatico dell'epoca ellenistica, le calcolatrici di Pascal e Leibniz e i giocatori di scacchi del secolo XVIII.

Un'altra idea che qui ci limitiamo a tratteggiare, ma che va approfondita, riguarda il fatto che questi dispositivi di salto consentono di organizzare la 'totalità" delle elaborazioni deterministiche concepibili.

Va tuttavia segnalaro 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 possono condurre ad esecuzioni che, dopo un numero anche molto elevato di passi esecutivi, non riescono a concludersi e lasciano il dubbio se possano portare a un risultato utile.

Questo comporta la necessità di affrontare un nuovo problema: quello del garantire che un programma non incorra nella possibilità di non arrestarsi mai.
Questo problema può porsi per un particolare programma o per una classe di automatismi e in questo secondo caso viene detto "problema dell'arresto del tipo di automatismo".

Va anche considerato che con programmi a esecuzione non solo sequenziale si giunge a utilizzare strumenti che aprono la possibilità di un infinito potenziale.
È assodato che si vogliono controllare attività concrete, e quindi definite finitamente e che dovrebbero concludersi in tempi finiti con l'impiego di risorse finite; ora si propone di farlo servendosi di strumenti che in linea di principio possono portare avanti le loro operazioni illimitatamente.
In altre parole si adottano strumenti con capacità operative potenzialmente infinite e questo comporta nuovi problemi, a cominciare da quello dell'arresto.
Evidentemente sarebbe stato ingenuo, e anche illusorio, sperare di sfruttare appieno gli automatismi e la loro autonomia senza dover incontrare nuovi seri problemi.

B70h.05 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, nuovi dispositivi, ampliamenti delle applicazioni, crescita del numero delle macchine, nascita degli studi sulla programmazione e crescita dell'importanza sociale del mondo dell'informatica.
Inoltre si è avuto il prevedibile aumento dei neologismi (software) e della indignazione di molti puristi della nostra lingua.

La crescita della programmazione ha riguardato sia il il numero dei programmi richiesti e messi a punto, sia il numero degli addetti, sia le metodologie della produzione e del mantenimento dei programmi, sia le aspettative riposte nelle soluzioni computerizzate da tanti ambienti, a cominciare dagli amministrativi, dai produttivi, dai militari e dal modo della ricerca.
Intorno al 1960 la programmazione quindi ha cominciato a rivestire notevole importanza culturale, sociale e politica in tutti i paesi industrializzati.

Negli anni successivi, in seguito all'esigenza di disporre di programmi sempre più estesi, incisivi e affidabili e al naufragio di molti progetti, si è andata imponendo l'esigenza di attività di programmazione più consapevoli e più disciplinate.

Sempre più spesso si è reso 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 estesi ed eterogenei, vuoi per aumentare la precisione e la attendibilità dei risultati quantitativi.

A questo punto, poco dopo l'introduzione del termine "software", si è iniziato a prendere in considerazione l'intero **ciclo di vita del software**.

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, scalabilità, esauriente documentazione e quindi leggibilità dei relativi testi sorgente.

B70**h.06** In quel periodo di crescita ci si è accorti che i programmatori erano pochi, che erano portati a seguire abitudini personali spesso estemporanee, che poco si preoccupavano della documentazione dei programmi e della leggibilità dei loro testi sorgente e che poco si interrogavano sui criteri di organizzazione dei loro programmi e poco si confrontavano con colleghi che avevano il compito di rielaborare i relativi testi sorgente.

Più specificamente si è osservato che i programmi nei quali comparivano numerose frasi i `goto` spesso risultavano difficilmente comprensibili anche dagli stessi autori incaricati di riprenderli qualche tempo dopo la loro utilizzazione.
Inoltre si osservava quanto fosse oneroso e privo di linee guida il lavoro per il controllo della correttezza dei sempre più numerosi programmi che si riteneva necessario sottoporre a continui aggiornasmenti.

In effetti il crescere della domanda di eleborazioni automatiche comportava spesso la richiesta di ritoccare e ampliare, spesso con urgenza, le prestazioni di programmi in uso con l'aggiunta di porzioni di altri programmi, compito che spesso si otteneva con l'aggiunta di qualche `goto` verso un blocco di frasi individuato affrettatamente.

In genere questi `goto` avevano bersagli in posizioni poco facili da individuare e i loro effetti complessivi non venivano analizzati con sufficiente completezza. Inoltre quando venivano attuate successive modifiche il controllo e la strategia della organizzazione complessiva delle elaborazioni possibili tendevano a a deteriorarsi progressivamente.

B70**h.07** 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 di salto rimpiazzandole con le strutture di selezione e con le strutture di iterazione che vedremo tra breve.

Si sono quindi progressivamente imposte le accennate strutture di controllo e modalità per la loro organizzazione secondo una certa rigidità ma che, se adottate come criteri finalizzati alla programmazione disciplinata, portano alla disponibilità di programmi più leggibili, più controllabili, piëstendibili e più riutilizzabili.

Queste qualità evidentemente sono a sostegno della prospettiva di attività di programmazione sempre più sistematiche, con obiettivi sempre più ampi e con maggiori possibilità di progressiva evoluzione.

Questo modo di organizzare la programmazione è stato chiamato **programmazione strutturata** e anche di *goto-less programming*.

In seguito cercheremo di seguire diligentemente le prescrizioni della programmazione strutturata.

Tuttavia riteniamo che gli enunciati `goto` in alcune rare situazioni che si possono caratterizzare abbastanza bene consentono soluzioni chiare e poco rischiose re che in questi casi qualche deroga dalla programmazione strutturata può essere conveniente.

B70h.08 Nel corso di un'elaborazione può accadere che il controllo debba scegliere di affrontare diverse azioni sulla base dei valori attuali di alcuni parametri variabili.
In queste circostanze si dice che si devono organizzare **selezioni** tra diverse linee di azione.

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 che 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 sono prevedibili estensioni del precedente.

B70h.09 Seguendo la programmazione strutturata per la scelta di evitare una azione si adotta il costrutto che segue.

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

Una scelta tra due alternativa, chiamata anche scelta dicotomica o dilemma, si presenta come segue.

```
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**h.10** Veniamo ora alla preannunciata nozione di blocco, che risulta centrale per la programmazione strutturata.

Per blocco si intende un segmento di programma delimitato con chiarezza, in particolare delimitato tra due parentesi graffe coniugate.

Conviene poi distinguere vari tipi di blocchi.
I blocchi del primo tipo sono i moduli di programma caratterizzati da una premessa e da un corpo delimitato da una coppia di parentesi graffe.
Definiamo poi i blocchi primari o di livello 1 i blocchi interamente contenuti in un modulo (al quale si potrebbe attribuire il livello 0).
Abbiamo infine i blocchi di livello superiore ad 1 che sono porzioni di programma interamente contenuti in blocchi diverso da loro modulo; a questi blocchi si attribuisce il livello ottenuto aumentando di uno il livello del blocco che lo contiene.
Si dice anche che un blocco di livello 2 o superiore è **sottoblocco** di quello che lo contiene; questo si può chiamar "sovrablocco" di ciascuno dei sottoblocchi che contiene.

In un modulo di programma quindi si può riconoscere una struttura di arborescenza distesa [??] la cui radice è il modulo stesso e i cui archi collegano ciascun blocco ai suoi sottoblocchi.

B70**h.11** Aggiungisamo altre caratteristiche dei blocchi.

Si possono avere sia blocchi semplici costituiti da una sola frase esecutiva, sia blocchi costituiti da una sequenza di frasi formalmente autonome, sia blocchi variamente elaborati possibilmente dotati si sottoblocchi che si basano su costrutti selettivi come quelli introdotti sopra [:h09], su altri costrutti selettivi e su costrutti iterativi.

I blocchi possono contenere anche dichiarazioni (con eventuali inizializzazioni) di variabili che hanno uno **scope**, cioè un ambito di validità, ossia di visibilità e operatività, limitato al blocco stesso.

Va detto anche che per un blocco semplice a una sola frase le parentesi graffe che lo delimitano possono essere tralasciate, in quanto possono 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`.
Le caratteristiche strutturali dei blocchi contribuiscono a renderli delle unità operative dotate di rilevante autonomia, simile a quella dei sottoprogrammi.
come vedremo questo porta notevoli vantaggi ale attività di programmazione.

La stesura nel testo sorgente dei costrutti strutturati è opportuno sia realizzata seguendo sistematicamente dei criteri di collocazione delle scritture riservate 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 molti campi applicativi si verifica di frequente e talora è prevedibile.

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 che possono essere preventivati e pianificati.

Infine va aggiunto che i testi ben strutturati si possono presentare abbastanza agevolmente mediante diagrammi di flusso o mediante altre tecniche di visualizzazione che vengono accuratamente studiate nell'ambito dell'ingegneria del software, una disciplina di indubbia importanza industriale ed economica.

B70**h.12** 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 doveroso meccanismo per questa scelta viene implementato dal costrutto rappresentato dagli schemi che seguono (nei quali 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 facilmente intuibili si adottano per 4 o più possibili scelte.

In questo genere di circostanze occorre esaminare l'insieme dei possibili successivi percorsi operativi e ripartirlo in una sequenza di sottoinsiemi di successivi percorsi, ciascun componente della quale sia caratterizzato da un parametro disponibile sul quale si procede a stabilire con il costrutto `if ... then` se imboccare il corrisponebte percorso; l'ultimo sottoinsieme di possibili successivi percorsi è il complementare dell'unione dei precedenti e porta al blocco da far seguire alla occorrenza di `else`, altrimenti.
Si noti che `else` è un esempio di una cosiddetta **parola chiave** o **parola riservata** .

Se tutte le parti dell'insieme dei possibili percorsi successivi sono chiaramente associate a valori di parametri discriminanti conviene organizzare costrutti selettivi privi della possibilità preceduta dalla semplice chiave `else` in modo da avere tutte le vie da selezionare caratterizzate da clausole esplicite chiaramente riconoscibili.

Gli schemi di questi costrutti riguardanti 2 e 3 possibilità (in B70h08 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 valeclausola 2
      }

if(clausola 1) {
```

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

B70h.14 Presentiamo, ancora sotto forma di frammenti di programma ridotti all'essenziale, alcuni esempi di costrutti con qualche sentore applicativo.

```
// Associa al progressivo del mese il numero dei suoi giorni
in un anno nonbisestile
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 << "deve ripetere prova scritta" << endl ;
else if(vps < 18) {
    cout << "si consiglia di ripetere prova scritta" << endl ;
    }
else if(vps < 24) cout << "presentarsi alla prova orale << endl ;
else {
    cout << "la sola prova scritta comporterebbe il voto 25/30" << endl ;
    cout << "per un voto migliore presentarsi alla prova orale" << endl ;
    }
```

B70h.15 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 organizzare 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 un punto $\langle x, y\rangle$ del piano sugli interi ad una delle 9 parti di questo insieme determinate dagli assi.

```
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**h.16** **(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 scriva un frammento di programma che sia in grado di porre in ordine due, tre oppure quattro numeri interi positivi, presupponendo che i numeri sono immessi successivamente e sono seguiti dalla immissione di uno 0 conclusivo; va segnalato anche il caso non voluto di immissione di 5 numeri positivi. .

(4) Eserc. Si scriva un frammento di programma in grado di porre in ordine alfabetico tre sigle automobilistiche, o più in generale tre digrammi, ossia tre stringhe di due lettere.

(5) Eserc. Redigere un frammento di programma nel quale si esaminano quattro numeri interi e si distinguono i casi nei quali vi sono delle ripetizioni.

B70 i. strutture di controllo iterative

B70i.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 a mente o servendosi di una semplice calcolatrice numerica meccanica, elettromeccanica o elettronica.

B70i.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 se 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 ragionevolmente 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, le quali 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; tuttavia ancora questi delimitatori possono essere evitati per i blocchi costituiti da una singola frase.

Questi blocchi più specificamente li chiamiamo **blocchi iterativi** e chiamiamo **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 i blocchi iterativi devono avere una sufficiente versatilità e non devono essere soltanto pedissequamente ripetitivi, ma si deve avere la possibilità di cicli contenenti tutti gli elementi variabili che risultano convenienti.

Per la versatilità dei cicli si possono adottare vari accorgimenti.

Innanzi tutto nei cicli possono comparire operandi che cambiano nelle successive esecuzioni.

Un primo modo per ottenere questo consiste nel porre nel blocco operazioni di lettura che forniscono nuovi dati a ogni nuova esecuzione del blocco stesso, ossia in ciascuna di quelle che chiamiamo **esecuzioni cicliche** o **cicli [esecutivi]** del blocco; talora si usa il termine azione da reiterare.

Un secondo modo di procedere 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 al 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.

B70i.03 Si impone quindi la necessità didisporre inserire in un linguaggio di programmazione dei **costrutti iterativi**, cioè dei complessi di istruzioni costituiti da un blocco che chiamiamo "blocco iterativo", e da elementi organizzativi (parole chiave ed espressioni) con il compito di controllare la sequenza delle esecuzioni delle frasi dal blocco.
Una esecuzione delle frasi di un blocco iterativo la chiamiamoo **ciclo iterativo**, in breve "ciclo", o **manovra ciclica**.
Una esecuzione di tutti i cicli attualmente richiesti da un costrutto iterativo viene detta **iterazione**.

I blocchi iterativi strutturalmente più semplici sono costituiti da una o più frasi esecutive delimitate da parentesi graffe coniugate; queste possono essere trascurate (facotative nel caso di una sola frase.
Si possono anche organizzare blocchi iterativi molto articolati contenenti interamente altri costrutti iterativi e altri costrutti selettivi i quali a loro volta possono essere articolati quanto si vuole.

Le azioni compiute nei diversi cicli presentano differenze tendenzialmente contenute, in quanto devono rendere possibile e conveniente richiederle con un unico blocco di frasi.

Nel linguaggio C++, come in gran oarte dei linguaggi procedurali, si possono organizzare svariati tipi di costrutti iterativi che cercano di soddisfare esigenze diverse e seguono 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 tendenzialmente 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.

I costrutti del tipo (I) vengono retti 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 una modifica, che in genere è un incremento o un decremento dell'indice) da eseguire prima di un nuovo ciclo esecutivo e da una relazione che stabilisce se si avrà ueffettivamente un ciclo successivo o se viceversa l'iterazione è conclusa.

Le iterazioni del tipo (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 un eventuale nuovo ciclo, quelle governate da una clausola esaminata alla fine di ogni manovra e quelle rette da una clausola esaminata in un certa frase del blocco iterativo o anche in pi¸ ossia in qualche fase dell'iterazione.

B70i.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 `for` con un blocco iterativo ridotto ad una semplice frase di accumulo, un indice di reiterazione, `i`, che viene definito, inizializzato ed utilizzato nel nido tra parentesi tonde che segue la parola chiave `for`; questo indice assume i successivi valori 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, in una sorta di ritorno al passato.

```
    for(int mese=12; mese>0; mese--)
        cout << "mese numero " << mese << val[mese] << 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 la distinzione tra anni bisestili e nonbisestilche vanno 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] = 365 ;
        }
```

L'indice di un costrutto `for` può essere modificato come si vuole 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 fraintendimenti.

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 descriverle come visite di una sequenza di posizioni visualizzabili; questa sequenza potrebbe essere un intervallo numerico, una progressione aritmetica, una progressione geometrica, 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**i.05** In generale un costrutto `for` presenta una assegnazione iniziale ad una variabile intera (ma potrebbe anche essere una variabile real) che assume il ruolo di **indice del costrutto**, una clausola da testare prima di effettuare un 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 in tal caso 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 quando si organizza un costrutto `for` che rinuncia a servirsi di un suo 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 può essere modificato a ogni nuovo ciclo, ma che può anche riguardare più parametri nessuno dei quali viene incaricato del ruolo di indice (unico) dell'iterazione.
Una tale clausola può essere molto complessa e/o essere incapsulata in una function [B70e] richiamata nel blocco iterativo; in quest'ultimo caso la clausola può non essere agevolmente rilevabile dal testo

del programma e questa poca visibilità dell'indice comporta un certo rischio di poca controllabilità del costrutto da parte del programmatore.

A loro volta le azioni che vengono eseguite tra una esecuzione di un ciclo e la eventuale successiva possono mancare, oppure possono consistere in un semplice incremento o decremento, oppure essere rette da complessi gruppi di frasi esecutive, in particolare possono venire "incapsulate"' in una function.

B70i.06 Presentiamo altri frammenti con 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[1000] e due soglie dTHMI e dTHMA,
    // porre in dacc[<daccL] la sequenza dei suoi valori compresi tra le soglie
    // e porre nell'array daccpos[<daccL] la sequenza delle rispettive posizioni
    daccL=0;
    for(int i = 1 ; i++ ; i<1000) {
    if(dTHM <= d[i] && d[i] <= dTHMA) {
    dacc[daccL] = d[i]; daccpos[daccL++] = i;
    }
```

B70i.07 (1)
```
// Dato un array int d[<dL] di interi che esprimono valori percentuali,
    // costruire l'array decil[<10] dei numeri di occorrenze dei valori
    // 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;` che si riduce a questa sola parola chiave seguita dal terminatore di frase ";"'. 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 in blocchi iterativi degli altri tipi che vedremo nelle prossime sezioni, blocchi caratterizzati dalle parole chiave `while`, e `do`; inoltre può comparire e nella struttura selettiva caratterizzata dall parola chiave `switch`.

B70i.08 (1)
```
// Lettura di un array monodimensionale di interi vi[] di 31 componenti
    for(int ivi = 0 ; ivi++ ; ivi < 31) {
```

```
        cin >> vi[ivi];
    }
```

(2)
```
// Lettura nell'array code[] di una stringa di caratteri diversi da ';'
    avente al più 40 caratteri e precisazione del loro numero nuc
    int nuc = 0; char carletto;
    for(int nuc = 0 ; nuc++ ; nuc < 40) {
        cin >> carletto ;
        if(carletto == ';') break;
        }
```

(3)
```
Lettura di un elenco di sigle in numero non superiore a 35, ciascuna di al più 10
caratteri alfabetici;
i caratteri delle sigle sono seguite da blank;
le sigle, se minori di 15 sono seguite da sigla blank.
    e 0; deteminazione del loro numero Nsigl
    e loro collocazione nei primi Nsigl intervalli di 10 posizioni ciascuno
    facenti parte dell'array sigl[]
    char sigl[350], lett; int Nsigl, c;
    for(c = 0; c++; c<350) sigl[c] = ' '; // predispone blanks in sigl
    for(Nsigl = 0 ; Nsigl++ ; Nsigl < 35) {// corsa sulle sigle
        forc = Nsigl*10; c++; c < (Nsigl*10+10 {// corsa sui caratteri letti
            cin >> lett ;
            sigl[c] = lett;
            if(lett == ' ') break;
            }
        if(c == Nsigl*10) break;
        }
```

B70i.09 (1)
```
// Scrittura delle stringhe precedentemente lette in righe successive
allineate a sinistra; scrittura delle stringhe precedenti in successive colonne
separate da colonne di un blank da leggersi dall'alto in basso
    char sigl[350], lett; int isigl, Nsigl, c;
    // stampa per righe
    for(Nsigl = 0 ; Nsigl++ ; Nsigl < 35) {// corsa sulle sigle
        if(sigl(Nsigl*10 == ' ') break;
        for(c=0; c++; c<10) {
            cout << sigl[Nsigl*10+c] ;
            endl;
        }
    }
    // stampa per colonne
    for(c=0; c++; c<10) {
        for (isigl = 0 ; isigl = isigl+10 ; isigl < Nsigl) {
            cout << sigl[isigl*10+c] ; cout << ' ';
        }
    }
```

B70**i.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.

(4) Eserc. Scrivere un frammento di programma che genera la tabellina della moltiplicazione tra interi positivi da 1 a 10.

B70**i.11** Il comando `while` si può considerare una semplificazione del comando `for`, in quanto come argomento presenta solo la clausola che serve a decidere se eseguire o meno una nuova manovra ciclica. Il costrutto basato su `while` presenta la forma seguente

azioni di inizializzazione
`while(`*clausola per consentire nuova iterazione*`) {`
    blocco iterativo
    `}`

In alcune esecuzioni il blocco di una iterazione `while` potrebbe non essere affatto eseguito; questo accade sse preliminarmente alla esecuzione della prima manovra ciclica l'espressione condizionale vale false.

Consideriamo un paio di esempi.

```
(1) si abbia una sequenza di valori interi vi[i] per i < Nvi;
    raccogliamo in viOK[] i primi 20 valori superiori a 273 trovati
    in vi[], tenendo conto che potremmo trovarne meno di 20.
    int vi[300], Nvi;
    lettura o costruzione di vi[0 ... ¡ Nvi
    int ivi = 0, viOK[20], NviOK = 0;
    while(NviOK < 20 && ivi < Nvi) {
        if(vi[ivi] > 273) viOK[NviOK++] = vi[ivi];
        ivi++;
        }
    utilizzo di viOK[0 ...  < NviOK]

(2) si abbia una sequenza di valori interi pos[i] per i < Npos;
    raccogliamo in vic1[Nvic1] i primi 10 valori che distano da pos1 meno di 20
    in vic2[Nvic2] i primi 15 valori che distano da pos2 meno di 25;
    Nvic1 e Nvic2 potrebbero essere inferiore a 10; pos1 e pos2 sono molto distanti
    int Npos, ipos=0, pos[200], Nvic1=0, vic1[10], Nvic2=0, vic2[10];
    whileipos < Npos {
        posatt = pos[ipos];
        if(Nvic1 < 10) {
            if(pos1-20 <= posatt &&posatt < pos1+20) {
                vic1[Nvic1++] = posatt; ipos++;
                if(ipos <= Npos) posatt = pos[ipos];
            }
```

```
    if(Nvic2 < 15) {
        if(pos2-25 <= posatt &&posatt < pos2+25) {
            vic2[Nvic2++] = posatt; ipos++;
        }
    if(Nvic1+Nvic2 >= 25) breaak;
}
```

B70i.12 Presentiamo il costrutto `do - while`, costrutto iterativo che si può considerare una semplificazione del costrutto `for`, oppure come una variante del costrutto `while`.
Esso presenta la forma seguente leggermente più articolata della precedente.

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

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 nuova manovra ciclica successiva. Vediamo alcuni esempi.

(1)
```
consideriamo una sequenza di valori interi vi[i] per i < Nvi,
sicuramente inferiori a 2000;
poniamo in vinmadime il primo valore minore di 60 oppure,
in mancanza di meglio, il minimo dei valori in vi[<Nvi]
int vi[300], Nvi;
```
lettura o costruzione di `vi[0 ...  < Nvi]`
```
int vinmadime = 2000; posivi = -1; ivi=0;
do {
    if(vi[ivi] < vinmadime) {
        vinmadime = vi[ivi];
        posivi = ivi;
        }
    }
while(vinmadime >= 60) ;
```
utilizzo di vinmadime

B70i.13 Introduciamo i due comandi `continue` e `break` che possono essere utilizzati all'interno dei blocchi iterativi per contribuire agli effetti di ciascuno dei cicli iterativi.

Il comando `continue` può essere posto dopo un comando selettivo `if`, `else if` o `else` oppure nella posizione conclusiva 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 iterazione successiva, 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
    }
```

B70i.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 B70?18, in relazione a blocchi `case` in costrutti `switch`.
La sua esecuzione comporta l'interruzione dell'esecuzione del blocco iterativo; esso quindi viene attivato quando si verificano le condizioni che richiedono questa interruzione.
Tipicamente tale comando compare in una posizione intermedia dei un blocco iterativo in modo da consentire l'esecuzione di una prima parte di una manovra iterativa che risulterà essere l'ultima, evitando l'esecuzione della sua seconda parte.

L'effetto di conclusione della iterazione di un `break` può accompagnare l'effetto di una clausola di reiterazione, oppure servire per controllare la conclusione di 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
    }
operazioni che si servono di dati modificati nel costrutto while
```

(1) Eserc. Precisare un frammento di programma che inizia con il commento: `// In seq[0..49] si trova una sequenza crescente di interi positivi;`

```
// individuare la posizione del massimo valore inferiore a 100.
int maxvalinf100 = 0;
forint i = 0; i++; i ¡50 {
    if(seq[i] ¿= 100) break;
    if(seq[i] ¿ maxvalinf100) maxvalinf100 = seq[i];
    }
```

B70i.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, che ovviamente deve essere evitata.
Lo schema di un frammento riguardante 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 di un tale blocco iterativo deve essere studiata attentamente per garantire che in tempi ragionevoli si abbia effettivamente l'esecuzione di un comando `break`. In caso contrario si avrebbe una ripetizione illimitata della manovra ciclica e il computer resterebbe illimitatamente silenzioso e inutilizzabile.

In una tale situazione si usa dire che "il computer è in loop" e il verificarsi di questa circostanza è del tutto negativo; l'esecuzione va interrotta e il programma va modificato e questo è notevolmente svantaggioso se risulta difficile capire quando si deve arrestare il procedere dell'esecuzione e quali sono le correzioni da apportare al programma.

B70i.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 più in generale di una 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 assunto della variabile o dalla espressione che segue `switch`.

Nelle soluzioni più semplici 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 switch può iniziare con la semplice parola chiave `default` collocata prima della formulazione delle azioni che sono da eseguire solo quando il valore della variabile di switch è risultato diverso da tutti quelli previsti per i sottoblocchi presenti.

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

(1)
```
// 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, che è evidentemente pleonastico.

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, procede ad eseguire le operazioni previste nel sottoblocco successivo.

Dunque se più sottoblocchi mancano del `break` finale, dopo l'esecuzione delle manovre previste dal sottoblocco si può avere l'esecuzione delle operazioni di vari sottoblocchi successivi, fino a quelle del primo blocco nel quale viene attivato un comando break.

È prevedibile che in talune di queste situazioni i possibili percorsi del controllo sulle varie frasi possono essere un po' complicati da seguire; questa possibilità consente però di risparmiare molte ripetizioni di comandi.

B70i.17 Negli schemi dei costrutti precedentemente introdotti compaiono blocchi di programma che possono esprimere manovre molto articolate.
In un programma che vuole risolvere un problema ben definito attraverso un procedimento ben organizzato è opportuno che si incontrino blocchi ciascuno dei quali possegga una finalità operativa che possa risultare motivata molto chiaramente agli operatori che hanno il compito di controllare la qualità del programma stesso, o per valutarne il valore tecnico o economico, o per stabilire se e come adattarlo a nuove esigenze.
Questi controlli di qualità possono riguardare diverse scale valutative riguardanti i molteplici parametri collegati agli obbiettivi che si devono tenere presenti.
Ai diversi parametri si possono dare diversi pesi e diverse soglie desiderabili.
Questi parametri in genere riguardano qualità importanti e delicate quali adeguatezza, correttezza, velocità, efficienza esecutiva, facilità d'uso, versatilità, riutilizzabilità e altro ancora.

I blocchi che si possono riconoscere in un programma prendendo in esame le occorrenze delle espressioni riservate e delle parentesi `{`, `}`, `(`, `)`, `[` e `]` possono avere articolazioni e lunghezze molto diverse, dalle più ridotte alle alle più estese.

I blocchi più minuti sono costituiti da un solo enunciato da eseguire o da una sola espressione da valutare.
Altri blocchi sono costituiti da sequenze di enunciati e da sottoblocchi, cioè da blocchi interamente contenuti nel blocco dal quale dipendono; di ogni sottoblocco si dice che è interamente annidato nel blocco nel quale si colloca.
Vi sono poi blocchi costituiti da uno dei costrutti selettivi o iterativi visti in precedenza, costrutti nei quali si individuano dei sottoblocchi.

Analizzando in termini di blocchi e sottoblocchi annidati i testi dei programmi ben strutturati ottenuti a partire da frasi esecutive con le composizioni ottenute con sequenziamenti e costrutti selettivi e curando che tutti i blocchi e tutte la composizioni siano delimitate da proprie parentesi graffe (che possono anche risultare risparmiabili) si possono individuare strutture formali che vengono chiamate arborescenze distese e che sono esaminate in D30.
In queste strutture si possono avere blocchi con livelli di annidamento anche molto diversi, corrisponenti ad arborescenze con cammini massimali di lunghezze molto diverse.

Queste strutture raccomandabili tuttavia non sono le più generali; per schematizzarle tutte occorre considerare anche le composizioni con costrutti iterativi, anch'essi delimitati da proprie parentesi graffe che possono non essere indispensabili.

B70i.18 La programmazione strutturata consente di esprimere tutte le procedure pensabili. Qui non cerchiamo di dimostrare questa affermazione (per la quale rinviamo a ??), ma ci limitiamo a segnalare questa "illimitata" potenzialità della programmazione strutturata facendo ricorso all'intuizione.

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, ossia che va ssumendo sequenze di valori, 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 una composizione di sottoproblemi meno compositi.
In effetti la aoatanziale totalità dei problemi da affrontare con procedure si sono rivelati trattabili con la programmazione strutturata.

I precedenti modi per ridurre un problema portano a delineare un modo di sviluppare un tipo di procedimento complessivo costituito da 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 contenenti indicazioni più dettagliate.
Questo tipo di riduzione a sottoproblemi da studiare con maggiori dettagli si può attuare a più livelli.
In tal modo si può proseguire fino a che, o si hanno solo blocchi facilmente esprimibili, o ancra più convenientemente blocchi già studiati e formalizzati in precedenza riutilizzabili con facilità, oppure 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, cosa che può richiedere una riduzione degli obiettivi pretesi, oppure una riformulazione più corretta e accurata del problema stesso e dei fattoriche lo determinano.
Nel primo caso si sta ottenendo un programma ben strutturato in grado di risolvere il problema dato.

`L'esposizione in https://www.mi.imati.cnr.it/alberto/ e https://arm.mi.imati.cnr.it/Matexp/matexp_main.php`