ESERCIZI SUL LINGUAGGIO C++

Mario G. Cimino

Pisa, 2005

Il presente materiale è stato prodotto durante l'attività didattica per l'insegnamento di Informatica

(Corso di Laurea in Ingegneria Biomedica, a.a. 2005-06)

ESERCITAZIONE 1

IL SISTEMA OPERATIVO E L'AMBIENTE DI SVILUPPO DI APPLICAZIONI

1.1 Concetti introduttivi

- Un calcolatore elettronico (**computer**) è una macchina programmabile, ossia in grado di eseguire dei compiti secondo disposizioni fornite attraverso una sequenza di istruzioni (**programma**).
- Un computer, per essere tale deve avere almeno una **memoria di calcolo** (o RAM) ed una unità di elaborazione (**processore** o CPU, es. il *Pentium IV*). Tutto ciò può risiedere in un dispositivo microelettronico di pochi millimetri di spazio.
- Vi sono computer nelle automobili, cellulari, orologi, lavatrici, apparecchiature mediche, sistemi di previsione metereologica, aerei, sistemi di controllo del traffico aereo, centrali nucleari, etc.
- Noi faremo riferimento al tipo di computer più diffuso, il **personal computer**, tipicamente dotato anche di **memorie di archiviazione** a supporto fisso (disco rigido o hard disk) e removibile (floppy, CD, DVD, memorie flash,…) per archiviare dati e programmi, e di **dispositivi di ingresso/uscita** (monitor, tastiera, mouse, stampante, webcam, scanner…) per interagire con gli esseri umani.

- È importante distinguere tra la parte fisica di una macchina (**hardware**, il "corpo") e la logica di funzionamento (**software**, la "mente"). Quest'ultima viene impartita attraverso un programma. Anche se i "corpi" delle macchine sono molto diversi, le "menti" possono operare secondo i medesimi principi. In questo corso parleremo solo di come si programma la logica di funzionamento.
- Es. un programma per gestire gli sms in un cellulare ha la medesima logica di un programma per gestire la posta elettronica su un computer di ufficio: entrambi si basano sulle funzioni "componi", "invia", "ricevi" e "cancella" messaggio.
- Le istruzioni aggiuntive rispetto al telefonino cellulare, che nel personal computer occorrono per gestire dei dispositivi più potenti (es. lo schermo con più ampiezza e profondità di colore, invece del display, consente la creazione di effetti grafici più complessi) si suppongono già fornite da altri programmi di servizio, quali il sistema operativo, i driver, ed altri programmi raggruppati in pacchetti detti **librerie**.

1.2 Il sistema operativo, i programmi, i linguaggi di programmazione

- Il **sistema operativo** è il programma (software) che si occupa della gestione diretta dei dispositivi fisici (hardware) che compongono un calcolatore, del controllo degli accessi degli utenti e degli altri programmi in esecuzione.

- Esempi di sistemi operativi: *Apple Mac-OS, FreeBSD, Linux, Microsoft Windows, Sun Solaris*. Generalmente su un personal computer si possono installare molti sistemi operativi, ma se ne può avviare uno solo ad ogni accensione.
- Come fa un sistema operativo a comunicare con gli innumerevoli dispositivi che si producono e si produrranno, ad esempio tutti i tipi di stampante? In realtà il sistema operativo impartisce lo stesso ordine a tutte le stampanti. Tale ordine viene recepito da un altro programma, detto **driver** (che viene fornito assieme alla stampante), che lo traduce dal linguaggio generico nel linguaggio particolare della stampante.
- Ecco perché, quando si compra una stampante (o qualsiasi altro dispositivo), occorre installare anche il programma driver per farla funzionare. In realtà i sistemi operativi moderni hanno già in partenza i driver più comuni.
- I **programmi** vengono solitamente scritti in un **linguaggio di programmazione di alto livello**, un linguaggio testuale con costrutti linguistici simili al linguaggio naturale degli esseri umani. Esempi di linguaggi di alto livello: *Basic, C, C++, C#, Delphi, Java, Phyton, Prolog, Perl.*
- Un programma scritto in linguaggio C++, detto **codice sorgente**, viene tradotto (o **compilato**) in **linguaggio macchina eseguibile**. La struttura del programma eseguibile è diversa per ogni sistema operativo, per cui passando da un sistema operativo ad un altro occorre ricompilare il codice sorgente, ricollegarlo ai programmi di servizio, e ricreare il

programma eseguibile.

- Ecco perchè i programmi di **installazione** di pacchetti applicativi sono diversi per ogni sistema operativo. Essi contengono un programma eseguibile che provvede a copiare opportunamente tutti i componenti di una applicazione nella memoria di massa, ed ad eseguire opportune configurazioni sul sistema operativo.

- Esempi di programmi (o applicazioni) :
  - *Microsoft Word:* per elaborare testi
  - *Netscape Navigator:* per navigare in Internet
  - *W32.Blaster.Worm:* virus informatico (frammento di programma)
  - *Eliza:* programma-psicologo con cui si può conversare (intelligenza artificiale)
  - *Deep Blu:* programma-scacchista tra i più forti al mondo
  - *Bloodshed DevC++:* programma per scrivere altri programmi, cioè un ambiente di sviluppo di applicazioni in linguaggio C++

1.3 I file, le cartelle, le finestre, il mouse

- Un sistema operativo consente di memorizzare contenuti (dati o programmi) nella memoria di archiviazione tramite il concetto di **archivio** (più comunemente noto come

file). Ogni archivio è come un nastro di carta su cui si può leggere o scrivere un flusso di simboli, ed è identificato da un nome. È possibile generarne uno nuovo, aprirlo, chiuderlo, leggerlo, scriverlo, eliminarlo, spostarlo.

- I file sono organizzabili in **cartelle** (**directory**). Una cartella può contenenere file ed altre cartelle. È possibile creare, eliminare o spostare cartelle. La memoria di massa è suddivisibile in **partizioni**, ed a queste vengono associate delle **unità**, nominate con le lettere dell'alfabeto. Es. unità "A:" per il lettore floppy, "C:" per il disco rigido, "D:" per il lettore CD-ROM.

- Al nome del file tipicamente viene aggiunto il simbolo "." ed una breve sequenza di caratteri (**estensione**) che denota il tipo di applicazione che può leggere quei dati. Esempi

tipo di file	estensione	esempi
pagina web	*.html* *.htm*	miapagina.html
documento di testo senza stile	*.txt*	rubrica.txt
file eseguibile (programma)	*.exe*	videogame.exe setup.exe
documento con stile, in formato leggibile su molti sistemi operativi	*.pdf*	tesi.pdf
immagine	*.gif* *.png* *.jpg* *.bmp*	miafoto.jpg
documento con stile, in formato proprietario, leggibile con	*.doc*	tesi.doc

l'applicazione Microsoft Word		
file sorgente C++	*.cpp*	esercizio.cpp
filmato	*.avi* *.mpg*	film.avi
file progetto (in DevC++)	*.dev*	esercizio.dev
brano musicale	*.mp3*	serenata.mp3
base musicale	*.mid*	valzer.mid

- NOTA: le estensioni sono solo una convenzione. Anche se si cambia l'estensione di un file rinominandolo, il suo contenuto non cambia. Quindi è sempre possibile aprirlo con la medesima applicazione, se si tratta di un file unico con dei contenuti intellegibili (es. un documento, un film, una immagine). Se invece si tratta di file intermedi o processabili da altri programmi (es. file sorgente) allora spesso i programmi sono fatti in modo da accettare solo alcune estensioni.
- La posizione di un file all'interno della memoria di massa viene descritta mediante il percorso (**path**). Ad esempio, nei sistemi operativi Windows:

  `C:\vacanze2005\foto\istanbul.jpg`

  identifica il file sorgente di tipo immagine di nome "istanbul.jpg" contenuto nella cartella "foto", che a sua volta è contenuta nella cartella "vacanze2005", presente nell'unità "C:" associata al disco rigido.

- I moderni sistemi operativi per personal computer sono dotati di un componente software integrativo, detto **interfaccia grafica utente**, che mira a consentire all'utente di interagire col calcolatore manipolando graficamente degli oggetti (rappresentati come **icone**), e svincolandolo dal dover imparare una serie di comandi da impartire con la tastiera come invece avviene con le interfacce testuali.
- L'interfaccia grafica nei sistemi operativi moderni è concepita come la metafora di un piano di lavoro rappresentato dallo sfondo (detto scrivania o **desktop**), gli archivi iconizzati come foglini (con vari ornamenti), le cartelle come cartelline, se chiuse, come **finestre**, se aperte. Le applicazioni sono rappresentate con icone particolari per ogni applicazione.
- Un tipo particolare di file sono i **collegamenti**, ossia dei file icona vuoti che sono collegati ai veri file sparsi nel disco fisso. Sono utili soprattutto sul desktop, per avere "a portata di click" tante applicazioni. Si riconoscono per il simbolo

  in basso a sinistra dell'icona. Eliminando il collegamento non si elimina il file.
- Ciascuna applicazione, una volta avviata si apre come una nuova finestra ed offre una propria interfaccia grafica utente. Ad esempio, un programma per disegno avrà come icone delle penne, squadre, pennelli, colori,…

- In una finestra, le diverse funzionalità possono essere selezionate tramite **menu a scorrimento** che contengono elenchi di voci selezionabili.
- Il principale dispositivo adoperato per interagire con gli elementi di una interfaccia grafica è il **mouse**. Ai movimenti del mouse sul piano di lavoro, corrispondono movimenti simili del **puntatore** sullo schermo. Il puntatore è la metafora del proprio "dito", e puo` interagire con i componenti grafici mediante il **click**, il **doppio click**, o il **trascinamento** (ossia tenere il tasto premuto e spostare il puntatore) mediante il tasto sinistro. Il tasto destro ha il click per aprire un menu di proprietà di un elemento.

1.4 Il sistema operativo Microsoft Windows

- Windows è un sistema operativo **multiutente**. Ciascun utente accede al sistema mediante una operazione detta **login** o **logon**. Dopo l'accensione del computer e l'apparizione di qualche messaggio di servizio, compare una finestra in cui occorre inserire il nome utente (o **username**) e la **password**, quindi cliccare su "OK"
- Nel laboratorio adopereremo l'utente con nome *studenti* e password *studenti*. Ciascun utente ha un proprio desktop e dei propri **diritti** di accesso ai servizi. Ad esempio l'utente **Administrator** è l'utente amministratore, con tutti i diritti. L'utente studenti non ha i diritti per installare nuove applicazioni.
- Il click singolo server per: aprire menu e selezionarne le voci, selezionare icone, per

azionare bottoni, per attivare una finestra. Il doppio clic serve per aprire un file o avviare un'applicazione. Il trascinamento per spostare file e finestre, e per ridimensionarle (agendo sui bordi).

- Se una finestra è troppo piccola per il contenuto, sui bordi compaiono le barre ed i pulsanti di scorrimento, trascinando e cliccando le quali si vede "scorrere" il piano interno della finestra con tutti gli oggetti ad esso posizionati.
- Windows è un sistema operativo **multiprocesso**, ossia consente a diverse applicazioni di essere eseguite in parallelo. La Fig.1 mostra che l'applicazione *Notepad* è aperta assieme all'applicazione *DevC++*. Un **processo** è un programma in esecuzione. Si può anche avviare più volte la medesima applicazione, generando diversi processi.

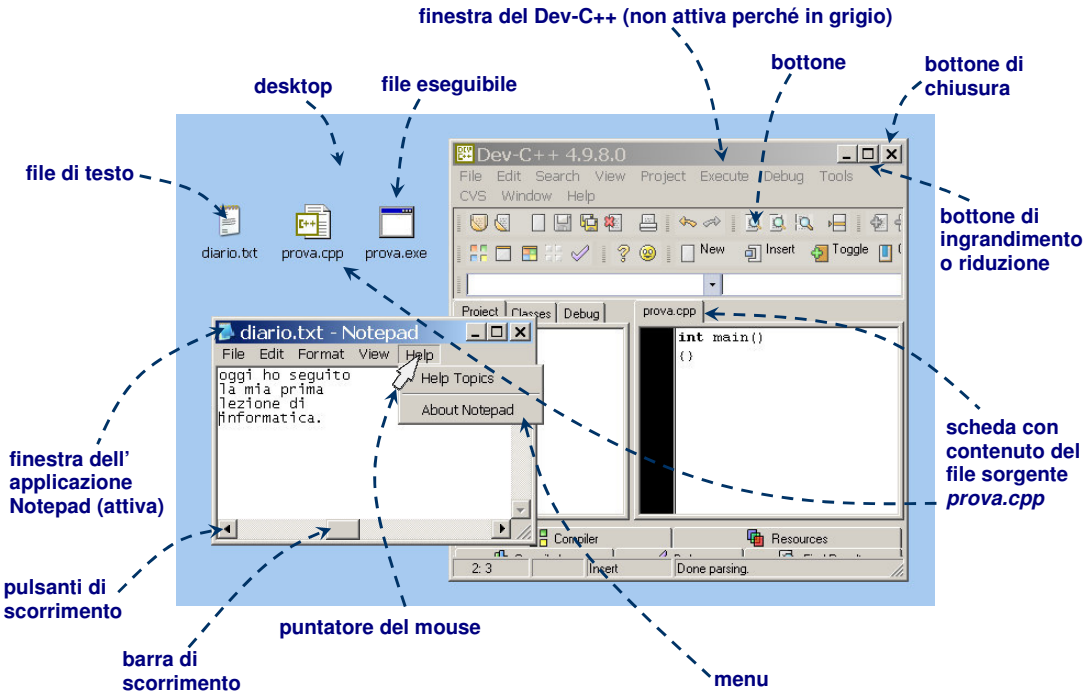


Fig. 1 – Interfaccia grafica nel sistema operativo Windows

1.5 Manipolare i contenuti del disco e file di testo

- Tasto destro sul pulsante *START* → Esplora
- Appare una nuova finestra, divisa in due sotto-finestre, che consente di esplorare qualsiasi cartella come se fosse il desktop, sia dal disco fisso che dalle unità rimovibili.
- Cliccando sulle icone della parte sinistra, si sfogliano le unità e le cartelle in esse contenute. Nella parte destra compaiono i contenuti della cartella selezionata (Fig.2) .
- Per creare una nuova cartella, cliccare nella parte di destra (dove non appare alcun elemento) con il tasto destro del mouse, e selezionare *nuovo → cartella.* Quindi digitarne il nome.
- Per creare un file di testo si può usare il programma *Notepad (Blocco note)* dal menu *START → programmi → accessori → Blocco note* (il simbolo "→" indica un sotto-menu o un sotto-elemento)
- Cliccando nella parte bianca si può scrivere qualcosa con la tastiera (Fig.1).

RICORDA: Tutte le modifiche eseguite risiedono nella memoria di calcolo, che è una **memoria volatile**, quindi allo spegnimento del PC saranno perse. Occorre salvarle su

memoria di archiviazione, che permane anche in assenza di energia (**memoria permanente**)

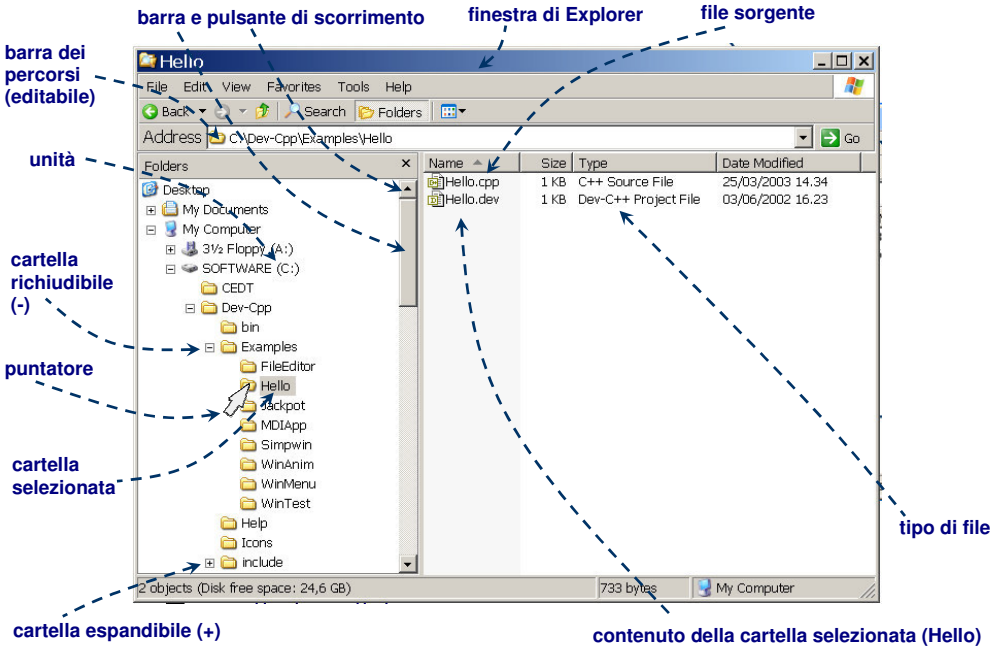


Fig. 2 – Explorer

- Per **salvare il file** su memoria di massa selezionare dal menu *File* → *Salva*. Questo sovrascrive automaticamente la vecchia versione del file. Per salvarne una copia con un altro nome selezionare menu *File* → *Salva con nome...*. Per aprire un file esistente selezionare menu *File* → *Apri*. Infine, per editare un file nuovo slezionare menu *File* → *Nuovo*.
- In tutti i casi, compare una finestra del tipo Explorer, con una casella *Salva in* in cui si può selezionare la cartella di destinazione, ed una casella *Nome file* in cui si può editare il nome.
- Per **aprire un file** di estensione nota, basta anche un doppio click. Verrà automaticamente aperta l'applicazione. Oppure, dopo aver avviato l'applicazione, c'è sempre un menu "open" che consente di aprire il file.
- Per **rinominare un file** *selezionalo con il tasto destro* → *rinomina*, quindi edita il nome (mantenendone l'estensione) e premi il tasto *Enter* (*Invio*) (che in un editor consente di andare accapo).
- Per **cancellare un file** *selezionalo con il tasto destro* → *elimina*, e rispondi "OK" alla richiesta di conferma. Si può anche recuperare il file cancellato sfogliando la cartella **cestino** presente nel desktop, a meno che non si faccia *selezione con il tasto destro sul cestino* → *svuota cestino.*

- Per fare una **copia di un file** *selezionalo con il tasto destro → copia*, poi collocarsi dove si vuole inserire la copia e *tasto destro → incolla*. Per **spostare il file** si può eseguire la medesima procedura in cui al posto di *copia* si seleziona *taglia* (cut).
- La tecnica del copia-incolla (oppure del taglia-incolla) si può eseguire anche per copiare o spostare frammenti di testo in un editor come *Blocco Note*.

RICORDA: non compiere operazioni su file che non conosci, potresti compromettere altre applicazioni o il sistema operativo.

1.6 Installare il DevC++ su un proprio PC

- Innanzitutto occorre il programma di installazione, da richiedere all'esercitatore oppure scaricabile dal sito *http://csjava.occ.cccd.edu/~gilberts/devcpp5*
- **Doppio click sul file eseguibile devcpp.exe**. Appare una finestra che raccomanda di non installare due diverse versioni del DevC++ in un computer. **Cliccare su *OK***.
- Appare una finesra che informa sui diritti d'uso del prodotto, compatibili con il corso, quindi **cliccare sul bottone *I Agreee*** (Sono d'accordo).
- Appare una finestra con delle opzioni. Lasciare tutto così com'è e **cliccare su *Next***.
- Appare una finestra con il percorso in cui verranno copiati i vari componenti del

programma. DevC++ è una grossa applicazione, quindi è composta da diversi file eseguibili, librerie, dati, esempi,... Lasciare il percorso indicato e **cliccare su *Install*** ed attendere qualche decina di secondi che tutto venga copiato.

- Appare una finestra che chiede se il DevC++ dovrà essere accessibile a tutti gli utenti del computer. **Cliccare sul bottone SI**.
- Al termine della procerura, evidenziato dalla comparsa della frase **completed** nella finestra informativa, **cliccare sul bottone Close**.
- Appare una finestra con informazioni irrilevanti ai fini del corso. **Cliccare su OK**.
- Appare una finestra in cui è possibile segliere la lingua (*Select your language*). Scorrendo in basso (mediante la barra o il pulsante di scorrimento) nell'elenco di lingue in ordine alfabetico compare ***Italian***. **Cliccare** tale voce con il tasto sinistro, facendola diventare blu (voce selezionata). Quindi **cliccare sul tasto OK** in basso.
- Si apre la finestra del DevC++, con un'altra finestra di suggerimenti sovrastante che si può chiudere cliccando sul tasto **Chiudi**. Se non si desidera rivederla ad ogni apertura di DevC++, basta cliccare nello spazio bianco accanto a "Non mostrare i suggerimenti all'avvio", selezionando tale opzione, quindi cliccare su Chiudi.

1.7 Primo programma con DevC++

- Per avviare il programma, dal menu START→ *programs*→ *Blooshed DevC++* → *DevC++*.
- Per creare un file sorgente, dal menu *File* → *nuovo* → *file sorgente*. Quindi cliccare nella finestra di destra ed inserire il seguente programma:

```
#include <iostream>

using namespace std;

int main()
{    cout << "ciao ciao\n";
     system("pause");
}
```

- **RICORDA**: il codice sorgente deve essere ben ordinato ed allineato come in figura, mettendo sempre uno spazio tra un elemento ed un altro. Per allineare diverse istruzioni usa il tasto **TAB**, che posiziona automaticamente il cursore (punto di scrittura) su posizioni (colonne) fisse, come se si aggiungessero tanti spazi.

- Dopo aver editato il file, per **salvarlo** cliccare sul pulsante di salvataggio (Fig.3). Appare una finestra in cui in alto (casella *Salva in:*) è possibile indicare dove salvare il file, ed in basso (casella *Nome file:*) è possibile editare il nome del file, ad esempio *prova.cpp*, e quindi salvarlo cliccando sul pulsante *Salva*.
- La seconda operazione è quella di **compilare** ossia verificare che non vi siano errori di sintassi ed in tal caso produrre il file eseguibile ***prova.exe***. Ciò si esegue cliccando sul pulsante di compilazione (Fig.3). Dopo qualche secondo appare una finestra in cui, se tutto va bene, si evidenzia un campo *Status:* ***Done*** ed un bottone **Chiudi** che occorre cliccare.
- A questo punto è stato prodotto un file eseguibile (nella stessa cartella del file sorgente) che è possibile avviare con un doppio click, oppure con il pulsante di esecuzione del devC++ (Fig.3)
- L'esecuzione produce una finestra nera in cui viene stampato "ciao ciao" e poi il messaggio "Premere un tasto per continuare…".
- In realtà il pulsante di compilazione prevede anche il salvataggio del file, se questo è stato modificato. Ed esiste anche il pulsante di compilazione-esecuzione (Fig.3).
- Supponiamo ora di fare un errore di sintassi, cambiando la parola **`system`** in **`systemm.`**

Fig. 3 – Funzioni base del DevC++

- Cliccando sul pulsante di compilazione, la riga errata viene evidenziata in rosso. In basso appare un dettaglio degli errori (Fig.4). Cliccando sul dettaglio, nella finestra in alto si passa automaticamente alla riga corrispondente.

Fig. 4 – Manifestazione di un errore nel DevC++

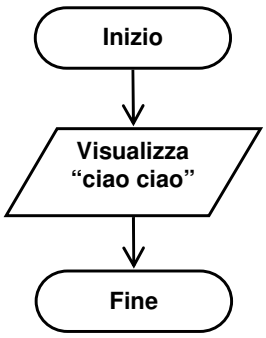
RICORDA: gli errori si correggono nell'ordine di scrittura, poiché spesso un errore in fondo al testo è conseguenza degli errori iniziali. Dopo aver cliccato sul punto dell'errore ed effettuato la correzione, ricompila per vedere se gli errori successivi sono scomparsi.

ESERCITAZIONE 2

LINGUAGGIO C++ : TIPI PREDEFINITI E FLUSSI DI INGRESSO-USCITA

1. Visualizzazione di un messaggio

- Scrivere un programma che visualizza un messaggio.
- Diagramma di flusso:



- Le parti contenute entro i caratteri “//” e “↵” (invio) oppure “/*” e “*/” sono commenti, ignorati dal compilatore.

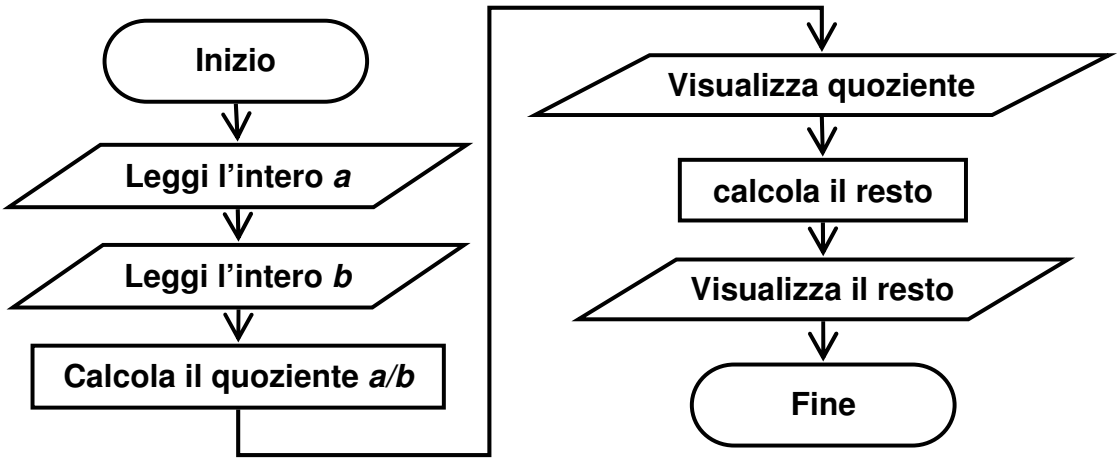
```
// ciao.cpp                                  //(1)

#include <iostream>                          //(2)

using namespace std;                         //(3)

int main()                                   //(4)
{    cout << "ciao ciao\n";                  //(5)
     system("pause");                        //(6)
}

/* Note
(1) Nome del file
(2) includi la libreria dei flussi di
    ingresso ed uscita
(3) usa lo spazio dei nomi dei servizi
    standarddi I/O
(4) definisce la funzione principale
(5) invia la stringa "ciao ciao\n" sul
    flusso in uscita, associato al video
(6) invoca il comando di sistema "pause",
    che chiede all'utente di premere un
    tasto.
*/
```


- La libreria `iostream` definisce un flusso di uscita (`cout`) per i dati. Per visualizzare un valore a video, basta inviarlo al flusso di uscita con l’operazione <<.
- Commentando solo la riga (2), il compilatore non riconosce il simbolo `cout`, perché manca la libreria. Anche commentando solo la riga (3) appare lo stesso errore; ma in tal caso specificando `std::cout` tutto riprende a funzionare.
- Infatti i nomi adoperati nella libreria sono incapsulati in uno **spazio dei nomi** (che è come una cartella dei nomi). Per usare i nomi occorre specificare lo spazio. In tal modo, se una seconda libreria `mialib` con spazio dei nomi `miospace` adoperasse lo stesso nome `cout`, potrei riferirlo come `miospace::cout` senza creare ambiguità.
- Poiché nell’esercizo adoperiamo un’unica libreria, basta mettere la dichiarazione (3) iniziale per dire al compilatore di cercare i nomi nello spazio `std`.
- Commentando la riga (4) il compilatore segnala diversi errori. I programmi devono necessariamente iniziare con una funzione principale `main`.
- Commentando la riga (5) il programma viene compilato ed eseguito, ma non viene stampato nulla. Infatti il compilatore non sa che vogliamo stampare qualcosa, per cui non è in grado di segnalare questa mancanza. Come pure non sarebbe in grado di segnalare errori nel messaggio di stampa. Es. provare a togliere il frammento “`\n`”.

- Commentando la riga (6) il programma viene compilato, ma la finestra scompare troppo velocemente per essere letta. Infatti, l'istruzione serve a lanciare il comando di sistema operativo `pause` che chiede all'utente di premere un tasto.

2. Operazioni tra interi

- Scrivere un programma che legge due interi da tastiera, ne stampa quoziente e resto.
- Diagramma di Flusso:



- Nei diagrammi di flusso, l'ovale indica un punto terminale, il parallelogramma un'operazione di ingresso o uscita dati, e il rettangolo una elaborazione.


```
// interi.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserire due interi da dividere: ";
    int a, b;                                          //(1)
    cin >> a >> b;                                     //(2)
    cout << a << " diviso " << b << " fa " << a/b      //(3)
         << " con resto di " << a%b << endl;           //(4)

    system("pause");
}

/* Note
(1)   dichiara due variabili intere di nome 'a' e 'b'
(2)   legge due valori dal flusso di ingresso (tastiera)
(3-4) x%y significa resto della divisione di x per y
*/
```

- La libreria `iostream` definisce anche un flusso di ingresso (`cin`) per i dati. Per leggere i valori inseriti da tastiera basta prelevarli dal flusso di ingresso con >>.
- Commentando la riga (1) il compilatore non riconosce i nomi 'a' e 'b' adoperati nelle altre righe.
- Le variabili intere definite in (1) non sono inizializzate con alcun valore. Provare a

visualizzare il valore di `a` inserendo l'istruzione `cout << a;` tra la riga (1) e (2).

- Tutto ciò che viene inviato al flusso di uscita è visualizzato al video. I messaggi tra virgolette sono visualizzati così come sono. Per le variabili, viene stampato il valore. Per le operazioni tra variabili, viene calcolato il risultato e quindi stampato.
- La variabile costante `endl` (abbreviazione di "*end line*") è equivalente a "`\n`".
- Il numero 2147483647 è divisibile per 11? Avviare l'applicazione ed inserire i valori `2147483647` e `11`. Risposta: no, perché Il resto è 1. Provare anche con valori negativi quali `-2147483647` e `11`.
- Il numero 2147483648 è divisibile per 2? Certo. Ma provando ad inserire tali valori, l'applicazione visualizza dei risultati senza senso! Il nostro programma ha un **baco** (difetto di funzionamento).
- Infatti una variabile `int` può contenere solo valori tra – 2 miliardi e + 2 miliardi circa (precisamente -2147483647 e 2147483647), perché una `int` è lunga **32 bit**.
- Se vogliamo risparmiare spazio, possiamo decidere di non usare i valori negativi, così recuperando metà intervallo. Modificare la riga (1) aggiungendo **`unsigned`** prima di `int`. Questa volta la divisione di 2147483648 per 2 funzionerà. Ma comunque, siamo limitati ai valori tra 0 e 4 miliardi circa (4294967295).

- Il numero 4294967295 è divisibile per 3 ?

ATTENZIONE: Modificando la riga (1) con **`unsigned long int`** non e detto che si abbiano interi con maggior capacità, se il processore non e in grado di fare calcoli con maggiori dimensioni.

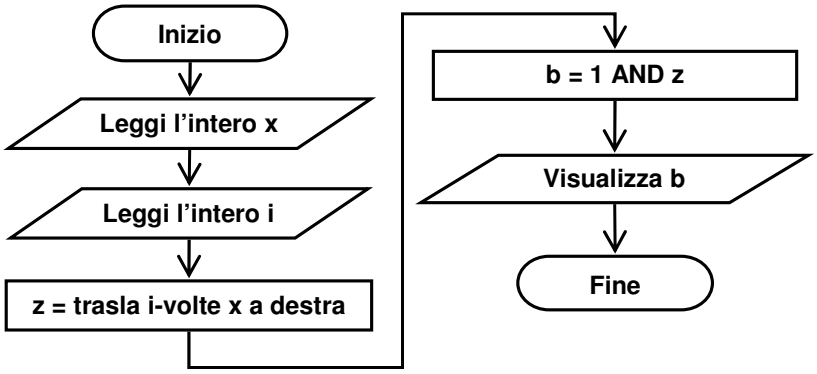
NOTA: Il fatto che i numeri nei processori abbiano una dimensione limitata, non significa che non si possano fare operazioni matematiche con numeri ancora più grandi. Con un po' di tecniche matematiche è facile scrivere programmi software in grado di fare operazioni con un numero di cifre praticamente illimitato. Es. vi sono programmi in grado di produrre un valore di π con qualsiasi numero di cifre, basta munirsi di molta carta per stampante e molta pazienza...

- É possibile anche accorciare le dimensioni di un intero, risparmiando memoria. Es. per memorizzare l'anno di nascita dei circa 6 miliardi di esseri umani che popolano la terra, basta un **`unsigned short int`** per ognuno (16 bit, tra 0 e 65535).

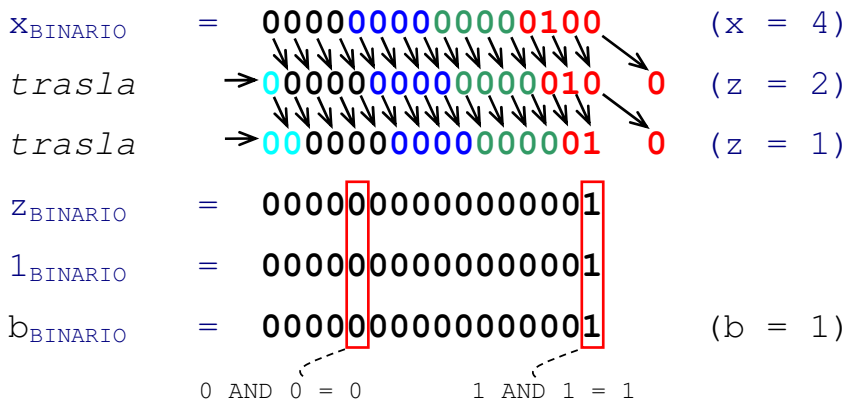
3. Operazioni bit a bit

- Scrivere un programma che legge un intero x ed un intero i da tastiera e stampa il valore dell'i-esimo bit della rappresentazione binaria di x.

Diagramma di Flusso:



- Esempio x = 4, i = 2.
- In base 2 abbiamo solo le cifre 0 ed 1, quindi i numeri 0,1,10,11,**100**,101,... equivalgono risp. a 0,1,2,3,**4**,5,...
- Ad ogni traslazione, viene aggiunto uno zero a sinistra.



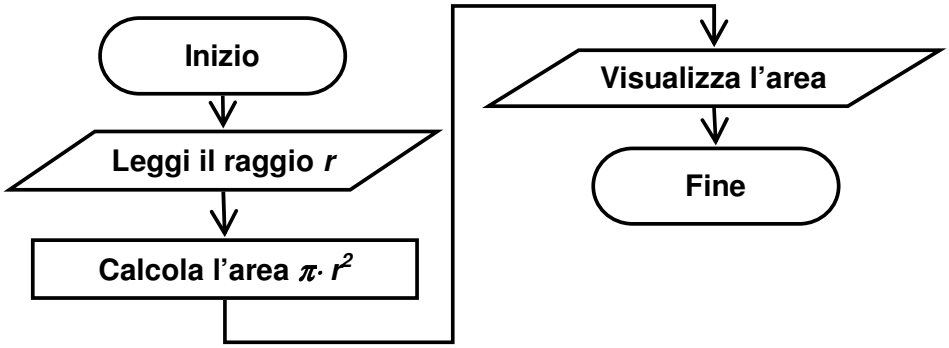
Soluzione proposta:

```
// bit.cpp

#include <iostream>

using namespace std;

int main()
{    unsigned short x, i, b;
     cout << "Inserisci un numero intero: ";
     cin >> x;
     cout << "Quale bit vuoi leggere ? ";
     cin >> i;

     b = 1 & (x >> i);                                  //(1)
     cout << "Il bit n." << i <<  " del numero "
          << x << " vale " << b << endl;

     system("pause");
}

/* Note
(1)  Trasla a destra i volte il numero x, e fai l'AND con
     la maschera 0..01 per selezionare il primo bit
*/
```

- Cosa succede se i = 20 ?.

4. Operazioni tra reali

- Scrivere un programma che legge il raggio di un cerchio e ne stampa l'area.
- Diagramma di Flusso:



- I numeri reali forniti da un processore hanno un limite minimo e massimo, una precisione limitata e non formano un insieme continuo. Come tale rappresentano un sottoinsieme dei numeri razionali.
- Spesso è comodo esprimere i reali in **notazione scientifica**. Es. 1235 = 1.235 x 10^3 = 1.235E+3. In altri termini, "E" significa "per 10 elevato a".
- I limiti dei reali dipendono dal processore. Tipicamente vi sono reali float (32 bit) e double (64 bit). Esempi per il più piccolo valore (in modulo) ed il più grande (in

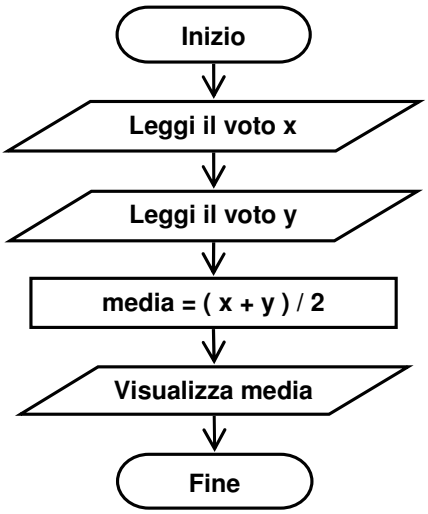
modulo) sono: 1E-38 ed 1E+38 per i float, 1E-308 ed 1E+308 per i double.

```
// reali.cpp

#include <iostream>

using namespace std;

int main( )
{     cout <<  "Inserire il raggio del cerchio: ";
      double raggio;
      cin >> raggio;

      double pigreca = 3.14159;                          //(1)
      double area = raggio * raggio * pigreca;

      cout  << "L'area del cerchio e': " << area << endl;
      system("pause");
}

/* Note
(1) double è il reale a 64 bit, float a 32 bit.
*/
```

ATTENZIONE: il numero di cifre significative spesso non dipende dal fatto che il tipo sia double o float. Modificando il valore di π nella riga (1) a 3.1415926535897, il risultato non cambia!.

- Prova ad inserire i seguenti valori per il raggio: 1E100, 1E200, 1E400.

- Scrivere un programma che calcola il valor medio tra due voti e lo visualizza a video.
- Diagramma di flusso e soluzione:




```
// media.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserire due voti: ";
    unsigned short v1, v2;
    cin >> v1 >> v2;

    double media = (v1 + v2)/2.0;         //(1)
    cout << " media " << media << endl;

    system("pause");
}

/* Note
(1)   prima si esegue il calcolo e poi si assegna
      il risultato alla variabile media.
*/
```

ATTENZIONE: Nella istruzione (1) viene innanzitutto calcolata la somma di due

interi; tale somma viene convertita in tipo double (conversione implicita) poichè va divisa con il double 2.0. Quindi il risultato, di tipo double, è assegnato a media. Cosa succede se si modifica 2.0 con 2? La somma non viene più convertita in double. Quindi si ha una divisione tra interi, che darà come come risultato un valore intero, convertito in un double ed assegnato a media. In pratica, si ha una media sempre senza virgola!

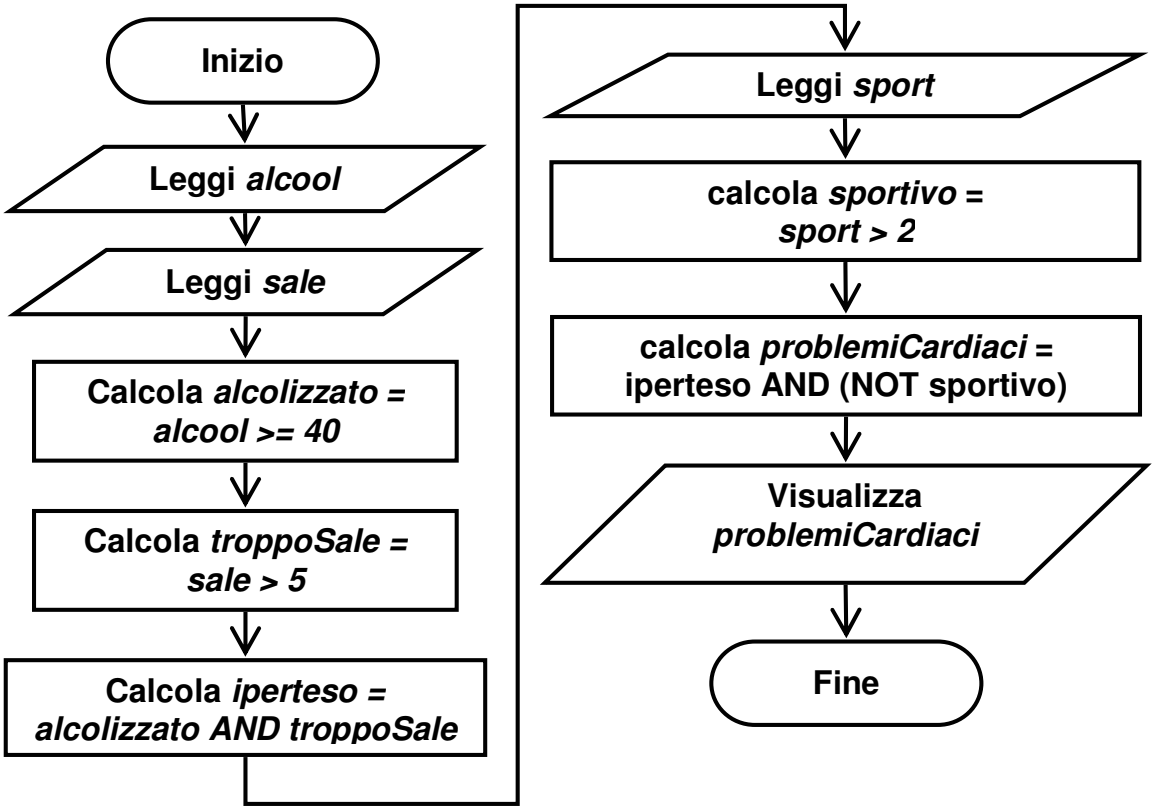
- Si può eseguire una conversione esplicita di tipo scrivendo `(double)(v1+v2)`. In tal modo, il risultato sarà indipendente dalla forma della costante 2.

ESERCITAZIONE 3

LINGUAGGIO C++ : ALTRI TIPI PREDEFINITI E TIPI ENUMERAZIONE

1. Operazioni tra variabili logiche

- Scrivere un programma che stabilisce la possibilità per un paziente di avere problemi cardiaci. Per semplicità, tale possibilità può essere "1" (possibile) oppure "0" (impossibile), sulla base dei seguenti criteri.
- É possibile che un paziente abbia **problemi cardiaci** se e solo se è **iperteso** e **non è sportivo**. Un paziente è **sportivo** se fa sport più di 2 volte a settimana, altrimenti non lo è. Un paziente è iperteso se e solo se è **alcolizzato** (cioè se beve almeno 40 bicchieri di alcolici a settimana) ed assume **troppo sale** (cioè dichiara di assumere un livello di sale maggiore di 5 su 10).
- Poichè molte informazioni del problema sono a due valori, ossia del tipo VERO/FALSO, adoperiamo le variabili logiche per memorizzare risultati intermedi.
- **ATTENZIONE**: Non confondere gli operatori bit a bit e gli operatori booleani. Questi ultimi agiscono sull'intera variabile e non sui bit della sua rappresentazione binaria. Nota che i due tipi di operatore hanno simbologie diverse.
- Diagramma di flusso:



- Soluzione:

```
// medico.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Quanti bicchieri di alcolici assumi per settimana ? ";
    int alcool;
    cin >> alcool;

    cout << "Quale il tuo livello di consumo di sale (tra 0 e 10)? ";
    int sale;
    cin >> sale;

    bool alcolizzato = (alcool >= 40);                          //(1)
    bool troppoSale  = (sale > 5);
    bool iperteso    = (alcolizzato && troppoSale);             //(2)

    cout << "Quante volte a settimana fai sport ? ";
    int sport;
    cin >> sport;

    bool sportivo    = (sport > 2);

    bool problemiCardiaci = (iperteso && !sportivo);            //(3)

    cout << "Hai possibilita` " << problemiCardiaci
         << " di avere problemi cardiaci" << endl;

    system("pause");
}
```



```
/* Note
(1) dato il valore dell'intero alcool, se questo e' maggiore o uguale
    a 40 allora la variabile alcolizzato assume il valore 1 (vero),
    altrimenti 0 (falso).
(2) la variabile logica (o booleana) iperteso assume valore 1 (vero)
    solo se sia "alcolizzato" che "troppoSale" hanno valore 1 (vero)
(3) la variabile logica problemiCardiaci vale 1 solo se iperteso=1 e
    sportivo=0, ossia !sportivo=1. "!" significa "non sportivo".
*/
```

- Quale possibilità di problemi cardiaci ha un paziente che...

  ...assume 80 bicchieri di alcool per settimana, moltisimo sale e fa molto sport? 0

  ...è astemio, non usa sale e non fa sport? 0

  giustificare le risposte calcolando il valore di tutte le variabili logiche intermedie.

2. Operazioni tra caratteri

- Scrivere un programma che, dato un carattere minuscolo, lo converte in maiuscolo ed emette un beep finale.
- La codifica di caratteri Americana (ASCII) memorizza i caratteri alfanumerici su 7 bit. La variabile **char** in C++ è di 8 bit (il bit più a sinistra vale sempre 0).
- Ciascuna configurazione di bit può essere vista come un numero intero in base 2, e quindi convertita in un numero intero in base 10, per cui si ha la seguente tabella

CODICI **ASCII** (binario-decimale-carattere)

0000000	0	NUL
0000001	1	SOH
0000010	2	STX
0000011	3	ETX
0000100	4	EOT
0000101	5	ENQ
0000110	6	ACK
0000111	**7**	**BEL**
0001000	8	BS
0001001	9	HT
0001010	**10**	**LF**
0001011	11	VT
0001100	**12**	**FF**
0001101	**13**	**CR**
0001110	14	SO
0001111	15	SI
0010000	16	DLE
0010001	17	DC1
0010010	18	DC2
0010011	19	DC3
0010100	20	DC4
0010101	21	NAK
0010110	22	SYN
0010111	23	ETB
0011000	24	CAN
0011001	25	EM

0011010	26	SUB
0011011	27	ESC
0011100	28	FS
0011101	29	GS
0011110	30	RS
0011111	31	US
0100000	**32**	**SPC**
0100001	33	!
0100010	34	"
0100011	35	#
0100100	36	$
0100101	37	%
0100110	38	&
0100111	39	'
0101000	40	(
0101001	41	)
0101010	42	*
0101011	43	+
0101100	44	,
0101101	45	-
0101110	46	.
0101111	47	/
0110000	**48**	**0**
0110001	**49**	**1**
0110010	**50**	**2**
0110011	**51**	**3**

0110100	**52**	**4**
0110101	**53**	**5**
0110110	**54**	**6**
0110111	**55**	**7**
0111000	**56**	**8**
0111001	**57**	**9**
0111010	58	:
0111011	59	;
0111100	60	<
0111101	61	=
0111110	62	>
0111111	63	?
1000000	64	@
1000001	**65**	**A**
1000010	**66**	**B**
1000011	**67**	**C**
1000100	**68**	**D**
1000101	**69**	**E**
1000110	**70**	**F**
1000111	**71**	**G**
1001000	**72**	**H**
1001001	**73**	**I**
1001010	**74**	**J**
1001011	**75**	**K**
1001100	**76**	**L**
1001101	**77**	**M**

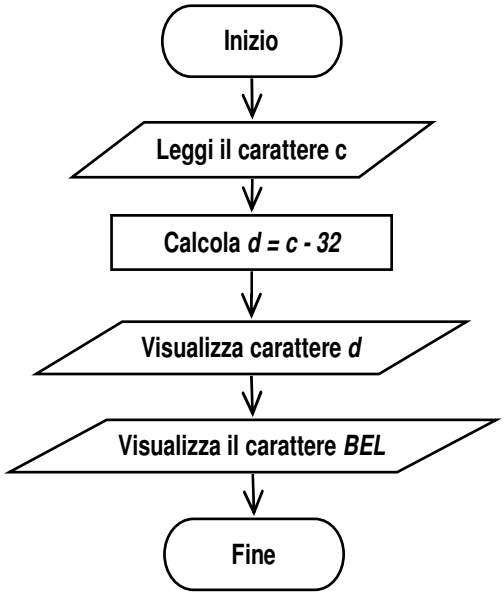
1001110	**78**	**N**
1001111	**79**	**O**
1010000	**80**	**P**
1010001	**81**	**Q**
1010010	**82**	**R**
1010011	**83**	**S**
1010100	**84**	**T**
1010101	**85**	**U**
1010110	**86**	**V**
1010111	**87**	**W**
1011000	**88**	**X**
1011001	**89**	**Y**
1011010	**90**	**Z**
1011011	91	[
1011100	92	\
1011101	93	]
1011110	94	^
1011111	95	_
1100000	96	`
1100001	**97**	**a**
1100010	**98**	**b**
1100011	**99**	**c**
1100100	**100**	**d**
1100101	**101**	**e**
1100110	**102**	**f**
1100111	**103**	**g**

1101000	**104**	**h**
1101001	**105**	**i**
1101010	**106**	**j**
1101011	**107**	**k**
1101100	**108**	**l**
1101101	**109**	**m**
1101110	**110**	**n**
1101111	**111**	**o**
1110000	**112**	**p**
1110001	**113**	**q**
1110010	**114**	**r**
1110011	**115**	**s**
1110100	**116**	**t**
1110101	**117**	**u**
1110110	**118**	**v**
1110111	**119**	**w**
1111000	**120**	**x**
1111001	**121**	**y**
1111010	**122**	**z**
1111011	123	{
1111100	124	|
1111101	125	}
1111110	126	~
1111111	127	DEL

- Alcuni carateri speciali: il carattere BEL produce un beep quando visualizzato; LF è

il LINE FEED e manda il cursore nella riga in basso (↓), mentre CR è il CARRIAGE RETURN e manda il cursore a sinistra (←). Il carattere "\n" o endl(↵) può considerarsi come LF+CR.

- Nel codice ASCII vi sono molti caratteri ormai obsoleti. Ad esempio, FF (FORM FEED) serviva alle stampanti ad aghi a far scorrere il rullo per posizionarsi su una nuova pagina.
- I moderni linguaggi (come Java) adoperano un codice a 16 bit, UNICODE, che contiene un numero maggiore di caratteri e consente l'uso di alfabeti di tutte le maggiori lingue del mondo, invece che solo quello americano.
- **NOTA**: il carattere '**A**' corrisponde al numero **65**, ed il carattere '**a**' al numero **97**. Poichè 97-65=32, si ha che 'A'='a'-32, quindi 'B'='b'-32,..., 'Z'='z'-32. Pertanto, per convertire un carattere da minuscolo in maiuscolo basta sottrarre 32 alla codifica decimale.
- Se nella tua tastiera mancano i caratteri '{' e '}' puoi batterli nel seguente modo. Attiva il tasto BLOC NUM e poi, tenendo premuto ALT digita la sequenza 123 o 125 rispettivamente.
- Diagramma di Flusso e soluzione:

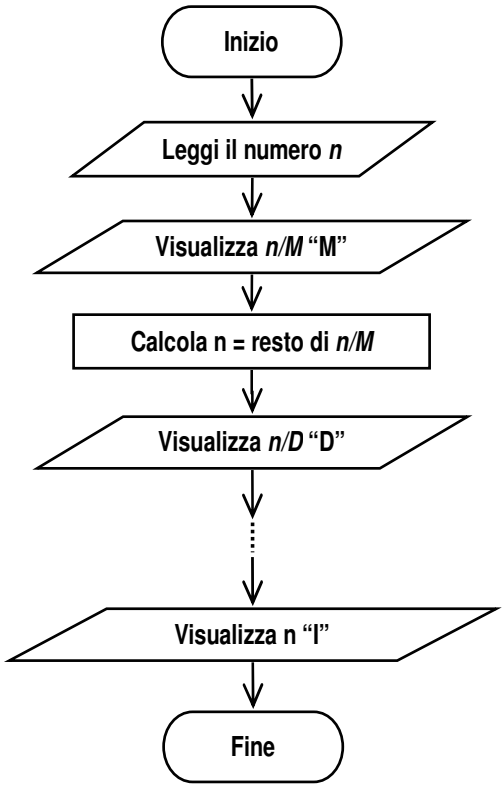



```
// caratteri.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserisci un carattere minuscolo: ";
    char c;
    cin >> c;

    char d = c - 32;                          //(1)
    cout << "L'equivalente maiuscolo e' "
         << d << endl;

    const char BEL = 7;
    cout << BEL;
    system("pause");
}
/* Note
  (1) conversione da minuscolo a maiuscolo
*/
```

3. Operazioni con tipi enumerazione

- Scrivere un programma che converte un numero intero decimale in un numero pseudo-romano, che consiste in un formato semplificato rispetto a quello dei numeri romani. In particolare, tale formato si compone della sequenza di migliaia, cinquecentinaia, centinaia, cinquantine, decine, cinquine ed unità presenti nel numero. Es. *2678* diventa *2M 1D 1C 1L 2X 1V 3I* (cioè *MMDCLXXVIII*)

- Diagramma di flusso e soluzione:



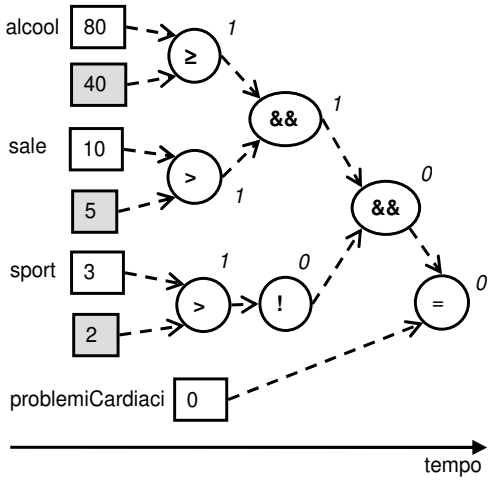

```
// enumerazione.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserisci un numero naturale: ";
    int n;
    cin >> n;

    enum Romano{I=1, V=5, X=10, L=50, C=100, D=500, M=1000};

    cout << "Conversione in pseudo-romano: ";
    cout << n/M << "M ";
    n = n%M;
    cout << n/D << "D ";
    n = n%D;
    cout << n/C << "C ";
    n = n%C;
    cout << n/L << "L ";
    n = n%L;
    cout << n/X << "X ";
    n = n%X;
    cout << n/V << "V ";
    n = n%V;
    cout << n   << "I";
    cout << endl;

    system("pause");
}

/* Note
   Esempio: 2678 diventa 2M 1D 1C 1L 2X 1V 3I
*/
```

4. Espressioni aritmetiche e logiche, operatore condizionale.

- Scrivere un programma che compie la medesima operazione dell'esercizio 1, calcolando in un'unica espressione la variabile logica *problemiCardiaci*. Inoltre, il programma stampa il messaggio *"è possibile che tu abbia problemi cardiaci"* nel caso in cui tale variabile logica valga 1, ovvero "*è impossibile che tu abbia problemi cardiaci"* nel caso in cui valga 0.

- **NOTA**: le espressioni composte sono difficili da comprendere. Cerca di evitarle, o comunque di adoperare le parentesi per evitare ambiguità nella tua interpretazione.

- La sequenza di passi per il calcolo della variabile *problemiCardiaci* è la seguente. Per prima cosa vengono calcolate le espressioni tra le parentesi più interne (quindi "alcool>=40", "sale>5" e "sport>2"). Poi viene applicato l'operatore unario "!" al risultato della terza espressione e l'operatore binario && sulle prime due. Sui risultati di queste ultime due operazioni, viene applicato l'operatore && ed il risultato viene assegnato a *problemiCardiaci*.




```
// espressioni.cpp

#include <iostream>

using namespace std;

int main()
{    cout << "Quanti bicchieri di alcolici assumi per settimana ? ";
     int alcool;
     cin >> alcool;

     cout << "Quale il tuo livello di consumo di sale (tra 0 e 10)? ";
     int sale;
     cin >> sale;

     cout << "Quante volte a settimana fai sport ? ";
     int sport;
     cin >> sport;

     bool problemiCardiaci = (((alcool >= 40) && (sale > 5)) && !(sport > 2));
     cout << "e' " << (problemiCardiaci ? "possibile" : "impossibile")   //(1)
          << " che tu abbia problemi cardiaci" << endl;

     system("pause");
}

/* Note
(1)  stampa il frammento "impossibile" solo se problemiCardiaci è zero.
*/
```

ESERCITAZIONE 4

LINGUAGGIO C++ : USO AVANZATO DEL DEVC++, IL DEBUGGING

1. Introduzione

- Scrivere programmi è un'attività intellettuale, come tale soggetta ad errori. La differenza tra un programmatore esperto ed un principiante non è tanto nel numero di errori introdotti, ma nella maggior capacità dell'esperto di trovarli e correggerli.
- Gli errori possono essere molto difficili da trovare. Se il programma non viene adoperato frequentemente e da tante persone, gli errori più insidiosi non verranno mai trovati, in quanto manifestabili sono in particolari circostanze.
- La principale regola di prevenzione da errori è quella di scrivere in modo **leggibile**:
  1) **indentare (scrivere in modo ordinato) il codice**
  2) **definire le variabili poco prima del loro utilizzo**
  3) **dare alle variabili nomi dal significato inequivocabile**
  4) **usare espressioni dalla complessità limitata**
  5) **adoperare le parentesi anche se non necessario**
  6) **commentare le righe non facilmente leggibili**

- Per indentare si usa il tasto TAB (carattere di tabulazione) invece che tanti spazi. Entrambi i caratteri sono trasparenti, ma il TAB ha delle utili proprietà. Nella figura seguente abbiamo evidenziato i TAB con una freccia e gli spazi con un quadratino. Per indentare (caso a) usiamo due TAB per le prime due righe, e gli spazi per le successive.

```
→int x;
→int y;

□□□int x;
□□□int y;
```

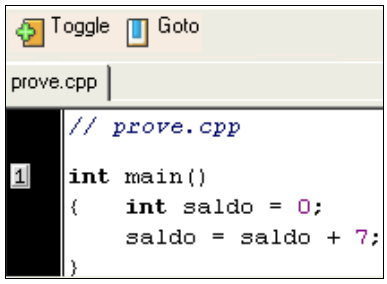
a)

```
{→int x;
 →int y;

{□□□int x;
□□□int y;
```

b)

- Supponiamo (caso b) di dover aggiungere una parentesi graffa. Mentre con il TAB la riga non si sposta avanti (rompendo l'allineamento tra righe), con gli spazi occorre rimettere a posto le righe successive. Il TAB tiene ferma la posizione del testo a destra fino a che c'è spazio per altri caratteri.
- Spesso è necessario cambiare il nome alle variabili. Questo può essere fatto in modo automatico. Selezionando dal menu *Search* → *Replace* appare una maschera in cui inserire il testo da cercare ed il testo con cui rimpiazzarlo.

- Esiste anche la funzione *Search* → *Find* che consente di trovare un termine.
- Un'altra comoda funzione è quella dei segnalibro (bookmark). Posizionandosi su una data riga, cliccare sul pulsante *Toggle* (v. fig. seguente) per inserire un segnalibro. Per riposizionarsi su quella riga cliccare sul pulsante *Goto* e poi indicare quale segnalibro.



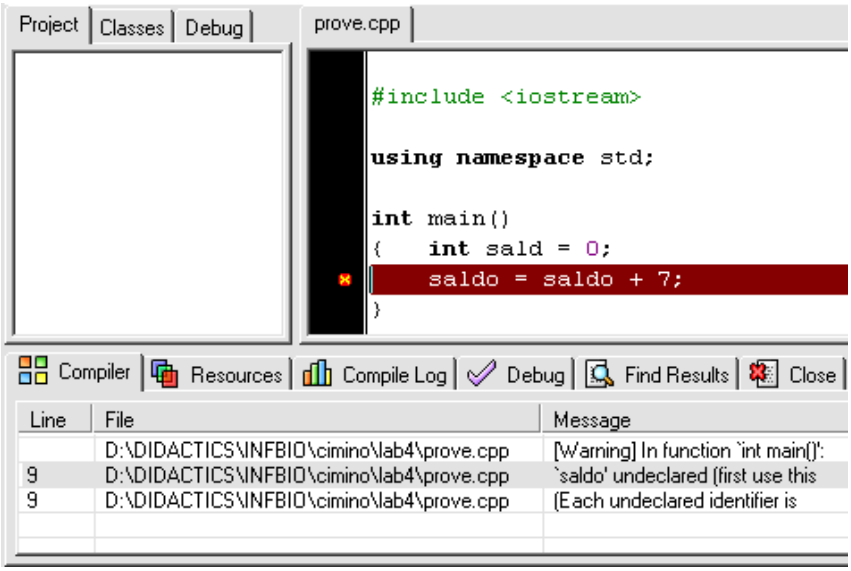
- **NOTA**: in basso a sinistra della finestra del DevC++ viene sempre indicata la riga e la colonna (nel formato `riga:colonna`) in cui si è posizionati.
- Se si commette un errore, cliccando sui bottoni undo (o redo)  si può tornare indietro (risp. ritornare avanti) alle versioni precedenti (risp. successive) del testo.

2. Errori sintattici

- Il tipo di errore più semplice da trovare è l'errore sintattico, un errore dovuto ad un

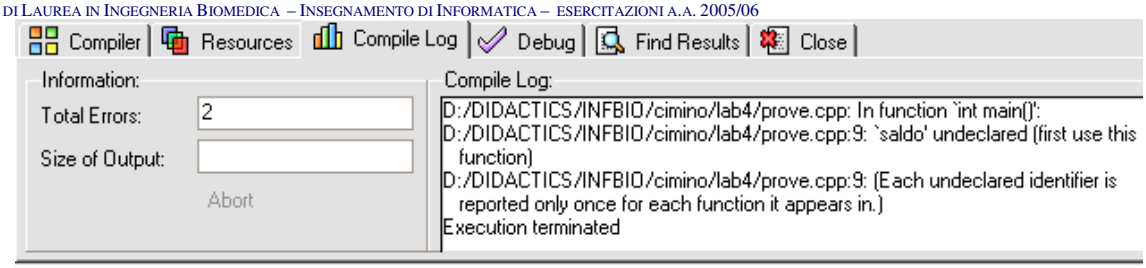
uso scorretto dei costrutti linguistici, detto anche **errore di compilazione** poiché viene segnalato dal compilatore.

- Ad esempio, se si scrive ***Main*** invece che ***main*** il compilatore del DevC++ segnala il messaggio ***undefined reference to 'WinMain@16'***. Ciò significa che si aspetta una funzione principale di nome "main" da riferire come punto di partenza del programma. Non è in grado di capire che "Main" è un nome simile a "main" per cui non segnala alcuna riga del codice in particolare.
- Nei casi più semplici il compilatore segnala esattamente la riga dove occorre intervenire. Es. se scrivo ***int saldo = 0:*** verrà segnalata esattamente tale riga, con messaggio di errore ***parse error before ":" token***, ossia *errore di analisi grammaticale prima del simbolo ":"*. Infatti *parsing* è l'attività di analisi grammaticale di un testo scritto in un linguaggio formale. *Token* è un simbolo adoperabile nel linguaggio (es. una parola chiave, un operatore o un identificatore). Quindi il compilatore non ammette il ":" finale al posto del ";".
- Spesso il compilatore non è in grado di segnalare il punto "x" in cui occorre intervenire, ma segnala il punto "y" (tipicamente dopo di "x") in cui non riesce ad andare avanti nel processo di traduzione. Nel seguente esempio



in cui inavvertitamente si dichiara il nome *sald* invece di *saldo* , viene segnalata la riga successiva con il messaggio ***'saldo' undeclared (first use this*** ossia *"saldo" non dichiarato*. In effetti è stata dichiarata una variabile con un altro nome, per cui non posso usare il nome *saldo* se prima non dichiaro cos'è.

- È possibile vedere il messaggio completo (*first use this function*) cliccando sulla scheda di log ("log" è un registro di trascrizione delle attività di un programma).



- **ATTENZIONE**: Spesso gli errori sono concatenati, per cui è meglio coreggerne alcuni tra i primi indicati (in ordine di scrittura) e poi ricompilare.

3. Errori semantici

- Un errore è semantico se il programma viene correttamente compilato, ma in esecuzione non fa correttamente ciò che deve fare. Ad esempio, un errore in una formula matematica in cui al posto di "+" si inserisce "*". Tali errori, comunemente chiamati **bug**, possono essere molto difficili da trovare e correggere.

- Consideriamo un programmino che scambia i valori di due variabili x ed y. Avviandolo ci accorgiamo che non funziona:

```
Inserisci x y 3 5
 x = 3, y = 5
 x = 5, y = 5
Premere un tasto per continuare . . .
```

- Ecco il codice sorgente del programma. Come facciamo a trovare l'errore ?

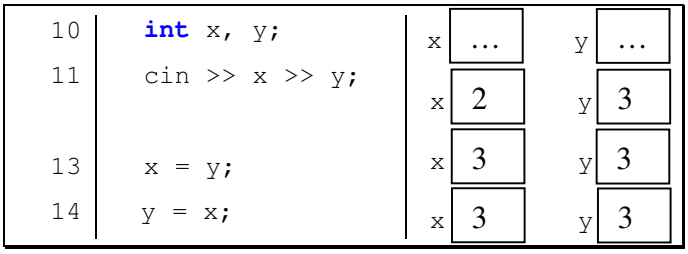
```
1   // scambia1.cpp
2
3   #include <iostream>
4
5   using namespace std;
6
7   int main()
8   {   cout << "Inserisci x y  ";
9
10      int x, y;
11      cin >> x >> y;
12      cout << " x = " << x << ", y = " << y << endl;
13      x = y;
14      y = x;
15      cout << " x = " << x << ", y = " << y << endl;
16
17      system("pause");
18  }
```

- Innanzitutto osserviamo che l'errore si trova probabilmente nelle istruzioni di scambio (righe 13-14). Infatti la linea 12 provvede a stampare subito i valori inseriti dall'utente, che risultano corretti.
- Se vogliamo controllare il valore della x dopo l'istruzione 13, inseriamo la riga


```
cout << x << endl; // debug
```

La pratica di inserire delle righe "cout" per stampare valori intermedi è spesso usata per scovare gli errori.

- **NOTA**: il commento `//debug` serve a ricordarsi che quella riga andrà poi tolta.
- Inserendo es. 2 e 3 come valori, ci accorgiamo che la x vale 3 dopo la riga 13. Del resto, questo poteva essere dedotto senza aggiungere la riga di stampa della x, poiché nella riga successiva si modifica solo la y e poi viene stampato il valore di entrambi.
- Provando con carta e penna (fig. seguente) ci accorgiamo che al rigo 13 perdiamo il valore 2 della x.



- Per risolvere tale problema, adoperiamo una variabile temporanea (o variabile di appoggio) t, in cui memorizzare il valore della x prima che questa prenda il valore della y, e poi assegnare alla y tale valore t.

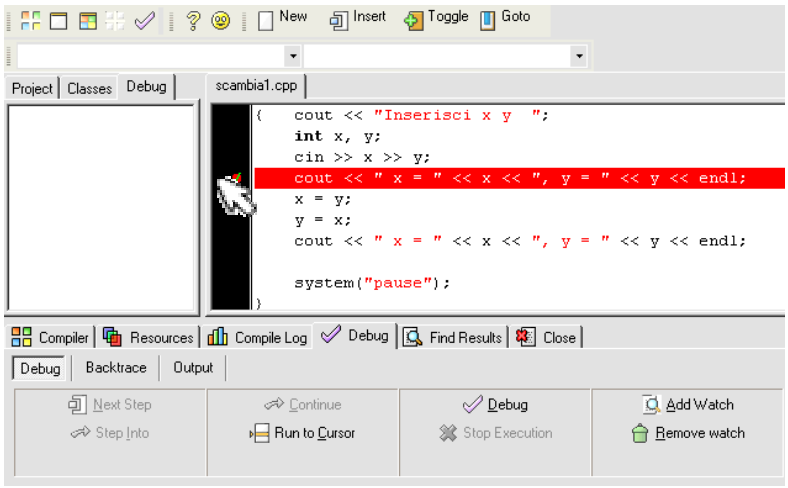
		x	y	t
10	`int x, y, t;`	...	...	...
11	`cin >> x >> y;`	2	3	...
13	`t = x`	2	3	2
14	`x = y;`	3	3	2
15	`y = t;`	3	2	2

4. Il debugging

- Per aiutare i programmatori in questa attività investigativa, è stato inventato il **debugger**, un sistema che consente di controllare l'operato di funzioni, variabili ed istruzioni, mentre il programma è in esecuzione.
- Innanzitutto, il debugger è in grado di sospendere il vostro programma in un dato punto (detto **breakpoint**) a partire dal quale vi interessa esaminare lo stato di avanzamento con maggior dettaglio.
- Per lanciare il programma nel debugger basta cliccare sul tasto debug  , oppure dal menu *Debug* → *debug* (scorciatoia: F8).
- Se non si hanno opportune informazioni di debugging configurate nel programma,

DevC++ chiederà se si vuole ricostruire (rebuild) il programma con tali informazioni abilitate. Dopo aver fatto ciò, si può riavviare di nuovo il debug, che produrrà un avvio del programma, ma senza particolari vantaggi.

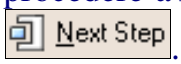
- Per un utilizzo proficuo del debugger, occorre impostare i punti di arresto (breakpoint) nel codice, quindi "fare un passo" (stepping) nel codice e visualizzare valori di variabili.
- Per **impostare i punti di arresto** cliccare con il tasto sinistro sulla banda nera a sinistra della riga, che verrà evidenziata (fig. seguente)



- A questo punto è possibile cliccare sul tasto di debug. L'applicazione partirà normalmente, ma si fermerà nel punto di arresto. La riga attualmente in esecuzione viene indicata in colore blu (fig. seguente).

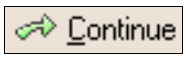
- **ATTENZIONE**: Se sono richiesti dei valori all'utente, comparirà anche la finestra nera di inserimento o visualizzazione dati. Inserire i valori e premere invio. È comodo ridimensionare e tenere sott'occhio entrambe le finestre, per avere la vista "interna" e quella "utente" del programma.

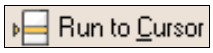
- Per visualizzare i valori correnti di alcune variabili, cliccare con il tasto destro nel pannello "Debug" a sinistra del codice → Add watch → e si introduce il nome della variabile. In tale pannello compariranno i valori sempre aggiornati delle variabili indicate (v. fig. seguenti)

- Per procedere avanti di una riga alla volta, cliccare sul tasto *Next Step* (scorciatoia: *F7*) 

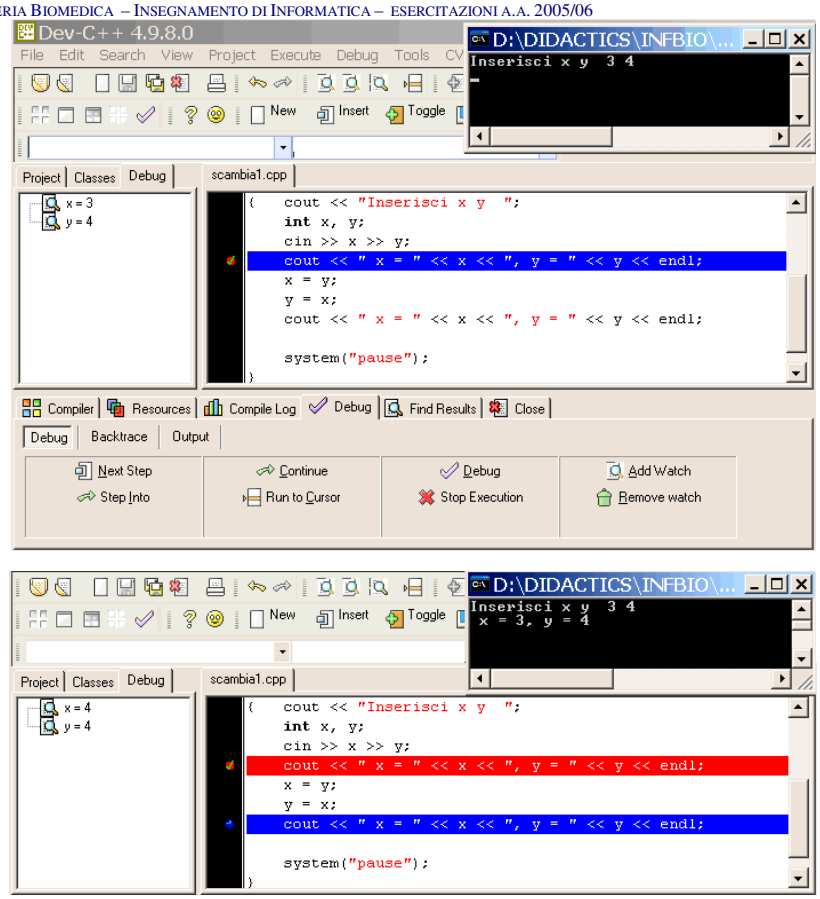
.

- **NOTA**: se vengono richiesti dati all'utente, il programma non prosegue finché tali dati non vengono inseriti.

- Per passare automaticamente al prossimo breakpoint, cliccare sul tasto 

- Il tasto 

consente di proseguire automaticamente dal cursore.



- Il breakpoint possono essere rimossi cliccando di nuovo sul simbolino rosso associato ad essi a sinistra della riga corrispondente.

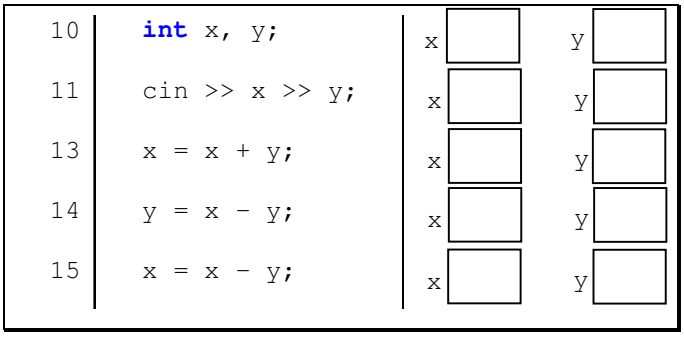
- Il tasto 

equivale a *step* nel caso in cui la riga di codice contenga istruzioni semplici (come l'assegnamento). Nel caso in cui si abbia una invocazione di funzione (argomento che verrà trattato successivamente), consente di passare alla esecuzione interna di quella funzione.

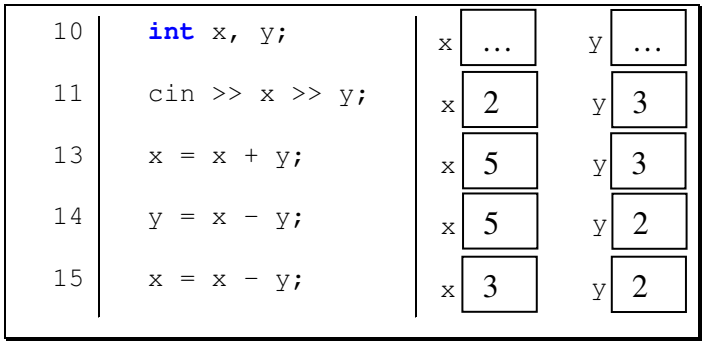
5. Il debugger come strumento di analisi

- Il debugger serve anche per capire cosa fanno alcune istruzioni difficili da leggere, se ad esempio il programmatore ha dimenticato di inserire dei commenti.

- Esercizio: cosa fanno le seguenti righe di codice ?



- Provando con il debugger (o compilando le caselle con carta e penna) si ha il seguente tracciato:



- Quindi il programma scambia i valori di x ed y senza usare variabili di appoggio.

- **ATTENZIONE**: non si usi questo come un buon esempio di programmazione: è preferibile la versione con la variabile di appoggio t (magari con un nome più significativo), molto più leggibile.

ESERCITAZIONE 5

LINGUAGGIO C++ : PRIME APPLICAZIONI CON L'ISTRUZIONE "IF"

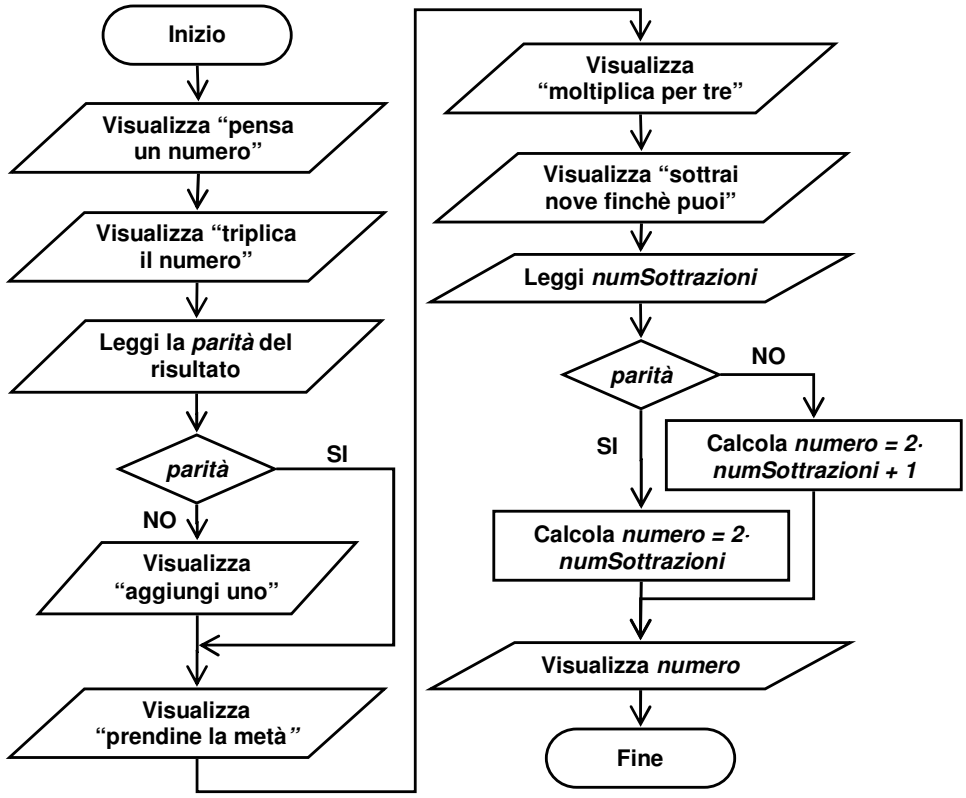
1. Indovina un numero

- Scrivere un programma che chiede all'utente di eseguire una serie di operazioni su un numero scelto a piacere, quindi domanda alcuni indizi su tali operazioni e indovina tale numero.

- In particolare, il programma: (a) chiede all'utente di pensare ad un numero, (b) quindi di triplicarlo; (c) chiede se il risultato è dispari, ed in tal caso di aggiungere uno; (d) prenderne la metà, (e) moltiplicarla per tre, (f) sottrarre il numero nove tante volte finchè possibile; (g) si chiede infine quante volte è stato sottratto il numero nove, poniamo n. Se il risultato al punto (c) è pari, il sistema dichiara che il numero pensato in partenza è 2n, altrimenti 2n+1. Ecco due esempi di funzionamento.

Messaggi stampati dal programma	*Calcoli dell'utente*	
-PENSA UN NUMERO	$x_0 = 5$	$x_0 = 4$
-TRIPLICA TALE NUMERO	$x_1 = 15$	$x_1 = 12$
-VIENE UN NUMERO PARI (S/N)? N	*N*	*S*
-AGGIUNGI UNO	$x_2 = 16$	
-PRENDINE LA META'	$x_3 = 8$	$x_2 = 6$
-MOLTIPLICA IL RISULTATO PER TRE	$x_4 = 24$	$x_3 = 18$
-SOTTRAI NOVE, TANTE VOLTE FINO A CHE E' POSSIBILE	$x_5 = 24-9 = 15,$	$x_4 = 19-9 = 9,$
	$x_6 = 15-9 = 6$	$x_5 = 9-9 = 0$
-QUANTE VOLTE HAI POTUTO SOTTRARRE IL NOVE? 2	*2*	*2*
-IL NUMERO PENSATO E' 5.	$x_7 = 2*2+1 = 5$	$x_7 = 2*2 = 4$

- In questo problema, la maggioranza dei calcoli viene svolta dall'utente. Il computer esegue solo una semplice operazione finale. Diagramma di flusso e soluzione:

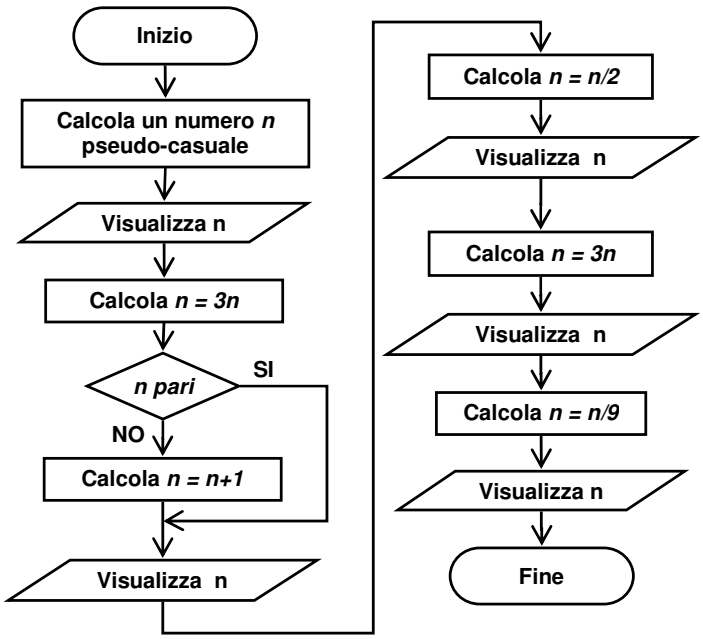


```
// indovino.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "- PENSA UN NUMERO" << endl;
    system("pause");
    cout << "- TRIPLICA TALE NUMERO" << endl;
    system("pause");
    cout << "- VIENE UN NUMERO PARI (S/N)? ";
    char pari;
    cin >> pari;
    if (pari == 'N')
    {   cout << "- AGGIUNGI UNO" << endl;
        system("pause");
    }
    cout << "- PRENDINE LA META'" << endl;
    system("pause");
    cout << "- MOLTIPLICA IL RISULTATO PER TRE" << endl;
    system("pause");
    cout << "- SOTTRAI NOVE, TANTE VOLTE FINCHE' POSSIBILE" << endl;
    system("pause");
    cout << "- QUANTE VOLTE HAI POTUTO SOTTRARRE IL NOVE? ";
    int numSottr;
    cin >> numSottr;
    cout << "- IL NUMERO PENSATO E' ";
    if (pari == 'S')
        cout << 2*numSottr;
    else
        cout << 2*numSottr+1;
    cout << "." << endl;
    system("pause");
}
```


NOTA: l'istruzione "*IF (condizione)*" consente al flusso di esecuzione di seguire diverse direzioni, sulla base di una condizione logica.

2. Pensa un numero ed esegui dei calcoli

Scrivere un programma che esegue esattamente le operazioni indicate dall'indovino, visualizzando ad ogni passo l'operazione ed il risultato parziale.



```
// utente.cpp

#include <iostream>

using namespace std;

int main()
{   long int timeStamp = time(NULL);            //(1)
    int n = timeStamp - (timeStamp/100)*100;    //(2)

    cout << "- HO PENSATO UN NUMERO (" << n << ")" << endl;
    system("pause");

    n = n*3;
    cout << "- L'HO MOLTIPLICATO PER TRE (" << n << ")" << endl;
    system("pause");

    if ((n % 2) == 0)
    {   cout << "- VIENE UN NUMERO PARI" << endl;
        system("pause");
    }
    else
    {   cout << "- VIENE UN NUMERO DISPARI" << endl;
        n = n + 1;
        cout << "- HO AGGIUNTO UNO (" << n << ")" << endl;
    }
    n = n/2;
    cout << "- NE HO PRESO LA META' (" << n << ")" << endl;
    system("pause");

    n = n*3;
    cout << "- HO MOLTIPLICATO IL RISULTATO PER TRE (" << n << ")" << endl;
    system("pause");
...
```



```
...
    n = n/9;
    cout << "- HO SOTTRATTO NOVE FINCHE' POSSIBILE (" << n << " VOLTE)" << endl;
    cout << "- QUALE NUMERO HO PENSATO?" << endl;

    system("pause");
}
/* Note
(1) restituisce il numero di secondi passati dalla mezzanotte dell'1/1/1970 in
    accordo al clock di sistema
(2) restituisce un numero tra 0 e 99.
*/
```

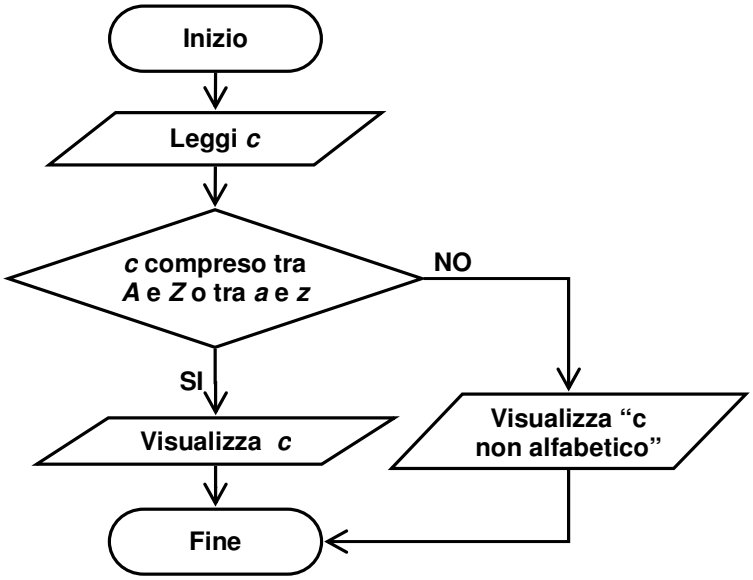
Esempio di esecuzione dei due programmi:

indovino.exe	***utente.exe***
- PENSA UN NUMERO **Premere un tasto per continuare . . .** **- TRIPLICA TALE NUMERO** **Premere un tasto per continuare . . .** **- VIENE UN NUMERO PARI (S/N)? S** **- PRENDINE LA META'** **Premere un tasto per continuare . . .** **- MOLTIPLICA IL RISULTATO PER TRE** **Premere un tasto per continuare . . .** **- SOTTRAI NOVE, TANTE VOLTE FINCHE' POSSIBILE** **Premere un tasto per continuare . . .** **- QUANTE VOLTE HAI POTUTO SOTTRARRE IL NOVE? 21** **- IL NUMERO PENSATO E' 42.** **Premere un tasto per continuare . . .**	***- HO PENSATO UN NUMERO (42)*** ***Premere un tasto per continuare . . .*** ***- L'HO MOLTIPLICATO PER TRE (126)*** ***Premere un tasto per continuare . . .*** ***- VIENE UN NUMERO PARI*** ***Premere un tasto per continuare . . .*** ***- NE HO PRESO LA META' (63)*** ***Premere un tasto per continuare . . .*** ***- HO MOLTIPLICATO IL RISULTATO PER TRE (189)*** ***Premere un tasto per continuare . . .*** ***- HO SOTTRATTO NOVE FINCHE' POSSIBILE (21 VOLTE)*** ***- QUALE NUMERO HO PENSATO?*** ***Premere un tasto per continuare . . .***

- Per generare dei numeri pseudo-casuali, si è adoperata la funzione **time** che restituisce i secondi trascorsi da una data di riferimento. Il numero n rappresenta le ultime due cifre di time, per cui un numero tra 0 e 99.

3. Controllo sui dati in ingresso

- Scrivere un programma che controlla se un carattere dato dall'utente è alfabetico.

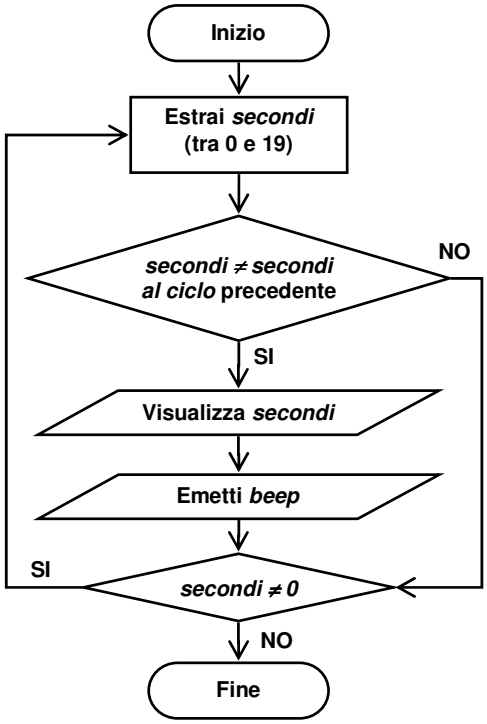



```
// carattere.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserisci un carattere: ";
    char c;
    cin >> c;
    if ( (c>='A') && (c<='Z') || (c>='a') && (c<='z') )
      cout << c << endl;
    else
      cout << "Il carattere non e' una lettera dell'alfabeto " << endl;
    system("pause");
}
```

ESERCITAZIONE 6
LINGUAGGIO C++ : ISTRUZIONI ITERATIVE

1. Contatore dei secondi

- Scrivere un programma che visualizza l'avanzare dei secondi in una posizione fissa dello schermo, emettendo un beep ad ogni secondo, iniziando con un numero a caso tra 0 e 19 e fermandosi dopo il 19° secondo.
- Diagramma di flusso:



- **NOTA**: il programma controlla ripetutamente lo scorrere del tempo; se la cifra dei secondi è cambiata rispetto al passo precedente, allora stampa il nuovo valore ed emette un beep.
- Le istruzioni iterative consentono di **ritornare indietro** nei diagrammi di flusso, ripercorrendo le medesime operazioni per più volte.

- Soluzione:

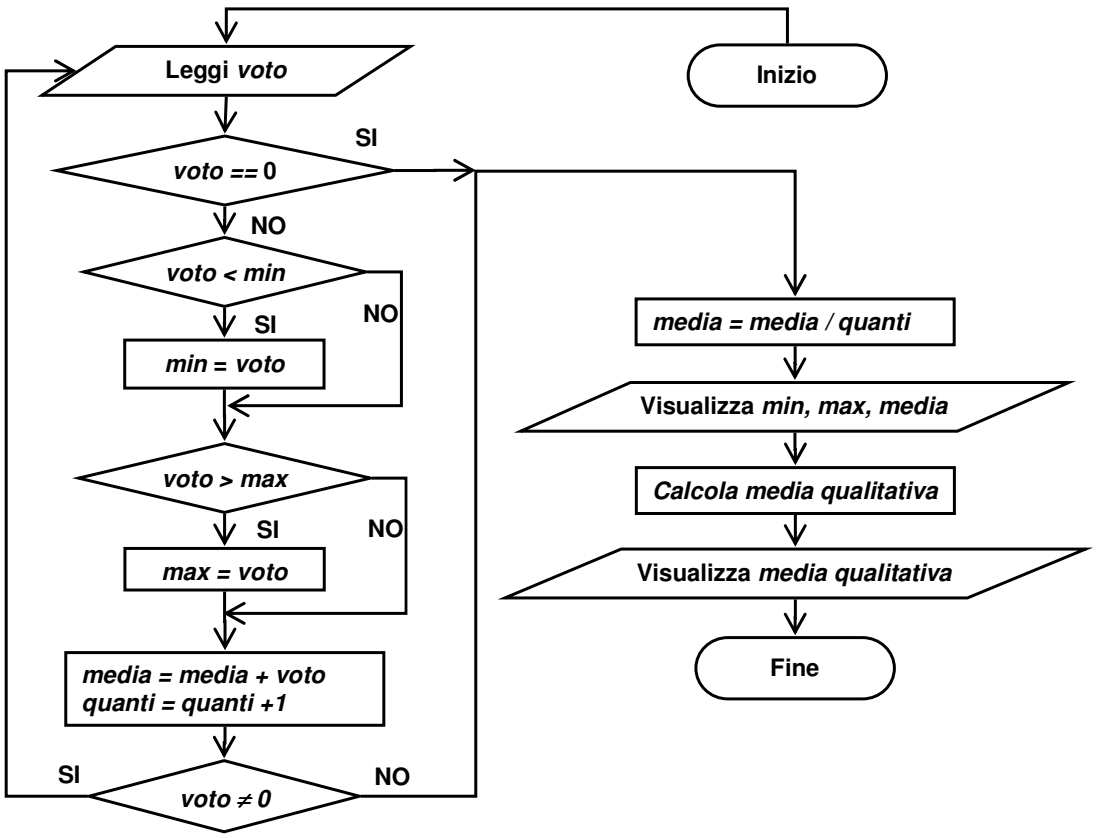
```
// tempo.cpp

#include <iostream>

using namespace std;

int main()
{   long int secondi, secondiPrec = 20;
    do
    {   secondi = time(NULL) % 20;
        if (secondi != secondiPrec)
        {   cout << '\r' << secondi << '\t';
            cout << char(7);
            secondiPrec = secondi;
        }
    }
    while (secondi != 0);
    system("pause");
}
```

- **NOTA**: il carattere `'\r'` consente di riposizionarsi all'inizio della riga corrente dello schermo, se non si è già andati accapo, per sovrascriverne il contenuto

2. Analisi del rendimento universitario

- Scrivere un programma che consente di inserire una sequenza di voti, ne calcola il valore minimo, massimo, medio e lo visualizza. Per terminare la sequenza di voti si adoperi lo zero.

- Visualizzare anche la seguente *valutazione qualitativa* della media: *sufficiente* in [18, 21), *discreta* in [21, 24), *buona* in [24, 27), *distinta* in [27, 30), *ottima* in [30, 33), *eccellente* in [33].
- Visualizzare i numeri con due sole cifre decimali.
- **NOTA**: il minimo, il massimo e la media possono essere calcolati senza memorizzare l'elenco di voti, ossia "in linea". Per il minimo, è sufficiente avere una variabile di appoggio *min* in cui ad ogni nuovo voto inseriamo il valore minimo tra tale voto ed il valore della variabile stessa. Idem per il massimo. Per la media basta sommare i voti man mano inseriti e poi alla fine dividere tale soma per il numero di voti.
- **NOTA**: per il calcolo della valutazione qualitativa, possiamo creare una funzione che per ogni intervallo di valutazione produce un numero intero diverso. Quindi mediante una istruzione **switch** possiamo stampare un messaggio diverso a seconda dell'intero calcolato.
- **NOTA**: l'approssimazione a due sole cifre decimali si può eseguire come nel seguente esempio: `media = 21,33333...`; moltiplicando per 100 si ha `media = 2133,3333`; arrotondando all'intero più vicino si ha `media = 2133`; infine, dividendo per 100 si ha `media = 21,33`.
- Diagramma di flusso:



- Soluzione:

```
// voti.cpp

#include <iostream>

using namespace std;

int main()
{   cout << "Inserire i voti, separati da spazi e terminanti con zero:" << endl;
    const int     MIN = 18, MAX = 33;
    const double  FATTORE = 100;
    int           min = 33, max = 18, voto, quanti = 0;
    double        media = 0;

    do
    {    cin >> voto;
         if ( voto==0 )
             break;
         if ( voto < min )
             min = voto;
         if ( voto > max )
             max = voto;
         media  = media + voto;
         quanti = quanti + 1;
    }
    while (voto!=0);

    media = media/quanti;
    cout << "voto minimo:\t" << min << endl
         << "voto massimo:\t" << max << endl
         << "voto medio:\t" << (int)( media*FATTORE + 0.5 )/FATTORE;
    ...
```



```
    ...
    switch( (int)( (media-MIN)/(MAX-MIN)*5) )
    {    case 0:   cout << "\t(sufficiente)";   break;
         case 1:   cout << "\t(discreto)";      break;
         case 2:   cout << "\t(buono)";         break;
         case 3:   cout << "\t(distinto)";      break;
         case 4:   cout << "\t(ottimo)";        break;
         default:  cout << "\t(eccellente)";
    }
    cout << endl;

    system("pause");
}
```

- L'istruzione `(int)(x)` approssima il reale x sempre per difetto. L'istruzione `(int)(x+0.5)` arrotonda all'intero più vicino. Es. `x=1.2; (int)(x+0.5)= (int)(1.7)=1.` Invece `x=1.6; (int)(x+0.5)= (int)(2.1)=2.`

3. Comunicazione empatica

- Scrivere un programma in grado di capire i messaggi di una persona, semplicemente attraverso indicazioni del tipo SI/NO. Tali messaggi sono comunicati lettera per lettera.
- Per capire a quale lettera dell'alfabeto si riferisca l'utente, il programma segue una **ricerca binaria**, che consiste nei seguenti passi.
- Inizialmente, si cerca una lettera in tutto l'intervallo 'A'...'Z', e si chiede all'utente

se tale la lettera venga prima o dopo la lettera centrale 'M'. Se viene prima, allora si cerca nel sotto-intervallo 'A'...'L', altrimenti nel sotto-intervallo, 'M'...'Z'.

- Quindi si ripete il ragionamento sul nuovo intervallo di lunghezza inferiore, ponendo sempre la domanda in riferimento alla lettera centrale. E così via, fino ad avere un intervallo composto da due lettere identiche, che rivela la lettera cercata.
- Il programma prosegue nel cercare nuove lettere sino a che non indicato diversamente dall'utente.
- Esempio di funzionamento:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
dopo M (s/n)? n
dopo G (s/n)? n
dopo D (s/n)? s
dopo F (s/n)? n
dopo E (s/n)? n
la lettera n.1 e` E
 Vuoi proseguire (s/n)?
```

- Dopo aver svolto la soluzione, per visualizzare ad ogni passo l'intervallo di ricerca si inserisca la seguente riga dopo le righe *(1)* e *(2)*:

```
cout <<  "[" << char(min) << "," << char(max) << "] ";
```



```cpp
// comunica.cpp

#include <iostream>

using namespace std;

int main()
{   for (int i=65; i<=90; i++)
        cout << char(i) << ' ';
    cout << endl;

    int  min, max, med, cont = 1;
    char c;

    do
    { min = 65, max = 90;
      do
        {   med = (min + max)/2;
            cout <<  "dopo " << char(med) << " (s/n)? ";  //(1)
            cin  >> c;
            if (c == 'n')
              max = med;
            else
              min = med+1;
        }
      while ( max>min );                                  //(2)
      cout << "la lettera n." << cont << " e` " << char(min) << endl;
      cont++;
      cout << " Vuoi proseguire (s/n)? ";
      cin >> c;
    }
    while ( (c!='n') );
    system("pause");
}
```

ESERCITAZIONE 7

LINGUAGGIO C++ : INTRODUZIONE A FUNZIONI, FILE ED ARRAY.

1. Il gioco del tris tra giocatori umani

- Scrivere un programma che realizza il gioco del tris (o filetto) tra due giocatori umani (Fig.1).
- In particolare, ciascun giocatore ha un simbolo ("x" oppure "o" a scelta) ed una mossa consiste nel posizionare, a turno, il proprio simbolo in una cella libera.
- Vince chi per primo riesce a comporre una sequenza di tre simboli uguali, allineati in orizzontale, verticale oppure diagonale.
- Se la scacchiera viene riempita senza alcun vincitore, vince chi ha più sequenze (non sovrapposte) di due simboli.

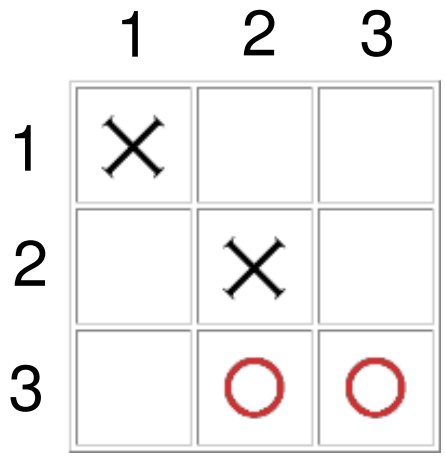


Fig.1 – La scacchiera nel tris. Ciascuna casella viene identificata con le corrispondenti coordinate riga colonna. Ad esempio, le coordinate 2 2 corrispondono alla casella centrale.

- Scomponendo il problema principale in sotto-problemi (Fig.2), si hanno le seguenti tre funzioni: "invididua tris", "visualizza celle", "sono celle piene".

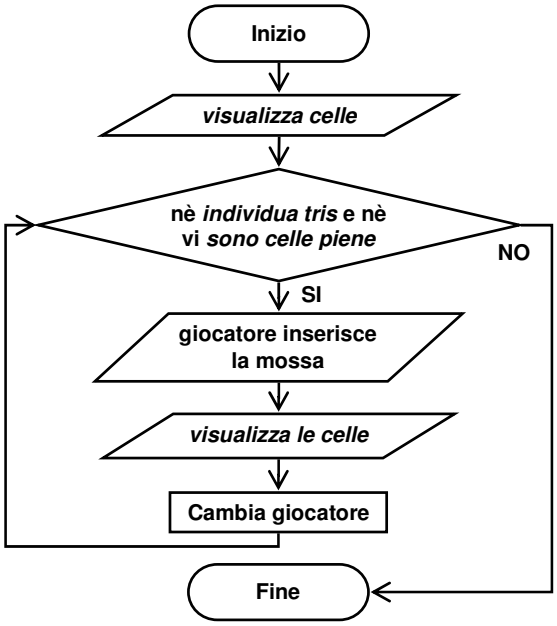


Fig.2 – programma *principale*

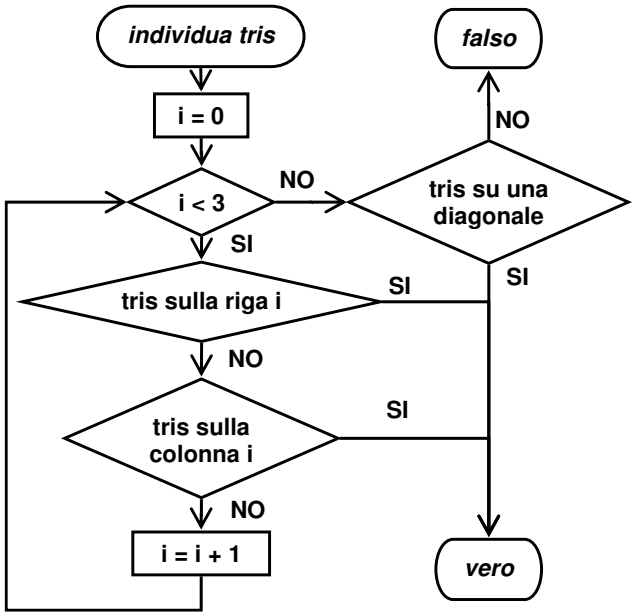


Fig.3 – Funzione *individua tris*

- Per individuare un tris (Fig.3), basta controllare su tutte le righe, su tutte le colonne, e sulle due diagonali.

- Per visualizzare tutte le celle, si visualizza per ogni riga tutti gli elementi della riga.
- Per controllare se tutte le celle sono piene, si leggono tutte le celle (come nella visualizzazione) e, se qualcuna è vuota, si esce restituendo *false*.

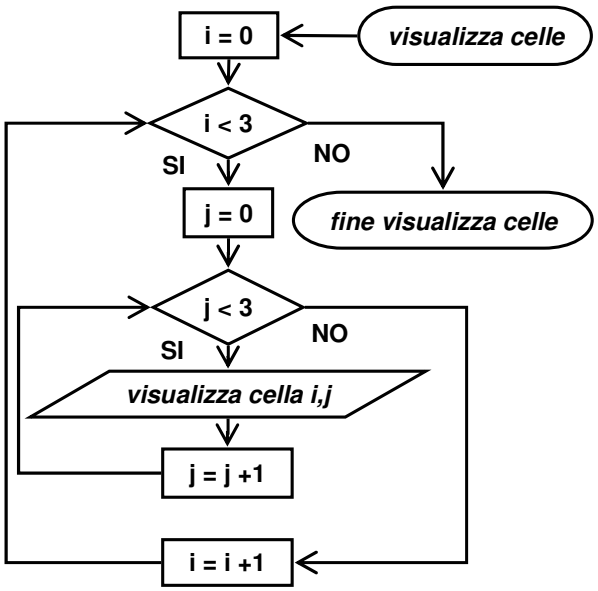


Fig.4 – funzione *visualizza celle*

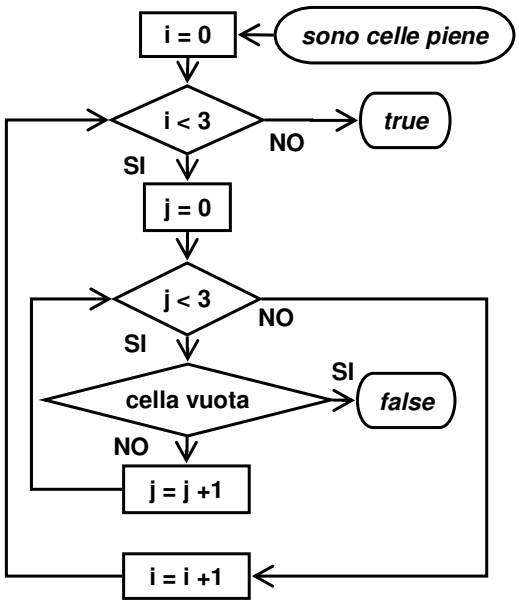


Fig.5 – funzione *sono celle piene*

- Soluzione:


```
// tris.cpp

#include <iostream>

using namespace std;

char celle[3][3] = {' ',' ',' ',
                    ' ',' ',' ',
                    ' ',' ',' ' };
char giocatore = 'x';
char avversario = 'o';

void visualizzaCelle()
{ for (int i=0; i<3; i++)
  { for (int j=0; j<3; j++)
      cout << celle[i][j] << '|';
    cout << endl;
  }
}

bool sonoCellePiene()
{ for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
      if (celle[i][j] == ' ')
        return false;
  return true;
}

void cambiaGiocatore()
{ if (giocatore == 'x')
  {  giocatore = 'o';
     avversario = 'x';
  }
  else
  {  giocatore = 'x';
     avversario = 'o';
  }
}
```

```
void chiediMossa()
{ int i,j;
  cin >> i >> j;
  celle[i-1][j-1] = giocatore;
}

bool individuaTris()
{ for (int i=0; i<3; i++)
  { if ( (celle[i][0] != ' ') &&
         (celle[i][0] == celle[i][1]) &&
         (celle[i][0] == celle[i][2]) )
      return true;
    if ( (celle[0][i] != ' ') &&
         (celle[0][i] == celle[1][i]) &&
         (celle[0][i] == celle[2][i]) )
      return true;
  }
  if ( (celle[0][0] != ' ') &&
       (celle[0][0] == celle[1][1]) &&
       (celle[0][0] == celle[2][2]) )
      return true;
  if ( (celle[0][2] != ' ') &&
       (celle[0][2] == celle[1][1]) &&
       (celle[0][2] == celle[2][0]) )
      return true;
  return false;
}

int main()
{ visualizzaCelle();
  while(!individuaTris() && !sonoCellePiene())
  { chiediMossa();
    system("cls");
    visualizzaCelle();
    cambiaGiocatore();
  }
  system("pause");
}
```

ATTENZIONE: per semplicità, il programma non controlla se i giocatori riempiono una cella già piena.

NOTA: La variabile carattere *giocatore* contiene il simbolo di chi deve fare la mossa, mentre la variabile *avversario* il simbolo di chi l'ha appena fatta. Tali variabili, assieme alla matrice *celle*, sono **globali**, ossia esterne a tute le funzioni, e quindi visibili dall'interno di ciascuna di esse. Viceversa, tutte le variabili interne (**locali**) ad una funzione sono visibili solo dentro la rispettiva funzione. La funzione *individuaTris*() controlla anche che le celle non siano vuote, altrimenti una scacchiera tutta vuota darebbe un tris.

2. Il gioco del tris con un giocatore virtuale a tattica prestabilita

Modificare il programma precedente in modo che uno dei due giocatori sia virtuale, e conduca la partita seguendo le seguenti tattiche prestabilite.

Controlla prima se può vincere (funzione `vinci()`), provando a posizionare il proprio simbolo in una cella libera (e provando per tutte le celle libere) per vedere se è possibile un tris in una sola mossa. In tal caso la esegue.

Se non è possibile vincere in una sola mossa, il giocatore virtuale prova a vedere se è l'avversario a poter vincere in una sola mossa occupando una data cella. In tal caso cerca di ostacolarlo (funzione `ostacola()`) occupando la cella mediante il proprio simbolo. Se vi fossero due celle di questo tipo l'avversario vincerebbe

comunque, in quanto in una sola mossa se ne potrebbe occupare una soltanto.

Se non avviene alcuno dei casi precedenti, il giocatore esegue una mossa casuale.

I seguenti diagrammi di flusso descrivono le funzioni ad un livello più generico rispetto a quello dei diagrammi precedenti: non si rappresentano le singole istruzioni (scomposizione a "grana fine") ma si esegue una scomposizione a "grana grossa"

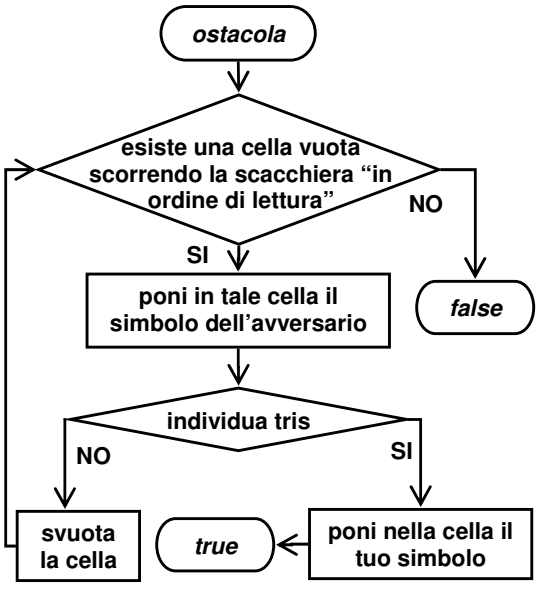


Fig.6 – funzione *ostacola*

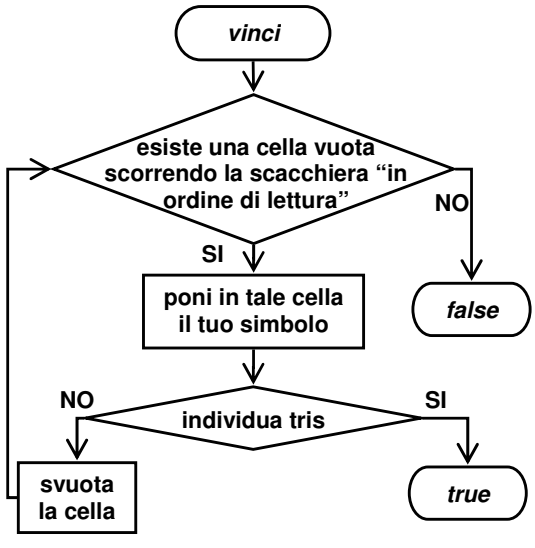


Fig.7 – funzione *vinci*

Soluzione in linguaggio C++:

```
// tris2.cpp
// le altre funzioni rimangono
// inalterate rispetto a tris.cpp

bool ostacola()
{ for (int i=0; i<3; i++)
   for (int j=0; j<3; j++)
     if (celle[i][j] == ' ')
     { celle[i][j] = avversario;
       if ( individuaTris() )
       {  celle[i][j] = giocatore;
          return true;
       }
       else
          celle[i][j] = ' ';
     }
 return false;
}

bool vinci()
{ for (int i=0; i<3; i++)
   for (int j=0; j<3; j++)
     if (celle[i][j] == ' ')
     { celle[i][j] = giocatore;
       if ( individuaTris() )
          return true;
       else
          celle[i][j] = ' ';
     }
 return false;
}
```

```
void eseguiMossa()
{ if (vinci())
    return;
  if (ostacola())
    return;
  srand(time(NULL));
  int i,j;
  do
  { i = rand()%3;
    j = rand()%3;
  }
  while (celle[i][j] != ' ');
  celle[i][j] = giocatore;
}

int main()
{ visualizzaCelle();
  while(!individuaTris() && !sonoCellePiene())
  { if (giocatore=='x')
      eseguiMossa();
    else
      chiediMossa();
    system("cls");
    visualizzaCelle();
    cambiaGiocatore();
  }
  system("pause");
}
```


NOTE: le funzioni *ostacola()* e *vinci()*, quando restituiscono *false* non hanno eseguito alcuna modifica sulla scacchiera. La funzione *eseguiMossa()* inizialmente prova ad eseguire entrambe queste funzioni. Se nessuna di esse restituisce *true*, prova a generare due coordinate casuali fino a che non individua una cella vuota. la funzione *main()* è stata modificata in modo che al turno del giocatore "x" (virtuale) corrisponda l'esecuzione di una mossa, all'altro turno la richiesta di una mossa.

NOTA: le funzioni hanno sempre un verbo come nome, mentre le variabili un sostantivo. Infatti le funzioni eseguono delle azioni, le variabili rappresentano entità.

3. Il gioco del tris con un giocatore artificiale in grado di apprendere

Modificare il programma precedente in modo che il giocatore virtuale (macchina) conduca la partita imitando il giocatore reale (umano). In particolare, la macchina memorizza tutte le mosse del giocatore reale in un file. Quando è il suo turno, cerca in tale file se in passato vi è stata una situazione identica per l'umano. In tal caso esegue la medesima mossa che eseguì l'umano, altrimenti esegue una mossa a caso.

Per semplicità, la macchina memorizza ogni mossa senza controllare se la medesima mossa era stata già memorizzata, quindi è possibile che vi siano dei duplicati. Inoltre essa memorizza ogni mossa senza controllare che sia una mossa buona o cattiva, per cui l'umano dovrebbe essere un giocatore esperto.

Per fare in modo che le situazioni presentate all'umano si ripropongano alla

macchina, la scelta di chi inizia per primo è casuale, non più fissata alla macchina.

- Per ogni mossa, viene salvato nel file lo stato della scacchiera e le coordinate della mossa eseguita. Lo stato della scacchiera viene codificato in un numero intero di 9 cifre, corrispondenti ai valori delle celle in "ordine di lettura". In particolare, *1* corrisponde a cella vuota, *2* al simbolo "o" e *3* al simbolo "x". Le codifiche *2* e *3* sono invertire per il confronto con le mosse eseguite, in quanto bisogna cercare una situazione in cui l'avversario era al posto della macchina.

- **NOTA**:La funzione *chiediMossa()* è simile alla omonima dell'esercizio precedente, tranne per il fatto che memorizza la mossa codificata all'interno del file. Il file viene aperto in modalità "**out or append**", cioè "scrittura" se il file ancora non esiste (e quindi va creato), "aggiunta" se già esso esiste.

- La funzione *codificaCelle(o,x)* ha due parametri interi di ingresso, *o* ed *x*, che corrispondono al valore numerico da associare ai simboli 'o' ed 'x' rispettivamente. Essa scorre tutta la scacchiera e, per ogni simbolo, aggiunge una cifra corrispondente a destra (moltiplica per 10 e somma la cifra). Il numero finale viene restituito dalla funzione come risultato. Es. una scacchiera vuota corrisponde a "111111111", una scacchiera con una "x" nel centro corrisponde a "111131111".

- La funzione *eseguiMossa()* codifica la scacchiera attuale (invertendo le cifre dei simboli "x" ed "o") e legge il file in cerca di una codifica identica. Per il resto è identica alla omonima funzione dell'esercizio precedente.


```
// tris3.cpp
// rispetto a tris2.cpp, togliere le
// funzioni ostacola() e vinci(), e
// introdurre le seguenti.
#include <fstream>

int codificaCelle(int o, int x)
{ int codice = 0;
  for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
      switch(celle[i][j])
      { case ' ':
          codice = codice*10+1;
          break;
        case 'o':
          codice = codice*10+o;
          break;
        case 'x':
          codice = codice*10+x;
      }
  return codice;
}

void chiediMossa()
{ int i,j;
  cin >> i >> j;
  fstream memoria;
  memoria.open("tris.txt",
               ios::app | ios::out);
  memoria << codificaCelle(2,3)
          << '\t' << i-1 << '\t'
          << j-1 << endl;
  memoria.close();
  celle[i-1][j-1] = giocatore;
}
```

```
void eseguiMossa()
{ int codifica = codificaCelle(3,2);
  int codificaPassata = 0;
  int i,j;
  bool trovata = false;
  fstream memoria;
  memoria.open("tris.txt", ios::in);
  while(memoria >> codificaPassata)
  {  memoria >> i >> j;
     if (codificaPassata == codifica)
     { cout << char(7);
       trovata=true;
       break;
     }
  }
  if (!trovata)
  { srand(time(NULL));
    do
    { i = rand()%3;
      j = rand()%3;
    }
    while (celle[i][j] != ' ');
  }
  celle[i][j] = giocatore;
  memoria.close();
}

int main()
{ srand(time(NULL));
  if ((rand()%2) == 0)
    cambiaGiocatore();
  // segue come in tris2.cpp ...
}
```

ESERCITAZIONE 8

LINGUAGGIO C++ : PUNTATORI, PARAMETRI, STRINGHE.

1. Decisioni di gruppo

- Scrivere un programma che consente ad un gruppo di persone di prendere decisioni senza incontri preliminari, su questioni di tipo consueto e ripetitivo quali il luogo e la data di un incontro.
- In particolare, ciascun membro formula (a rotazione) la propria proposta scrivendo su una *lavagna* e prendendo visione delle proposte altrui (Fig.1).

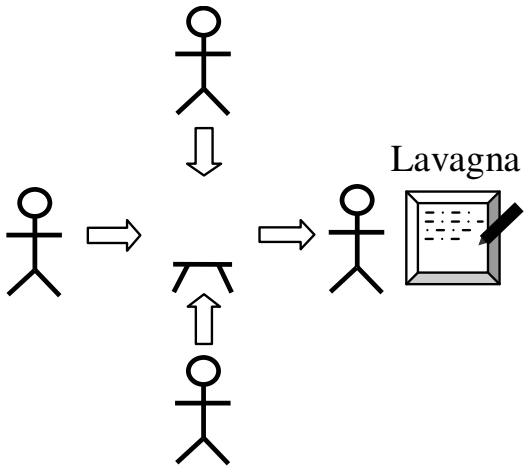


Fig.1 – Lavagna per decision making.

- Quando esiste una proposta identicamente formulata per la "metà più uno" dei partecipanti, la lavagna visualizza tale proposta. Altrimenti i partecipanti continuano a riformularle iterativamente, possibilmente tenendo conto della proposta più accreditata allo scopo di far giungere il gruppo ad una convergenza di opinioni.
- Esempio di utilizzo e diagramma di flusso principale:


```
1> Andiamo al cinema.
2> Andiamo in discoteca.
3>
messaggio n.3 (termina con '.')?
Andiamo a teatro.
```

```
1> Andiamo a teatro, alle 19.
2> Dai! andiamo in discoteca.
3> Andiamo a teatro.
messaggio n.3 (termina con '.')?
Andiamo a teatro, alle 20.
```

```
1> Andiamo a teatro, alle 19.
2> Dai! andiamo in discoteca.
3> Andiamo a teatro, alle 20.
messaggio n.1 (termina con '.')?
Andiamo a teatro, alle 20.
```

```
1> Andiamo a teatro, alle 20.
2> Dai! andiamo in discoteca.
3> Andiamo a teatro, alle 20.
decisione:  Andiamo a teatro, alle 20.
Premere un tasto per continuare . . .
```

Fig.2 – esempio di utilizzo

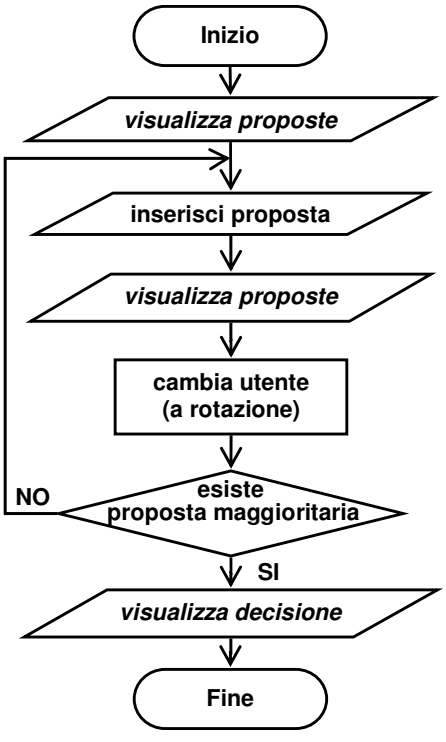


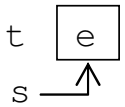
Fig.3 – Funzione principale

RIEPILOGO DEI PRINCIPALI CONCETTI SUI PUNTATORI

- Una variabile di nome **t** e di tipo **T** è un contenitore di valore di tipo **T**.

- Una variabile di tipo **T*** è un puntatore a variabile di tipo **T**. La '*' dopo un tipo significa 'puntatore a oggetto di tipo':

```
char t = 'e';
char* s = &t;
// *s nome alternativo a t
```



T* ⇔ puntatore a oggetto di tipo T

- Il puntatore contiene solo indirizzi. La '&' prima di un nome significa 'indirizzo di':

&t ⇔ indirizzo di t

- Non serve conoscere tali indirizzi, basta poterli assegnare ad un puntatore.

T* s = &t

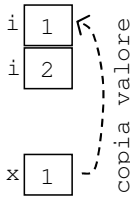
- Con un puntatore ad una variabile si può leggerne il valore. La '*' prima di un puntatore significa "oggetto puntato da"

***s ⇔ oggetto puntato da s**

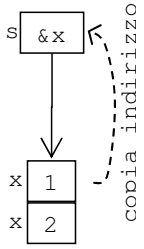
- In altri termini, ***s** è un altro nome per la variabile **t**. A cosa serve avere due nomi per la stessa variabile? Es. per consentire alle funzioni di modificare il contenuto di variabili non visibili. É sufficiente passare come parametro un puntatore alla variabile (passaggio per indirizzo), e quindi adoperare tale puntatore per avere un altro nome "esterno" per quella variabile.

- Nell'esempio, solo la versione di destra della funzione *incrementa* è corretta:

```
void incrementa(int i)
{ i++;              //(2)
}
int main()
{ int x = 1;
  incrementa(x);   //(1)
}
// nella chiamata (1) il pas-
// saggio è per valore, i=x.
// in (2) non si modifica x,
// bensì la i.
```




```
void incrementa(int* s)
{ (*s)++;           //(2)
}
int main()
{ int x = 1;
  incrementa(&x);   //(1)
}
// alla chiamata (1) il pas-
// saggio è per indirizzo,s=&x
// in (2) si modifica x.
```

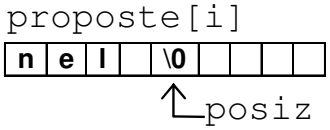


- Gli elementi di un array **v** hanno nome **v[i]**, per cui avranno indirizzo **&v[i]**. Nel caso particolare dell'elemento iniziale, si può usare **&v[0]** o semplicemente **v**. Ciò significa che il nome di un array, da solo, viene valutato come indirizzo della prima cella. Mentre il nome di una variabile di tipo predefinito (int, char, double,...) viene valutato come il valore della variabile. Quindi, se passiamo un array come parametro non viene fatta una copia dell'array (come avviene per i tipi predefiniti), ma viene fatta una copia dell'indirizzo, quindi le modifiche effettuate all'interno della funzione verranno automaticamente eseguite sull'array.

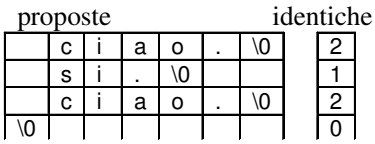
- Le stringhe sono array di caratteri il cui ultimo carattere è '\0'. Funzioni principali **strcpy** (copia stringa), **strcat** (concatena stringhe), **strchr** (trova carattere in stringa), **strcmp** (confronta stringhe).

```
// decisioni.cpp
#include <iostream>
#include <cstring>    // oppure <string.h>
using namespace std;
const int NUM_UTENTI = 3;
const int MAX_CARATT = 64;

void visualizzaProposte(const char proposte[][MAX_CARATT])
{ for (int i=0; i<NUM_UTENTI; i++)
    cout << i+1 << ">" << proposte[i] << endl;
}

void inserisciProposta(char proposte[][MAX_CARATT], int i)
{ strcpy(proposte[i],"\0");
  cout << "messaggio n." << i+1 << " (termina con '.')? " << endl;
  char* posiz = proposte[i];
  while(strchr(posiz,'.') == 0)      // finchè non trovi un punto
  {  strcat(posiz," ");              // aggiungi spazio in fondo
     posiz = strchr(posiz,'\0');     // porta posiz alla fine
     cin >> posiz;                   // leggi la prossima parola
  }
}

bool prendiDecisione(char proposte[][MAX_CARATT], char* decisione)
{ int identiche[NUM_UTENTI] = {0};
  for (int i=0; i<NUM_UTENTI && (strcmp(proposte[i],"\0")!=0); i++)
    for (int j=0; j<NUM_UTENTI && (strcmp(proposte[j],"\0")!=0); j++)
     if (strcmp(proposte[i],proposte[j])==0)
        identiche[i]++;
  int max = 0;
  char* decisioneMaggioritaria;
  for (int i=0; i<NUM_UTENTI && (identiche[i]!=0); i++)
    if (identiche[i] > max)
    {  max = identiche[i];
       decisioneMaggioritaria = proposte[i];
    }
  strcpy(decisione, decisioneMaggioritaria);
  return max > (NUM_UTENTI/2);
}...
```

- La funzione *inserisci Proposta* preleva il messaggio parola per parola.
- *posiz* è un puntatore sempre posizionato a fine stringa.



- La funzione *prendi decisione* conta i duplicati per ogni messaggio.



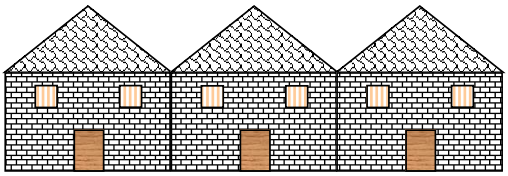

```
...
int main()
{ char proposte[NUM_UTENTI][MAX_CARATT] = {"\0"};
  char decisione[MAX_CARATT] = "\0";
  visualizzaProposte(proposte);
  int i=0;
  do
  {  inserisciProposta(proposte,i);
     system("cls");
     visualizzaProposte(proposte);
     i = (i+1) % NUM_UTENTI;
  }
  while(!prendiDecisione(proposte, decisione));
  cout << "decisione: " << decisione << endl;
  system("pause");
}
```

- Poi esegue un cliclo alla ricerca del messaggio con massimo numero di duplicati, e lo punta con *decisione Maggioritaria*
- In chiusura, copia il messaggio *decisione Maggioritaria* nella stringa passata tramite il puntatore *decisione*.

ESERCITAZIONE 9

LINGUAGGIO C++ : LE STRUTTURE, GLI ARRAY DI STRUTTURE.

1. Assassinio a Void City



- Sherlock Holmes è stato incaricato di indagare sulla misteriosa morte del pesce *Float,* ucciso da un'esplosione di bit avvenuta in una delle case a schiera nella periferia *Void City*. Il Dr. Watson si presenta a rapporto da Holmes in *Return Street*.
- Watson: "Ci sono novità, Holmes: pare che *Float* fosse uscito un'ora prima del decesso a prendere una boccata d'acqua". Holmes: "Ciò che è più strano, è che la bomba sia stata fabbricata appositamente per esplodere in un istante preciso, il che fa supporre che l'assassino conoscesse bene le abitudini del pesce, ossia ne fosse il custode".
- Watson: "Il problema è che in queste tre case a schiera, ognuna di un diverso colore, abitano tre custodi di diverse nazionalità, che prediligono bevande differenti e custodiscono tre diversi animali, a rotazione. Chi di essi custodiva il pesce?".
- Watson: "Ho pedinato le tre persone sino al bar *Char*, essendo impossibile interrogarli o entrare nelle loro case senza un indizio. Ma ciascuno di essi parla una lingua a me sconosciuta, per cui dai discorsi ho solo compreso che*: l italiano vive*

nella casa rossa; il proprietario della casa verde beve birra; chi ha la casa blu è francese; chi tiene il cane è distante da chi aveva il pesce; chi beve vino ha un gatto, chi abita nella casa blu è lontano dal tedesco; chi beve caffe è a sinistra della casa rossa; il gatto è a destra del cane".

- Holmes: "Elementare, mio caro Watson: cio che mi avete detto individua inequivocabilmente l'assassino: a me un portatile con DevC++!"
- La soluzione è la seguente:

	CASA	CASA	CASA
COLORE	blu	rosso	verde
NAZIONE	francese	italiano	tedesco
ANIMALE	cane	gatto	pesce
BEVANDA	caffe	vino	birra

- Ogni casa ha caratteristiche di diversa natura (colore, nazione, animale, bevanda). Le strutture (struct) sono il costrutto del C++ che consente di rappresentare elementi dai componenti di tipo diverso. Mentre gli array consentono di rappresentare elementi dai componenti dello stesso tipo.
- Quindi, ciascuna casa è una struttura con i membri *colore, nazione, animale, bevanda*. Poichè le case sono tutte dello stesso tipo, la sequenza di case è memorizzata come array di strutture.

- Il programma controlla tutte le possibili combinazioni, e stampa quelle che concordano con gli indizi. Tutte le combinazioni sono 3^{12} =531441. Infatti, i valori possibili possono essere rappresentati con enumerati da 0 a 2:

```
enum Colori     {ROSSO=0,     VERDE=1,     BLU=2};
enum Nazioni    {ITALIANO=0, FRANCESE=1, TEDESCO=2};
enum Animali    {CANE=0,      GATTO=1,     PESCE=2};
enum Bevande    {BIRRA=0,     VINO=1,      CAFFE=2};
```

- Ad esempio, la tabella di sopra si codifica come **`2102 0011 1220`**. Gli enumerati possono essere visti con valori alfabetici o numerici a seconda delle comodità. In sostanza ciascun tentativo sarà un numero a 12 cifre in base 3. Per provare tutti i tentativi, si usa un contatore decimale da 0 a 531440, da cui estrarre ogni volta le "terzine" (ossia le corrispondenti cifre in base 3). Questo viene svolto dalla funzione *provaAltraSoluzione*.

- La funzione *controlla* provvede a vedere se ciascun indizio è soddisfatto: tentativo per tentativo, se esiste una casa per cui un indizio è violato allora la funzione termina restituendo *false*. Altrimenti, se nessun indizio è stato violato, restituisce *true*.


```
// assassinio.cpp
#include <iostream>
#include <cmath>
using namespace std;

enum Colori     {ROSSO, VERDE, BLU};
enum Nazioni    {ITALIANO, FRANCESE, TEDESCO};
enum Animali    {CANE, GATTO, PESCE};
enum Bevande    {BIRRA, VINO, CAFFE};
const char nazioni[3][9] = { {"italiano"}, {"francese"}, {"tedesco"} };

struct Casa
{ Colori    colore;
  Nazioni   nazione;
  Animali   animale;
  Bevande   bevanda;
};
bool controlla(Casa schiera[])
{ for (int i=0; i<3; i++)
  { if ( (schiera[i].nazione==ITALIANO) && (schiera[i].colore!=ROSSO) )          return false;
    if ( (schiera[i].colore==VERDE) && (schiera[i].bevanda!=BIRRA) )            return false;
    if ( (schiera[i].animale==GATTO) && (schiera[i].bevanda!=VINO) )            return false;
    if ( (schiera[i].nazione==FRANCESE) && (schiera[i].colore!=BLU) )           return false;
    int j;
    if (schiera[i].animale==CANE)
    { for (j=0; schiera[j].animale!=PESCE && j<3; j++); if (j!=3 && abs(i-j)!=2) return false;}
    if (schiera[i].bevanda==CAFFE)
    { for (j=0; schiera[j].colore!=ROSSO && j<3; j++);  if (j!=3 && i>=j)        return false;}
    if (schiera[i].animale==GATTO)
    { for (j=0; schiera[j].animale!=CANE && j<3; j++);  if (j!=3 && i<=j)        return false;}
    if ((schiera[i].colore==schiera[(i+1)%3].colore))                           return false;
    if ((schiera[i].nazione==schiera[(i+1)%3].nazione))                         return false;
    if ((schiera[i].animale==schiera[(i+1)%3].animale))                         return false;
    if ((schiera[i].bevanda==schiera[(i+1)%3].bevanda))                         return false;
  }
  return true;
}...
```

```
...
void provaAltraSoluzione(Casa schiera[], int valore)
{ for (int i=0; i<3; i++)
  { schiera[i].colore    = Colori(valore%3);
    valore = valore/3;
    schiera[i].nazione   = Nazioni(valore%3);
    valore = valore/3;
    schiera[i].animale   = Animali(valore%3);
    valore = valore/3;
    schiera[i].bevanda   = Bevande(valore%3);
    valore = valore/3;
  }
}
int main()
{ cout << "Elementare Watson, l'assassino e' l'uomo ";
  Casa schiera[3];
  const int MAX = int(pow(3.0,12.0));
  for (int valore=0; valore<MAX; valore++)
  { provaAltraSoluzione(schiera, valore);
    if (controlla(schiera))
    { for (int i=0; i<3; i++)
       if (schiera[i].animale==PESCE)
          cout << nazioni[schiera[i].nazione] << endl;
    }
  }
  system("pause");
}
```

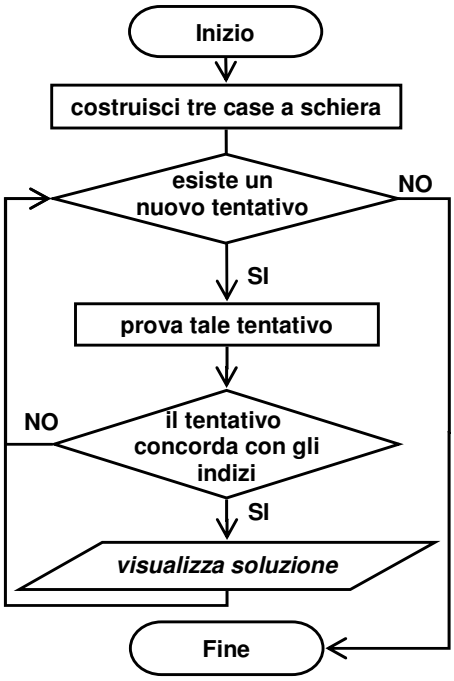


Fig.1 – Funzione principale

- Esercizio: si visualizzi tutta la soluzione, non solo la nazionalità del custode del pesce.

- Approfondimento (molto difficile): la versione originale del rebus fu formulata da Albert Einstein, il quale nel secolo scorso sostenne che il 98% della popolazione mondiale non sarebbe stato in grado di risolverlo.
  Ci sono 5 case, ognuna con un colore differente. In ogni casa abita una persona di nazionalità diversa. Ogni inquilino predilige una bevanda differente, fuma una marca diversa di sigarette e tiene un animale domestico diverso. Nessuna delle 5 persone beve la stessa bevanda, fuma le stesse sigarette o tiene un animale uguale al suo vicino. Il britannico vive nella casa rossa. Lo svedese tiene un cane. Il danese beve volentieri il tè. La casa verde si trova sulla sinistra di quella bianca. Il proprietario della casa verde beve caffè. La persona che fuma Pall Mall ha un uccello. L'uomo che abita nella casa in mezzo beve latte. Il proprietario della casa gialla fuma Dunhill. Il norvegese abita nella prima casa. L'uomo che fuma Malboro abita vicino a quello che tiene un gatto. L'uomo che tiene un cavallo abita vicino a quello che fuma Dunhill. Il fumatore di Winfield beve volentieri birra. Il norvegese abita vicino alla casa blu. Il tedesco fuma Rothmans. Il fumatore di Malboro ha un vicino che beve acqua. A chi appartiene il pesce ? (soluzione: al tedesco, che ha una casa verde, beve caffè e fuma sigarette Rothmans.)

ESERCITAZIONE 10

LINGUAGGIO C++ : PSEUDO-COMPITO (DIFFICOLTÁ BASSA).

1. Ascensore

- Al fine di smaltire le interminabili code di attesa per la visita alla Torre di Pisa, viene progettato un ascensore (Fig.1) con la seguente logica di funzionamento.
- Vi sono 7 piani (numerati da 0 a 6) e su ogni piano può entrare un solo visitatore. L'ascensore ha 3 posti. All' ingresso dell'ascensore ci sono sempre visitatori pronti ad entrare.
- Per entrare nella torre occorre passare attraverso l'ascensore, sia in ingresso che in uscita. L'ascensore è condotto da un custode attraverso un computer. Il custode ha a disposizione un monitor che visualizza lo stato di affluenza della torre, dell'ascensore, e della coda di attesa.

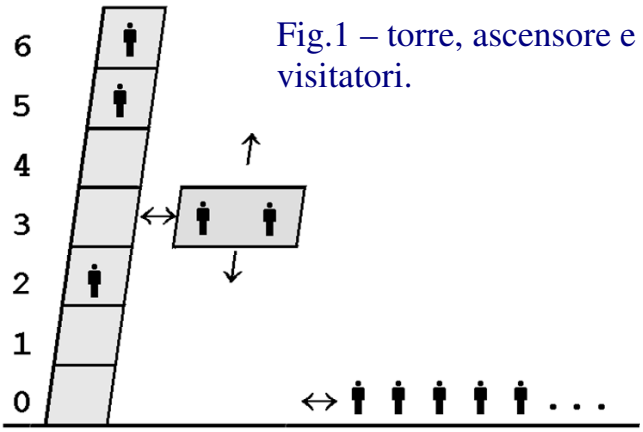


Fig.1 – torre, ascensore e visitatori.

- I comandi a disposizione del custode sono dati attraverso dei numeri (Fig.2), e corrispondono alle direzioni indicate sul tastierino numerico.
- **1 = fine** chiude il programma, es. quando si è fuori dall'orario di visita.

- **8 = sali di un piano** fa salire l'ascensore di un piano, se non è già all'ultimo piano. Altrimenti non fa nulla.
- **2 = scendi di un piano** fa scendere l'ascensore di un piano, se non è già al piano terra. Altrimenti non fa nulla.
- **7 = monta in ascensore** fa entrare in ascensore un visitatore in attesa, se l'ascensore è al piano terra e se c'è un posto libero. Altrimenti non fa nulla.
- **4 = entra nella torre** fa entrare un passeggero dell'ascensore nella torre, se la torre è vuota al piano in cui si trova l'ascensore, e se l'ascensore ha visitatori. Altrimenti non fa nulla.

```
6        |o|
5        |o|
4        | |
3        | | [o o]
2        |o|
1        | |
0        | | ooo...
_____________________________
8: sali di un piano
2: scendi di un piano
7: monta in ascensore <-
4: entra nella torre  <-
6: esci dalla torre   ->
5: disintegra tutti
3: scendi dall'ascensore ->
9: urgenza al piano n.
1: fine
```

Fig.2 – Programma di controllo.

- **6 = esci dalla torre** fa uscire un eventuale passeggero dalla torre, posto a livello dell'ascensore, e lo fa entrare nell'ascensore, se nell'ascensore vi sono posti. Altrimenti non fa nulla.
- **3 = scendi dall'ascensore** fa uscire un passeggero dall'ascensore, se l'ascensore è al piano terra e se non è vuoto. Altrimenti non fa nulla.

- **5 = disintegra tutti** fa scomparire istantaneamente tutti i visitatori nella torre e nell'ascensore.
- **9 = urgenza al piano n.** viene richiesto anche il piano in cui è stata chiesta assistenza urgente. Porta immediatamente l'ascensore al piano terra, lo libera di due posti (se non disponibili), fa salire un medico, ed infine porta l'ascensore istantaneamente al piano richiesto.
- Nella pagina seguente viene fornito un programma C++ che realizza il suddetto controllo. Le funzioni **`visualizza`** e **`main`** sono già svolte, e vanno preliminarmente analizzate per comprendere ciò che occorre aggiungere.
- Si noti che la torre è realizzata come array di 7 caratteri di nome **`piani`**, l'ascensore come un array di 3 caratteri di nome **`ascensore`**, ed il piano al quale si trova l'ascensore viene memorizzato nella variabile intera **`livello`**. Queste tre variabili sono locali al main.
- Completare le seguenti funzioni, in accordo a quanto detto ed alla traccia fornita a pag.4. La soluzione proposta viene fornita a pag.5.
- **`disintegra`**,che realizza il comando 5; **`sali`**, per il comando 8; **`scendi`**, per il comando 2; **`entraAsc`**, per il comando 7; **`esciTorre`**, per il comando 6; **`entraTorre`**, per il comando 4; **`esciAsc`**, per il comando 3; **`numPasseggeri`**, che serve a realizzare il comando 9 (già reealizzato nel main), ed in particolare restituisce un intero con il numero di passeggeri a bordo dell'ascensore.


```
// ascensore.cpp

#include <iostream>
using namespace std;

const int PIANI = 7;
const int POSTI = 3;

void visualizza(char piani[], char asc[], int liv )
{ system("cls");
  for (int i=PIANI-1; i>=0; i--)
  { cout << i << '\t' << "|" << piani[i] << "|";
    if (i==liv)
    { cout << " [";
      for (int j=0; j<POSTI; j++)
        cout << asc[j];
      cout<< "]";
    }
    if (i==0)
      cout << " ooo...";
    cout << endl;
  }
  cout << "_____________________________\n";
  cout << "8: sali di un piano \n"
       << "2: scendi di un piano \n"
       << "7: monta in ascensore <- \n"
       << "4: entra nella torre  <- \n"
       << "6: esci dalla torre   -> \n"
       << "5: disintegra tutti \n"
       << "3: scendi dall'ascensore -> \n"
       << "9: urgenza al piano n. \n"
       << "1: fine \n";
}

void disintegra(char piani[], char asc[])
{ //... }
int sali(int livello)
{ //... return 0; }
int scendi(int livello)
{ //... return 0; }
void entraAsc(char asc[], int liv)
{ //... }
void esciTorre(char piani[], char asc[], int liv)
{//...}
void entraTorre(char piani[], char asc[], int liv)
{ //... }
void esciAsc(char asc[], int liv)
{ //... }
int numPasseggeri(char asc[])
{ //... return 0; }

int main(){
  char piani[PIANI];
  char ascensore[POSTI];
  int livello = 0;
  disintegra(piani, ascensore);
  char c;
  int p;
  while(true)
  { visualizza(piani, ascensore, livello);
    cin >> c;
    switch(c)
    { case '8': livello = sali(livello);              break;
      case '2': livello = scendi(livello);            break;
      case '7': entraAsc(ascensore, livello);         break;
      case '4': entraTorre(piani, ascensore, livello); break;
      case '6': esciTorre(piani, ascensore, livello); break;
      case '5': disintegra(piani, ascensore);         break;
      case '3': esciAsc(ascensore, livello);          break;
      case '9': cin >> p;
                livello = 0;
                visualizza(piani, ascensore, livello);
                system("pause");
                while (POSTI-numPasseggeri(ascensore)<2)
                  esciAsc(ascensore, livello);
                visualizza(piani, ascensore, livello);
                system("pause");
                entraAsc(ascensore, livello);
                visualizza(piani, ascensore, livello);
                system("pause");
                livello = p;                          break;
      case '1': exit(0);                              break;
      default:  cout << char(7);
    }
  }
}
```

```
// ascensore.cpp
//...

void disintegra(char piani[], char asc[])
{ for (int i=0; i<PIANI; i++)
    piani[i] = ' ';
  for (int i=0; i<POSTI; i++)
    asc[i] = ' ';
}

int sali(int livello)
{ if (livello < (PIANI-1))
    return livello+1;
  else
    return livello;
}

int scendi(int livello)
{ if (livello > 0)
    return livello-1;
  else
    return livello;
}

void entraAsc(char asc[], int liv)
{ if (liv!=0)
    return;
  for (int i=0; i<POSTI; i++)
    if (asc[i]==' ')
    { asc[i]='o';
      return;
    }
}

int numPasseggeri(char asc[])
{ int num=0;
  for (int i=0; i<POSTI; i++)
    if (asc[i]=='o')
    num = num + 1;
  return num;
}
```

```
void esciTorre(char piani[], char asc[], int liv)
{ if (piani[liv]==' ')
    return;
  for (int i=0; i<POSTI; i++)
    if (asc[i]==' ')
    { asc[i]='o';
      piani[liv]=' ';
      return;
    }
}

void entraTorre(char piani[], char asc[], int liv)
{ if (piani[liv]!= ' ')
    return;
  for (int i=0; i<POSTI; i++)
    if (asc[i]=='o')
    { asc[i]=' ';
      piani[liv]='o';
      return;
    }
}

void esciTorre(char piani[], char asc[], int liv)
{ if (piani[liv]==' ')
    return;
  for (int i=0; i<POSTI; i++)
    if (asc[i]==' ')
    { asc[i]='o';
      piani[liv]=' ';
      return;
    }
}

void esciAsc(char asc[], int liv)
{ if (liv!=0)
    return;
  for (int i=0; i<POSTI; i++)
    if (asc[i]=='o')
    { asc[i]=' ';
      return;
    }
}
```

ESERCITAZIONE 11

LINGUAGGIO C++ : PSEUDO-COMPITO (DIFFICOLTÀ MEDIO-BASSA).

1. Top secret

- La principessa Lady *C* è amante del Conte *Dev*. Essi hanno bisogno di trasmettersi dei messaggi, come file allegati di posta elettronica o spediti in busta in un CD, in modo che il principe *Do* e la contessa *While* non possano scoprire la relazione amorosa segreta.
- Quindi scrivono il programma *topsecret*, che consente di cifrare e decifrare i messaggi contenuti nei file. Cifrare un messaggio significa modificarne i bit in modo che siano incomprensibili. Decifrare significa riportare i bit al contenuto originario, tramite un'opportuna parola segreta (password).
- Il programma consente di eseguire cinque operazioni a scelta (Fig. 1).
- **1 = scrivi un nuovo messaggio**, l'utente scrive un messaggio, che termina con un punto ('.'). Il messaggio viene salvato nel file "messaggio.txt".
- **2 = leggi il messaggio**, apre il file "messaggio.txt", controllando che tale file esista, e lo visualizza su video. Altrimenti visualizza "nessun messaggio".
- **3 = cifra il messaggio**, legge il contenuto del file "messaggio.txt" e costruisce il nuovo file "messaggioCifrato.txt" con i caratteri modificati nel seguente modo. Si prende ciascun carattere (`c`), si converte in intero (`int(c)`), si somma uno

(`int(c)+1`), e si riconverte in carattere (`char(int(c)+1)`). Questo equivale a trasformare un carattere nel carattere successivo della tabella di codifica ASCII.

- **4 = decifra il messaggio**, legge il contenuto del file "messaggioCifrato.txt" e costruisce un nuovo file chiamato "messaggio Decifrato.txt", visualizzandolo a video, in cui i caratteri sono modificati nel seguente modo. Si prende ciascun carattere (`c`), si converte in intero (`int(c)`), si sottrae uno (`int(c)-1`), e si riconverte in carattere (`char(int(c)-1)`). Questo equivale a trasformare un carattere nel carattere precedente della tabella di codifica ASCII. Come tale, il file "messaggioDecifrato.txt" è identico al file "messaggio.txt" inizialmente scritto.
- Per decifrare il messaggio occorre inserire una parola segreta, nota solo ai due amanti ("sbrodolina"). Se per tre volte di seguito si sbaglia ad inserire la parola segreta, il messaggio viene automaticamente distrutto, poichè certamente finito in altre mani.

```
1: scrivi un nuovo messaggio
2: leggi il messaggio
3: cifra il messaggio
4: decifra il messaggio
0: fine
? 1
----------------------------
Mi manchi molto, anzi
tantissimissimissimissimo.
```

```
1: scrivi un nuovo messaggio
2: leggi il messaggio
3: cifra il messaggio
4: decifra il messaggio
0: fine
? 3
----------------------------
Nj!nbodij!npmup-!bo{j
uboujttjnjttjnjttjnjttjnp/
----------------------------
```

Fig.1 – Programma *topsecret*.

- **0 = fine**, termina l'applicazione.
- Di seguito viene fornita una parte del programma *topsecret*. La funzione **`main`** è già svolta, e va preliminarmente analizzata per comprendere ciò che occorre aggiungere.
- Completare le seguenti funzioni, in accordo a quanto detto ed alla traccia fornita a pag.4. La soluzione proposta viene fornita a pag.5.
- **`scriviMessaggio`**,che realizza il comando 1, prendendo in ingresso una stringa con il nome del file da creare; **`leggiMessaggio`**, per il comando 2; **`distruggiMessaggio`**, che apre il file in scrittura, lo chiude subito dopo (così provocando la cancellazione del contenuto precedente) e visualizza "messaggio distrutto"; **`verificaParolaSegreta`**, che chiede all'utente di inserire la parola segreta, restituendo **`true`** in caso sia corretta, oppure **`false`** dopo tre tentativi ;
- **`codificaMessaggio`**, che consente di eseguire le operazioni 3 (cifra) e 4 (decifra). La funzione prende come parametro due nomi di file ed un intero. Quindi legge il primo file, ed ad ogni carattere esegue l'operazione di codifica e scrittura nel secondo file. L'operazione di codifica consiste nel convertire il carattere in tipo intero e sommare il valore del parametro intero. Se tale intero vale `1`, allora si ha la cifratura, se vale `-1` si ha la decifratura.

NOTA: per leggere il carattere `c` da un `flusso` si consiglia di usare l'operazione `flusso.get(c)` che, a differenza della funzione `flusso >> c` , non salta i caratteri spazio e accapo. Consultare i lucidi di teoria a pag. 100-102.


```
// topsecret.cpp

#include <iostream>
#include <fstream>

using namespace std;

void scriviMessaggio(const char* nomeFile)
{ //...
}

void leggiMessaggio(const char* nomeFile)
{ //...
}

void codificaMessaggio(const char* nomeFile,
                       const char* nomeFileCod,
                       int codice)
{ //...
}

bool verificaParolaSegreta()
{ //...
  return false;
}

void distruggiMessaggio(const char* nomeFile)
{ //...
}
```

```
int main()
{ char c = ' ';
  while (c!='0')
  { system("cls");
    cout << "1: scrivi un nuovo messaggio \n"
         << "2: leggi il messaggio \n"
         << "3: cifra il messaggio\n"
         << "4: decifra il messaggio\n"
         << "0: fine \n"
         << "? ";
    cin >> c;
    cout << "-----------------------------\n";
    switch(c)
    { case '1': scriviMessaggio("messaggio.txt");
                break;
      case '2': leggiMessaggio("messaggio.txt");
                break;
      case '3': codificaMessaggio("messaggio.txt",
                                  "messaggioCifrato.txt",1);
                break;
      case '4': if (verificaParolaSegreta())
                   codificaMessaggio("messaggioCifrato.txt",
                                     "messaggioDecifrato.txt",
                                     -1);
                else
                   distruggiMessaggio("messaggioCifrato.txt");
    }
    cout << "\n-----------------------------\n";
    system("pause");
  }
}
```

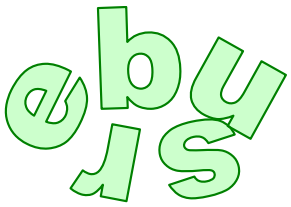
```
// topsecret.cpp
// ...
void scriviMessaggio(const char* nomeFile)
{ fstream flusso;
  char c;
  flusso.open(nomeFile, ios::out);
    cin.get(c);
    while (c!='.')
    { cin.get(c);
      flusso << c;
    }
  flusso.close();
}

void codificaMessaggio(const char* nomeFile,
                       const char* nomeFileCod,
                       int codice)
{ fstream ing, usc;
  char c;
  ing.open(nomeFile, ios::in);
  usc.open(nomeFileCod, ios::out);
  while(ing.get(c))
  { c = char( int(c) + codice );
    cout << c;
    usc << c;
  }
  ing.close();
  usc.close();
}
```

```
void distruggiMessaggio(const char* nomeFile)
{ fstream flusso;
  flusso.open(nomeFile, ios::out);
  flusso.close();
  cout << "messaggio distrutto\n";
}

void leggiMessaggio(const char* nomeFile)
{ fstream flusso;
  char c;
  flusso.open(nomeFile, ios::in);
    if (!flusso)
    { cerr << "nessun messaggio\n";
      return;
    }
    while(flusso.get(c))
      cout << c;
  flusso.close();
}

bool verificaParolaSegreta()
{ char parolaSegreta[16];
  for (int i=1; i<=3; i++)
  {  cout << "password (tentativo n." << i << ")? ";
     cin >> parolaSegreta;
     if (strcmp(parolaSegreta,"sbrodolina")==0)
       return true;
  }
  return false;
}
```

ESERCITAZIONE 12

LINGUAGGIO C++ : PSEUDO-COMPITO (DIFFICOLTÀ MEDIO-BASSA).

1. Rebus



- Un programma per giocare a rebus si compone di un dizionario contenente un insieme di termini, una funzione di estrazione che prende a caso un termine, ed altre operazioni che consentono di inserire le lettere mancanti e di contare gli errori.
- All'avvio del programma, il giocatore conosce il numero di lettere del termine estratto. Inserendo delle lettere, se queste appartengono al termine verranno inserite, altrimenti verrà incrementato il contatore degli errori (Fig.1).

```
rebus : R   A ARE (9 lettere)
errori: 5
O
```

Fig.1 – Programma rebus.

- A pagina 3 viene fornita una traccia di programma C++ che realizza il gioco. La funzione **`main`** è già svolta, e va preliminarmente analizzata per comprendere ciò che occorre aggiungere.
- Si noti che all'inizio del programma si dichiara una struttura **`rebus`**, composta dai campi **`soluzione`** (che contiene la parola da indovinare), **`tentativo`** (che contiene la parola con le lettere corrette tentate dal giocatore), **`lung`** (la lunghezza della parola da indovinare) ed **`err`** (il numeo di errori).

- Tale struttura consente di passare comodamente alle funzioni tutti i parametri mediante una sola variabile di tipo **`rebus`**.
- **ATTENZIONE**: a differenza degli array, il passaggio di una struttura avviene per copia. Quindi se si vuole passare una struttura per indirizzo, per consentire ad una funzione di modificarla, occorre farlo in modo esplicito come di seguito.
- **NOTA**: nel programma *strutt.cpp*, la funzione `assegna3` modifica solo la struttura locale `t` senza alterare la `s`. Mentre la funzione `assegna2` modifica direttamente la `s`, attraverso il puntatore `t` .
  Si noti che le notazioni `(*t).` e `t->` sono equivalenti, per cui la `assegna2` esegue due volte la medesima operazione.

```
// strutt.cpp
#include <iostream>
using namespace std;

struct miastr
{ int  c; };

void assegna2(miastr* t)
{ (*t).c = 2;
   t->c  = 2;
}

void assegna3(miastr t)
{  t.c = 3; }

int main()
{ miastr s;
  assegna2(&s);
  cout << s.c;
  assegna3(s);
  cout << s.c;
  system("pause");
}
```

  Di conseguenza, nel `main`, alla `assegna2` viene passato l'indirizzo di `s`, mentre alla `assegna3` viene passata la variabile `s`. Il programma stampa due volte il numero `2`.
- Completare le seguenti funzioni, in accordo a quanto detto ed alla traccia fornita. La soluzione viene proposta a pag.4.
- **`estraiParola`**, che sceglie un numero a caso n, tra 1 e 20, e legge dal file "dizionario.txt" la n-esima parola, copiandola nel campo `soluzione`.

`inizializzaRebus`, che inizializza il campo `lung` con la lunghezza della parola caricata, il campo `err` a zero, ed il campo `tentativo` con una stringa di lunghezza `lung` composta da caratteri spazio (`' '`). **`stampaRebus`** che stampa le informazioni come in Fig.1. **`provaLettera`** che cerca il carattere `c` nel campo `soluzione`. Ogni volta che trova tale carattere, lo copia nel campo `tentativo` al medesimo posto. Se il carattere non viene trovato, si incrementa il campo `err`.

```
// rebus.cpp

#include <iostream>
#include <fstream>

using namespace std;

struct rebus
{ char soluzione[32];
  char tentativo[32];
  int  lung;
  int  err;
};

void estraiParola(rebus* pr)
{ //... }

void inizializzaRebus(rebus* pr)
{ //... }
```

```
void provaLettera(rebus* pr, char c)
{ //... }

void stampaRebus(rebus reb)
{ //... }

int main()
{ rebus reb;
  estraiParola(&reb);
  inizializzaRebus(&reb);
  stampaRebus(reb);
  char c;
  do
  { cin >> c;
    provaLettera(&reb,c);
    stampaRebus(reb);
  }
  while (strcmp(reb.soluzione,reb.tentativo)!=0);
  system("pause");
}
```



```
// rebus.cpp
// ...
void estraiParola(rebus* pr)
{ fstream flusso;
  srand(time(NULL));
  int n = rand()%20 + 1;
  flusso.open("dizionario.txt", ios::in);
  for (int i=0; i<n; i++)
     flusso >> pr->soluzione;
  flusso.close();
}

void inizializzaRebus(rebus* pr)
{ pr->lung = strlen(pr->soluzione);
  for(int i=0; i < pr->lung; i++)
    pr->tentativo[i]=' ';
  pr->tentativo[pr->lung] = '\0';
  pr->err = 0;
}
```

```
void provaLettera(rebus* pr, char c)
{ bool trovata = false;
  for(int i=0; i < pr->lung; i++)
  if (pr->soluzione[i]==c)
  {   pr->tentativo[i]=c;
      trovata = true;
  }
  if (!trovata)
    { cout << char(7);
      (pr->err)++;
    }
}

void stampaRebus(rebus reb)
{ system("cls");
  cout << "rebus : " << reb.tentativo
       << " (" << reb.lung << " lettere)" << endl;
  cout << "errori: " << reb.err << endl;
}
```

Esempo di file “dizionario.txt” con 20 parole (da creare con Notepad)

```
BENESTANTE
COMODO
BIOMEDICO
INTREPIDO
FANTOMATICO
ASSALTARE
ASSORTIRE
BENEDIRE
SFATTO
CONIUGARE
MERCANTEGGIARE
CONTRATTARE
PIALLA
TRESPOLO
LOCULO
COMPITO
ADOLESCENTE
ROTTAMARE
SOSTENIBILE
INECCEPIBILE
```