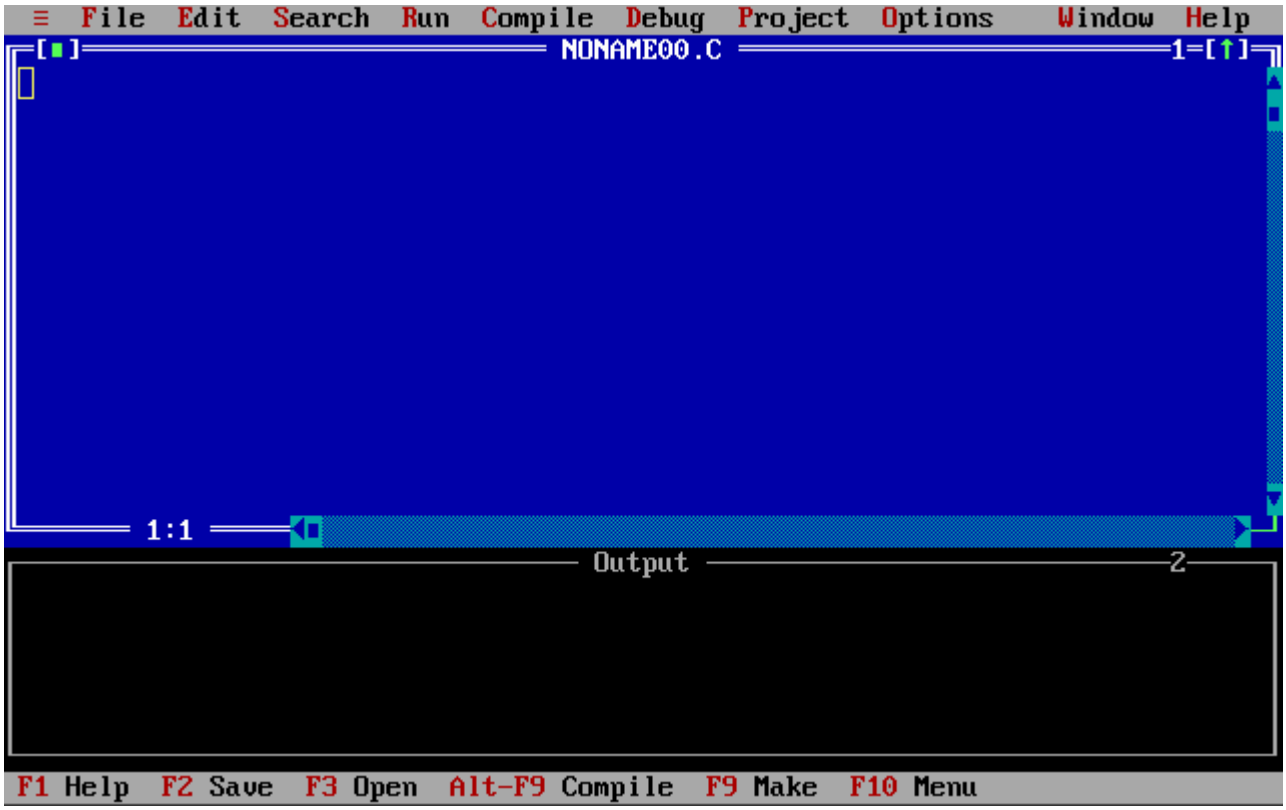
INTRODUZIONE

1. Ambiente TurboC/C++

Il TurboC/C++ è un ambiente integrato per la messa a punto di programmi in C e in C++. In questo ambiente è possibile creare, correggere, compilare, mandare in esecuzione e memorizzare su disco programmi C/C++.

1.1. L'Ambiente di sviluppo programmi

Lanciando il TurboC/C++ viene visualizzata la schermata seguente:



Tramite i menù è possibile selezionare i comandi desiderati. Questo può avvenire sia utilizzando il mouse sia posizionandosi con le frecce sul comando desiderato e premendo quindi il tasto invio. Ad un comando può inoltre essere associato un tasto `F` (`F1...F10`) oppure la combinazione `Alt-TASTO F`, oppure una sequenza costituita dal tasto `Alt` seguito da una lettera. La linea in fondo allo schermo indica alcuni comandi eseguibili direttamente con i tasti `F`. Tra questi il comando `F1` per richiedere aiuto (*help*), cioè spiegazioni sulle operazioni eseguibili nel contesto in cui si lavora ed il comando `F10` per spostare il cursore sulla barra dei menù (linea superiore dello schermo).

Se si vuole uscire definitivamente dal TurboC/C++ basta eseguire il comando `Quit` del menù `File` (`Alt-f` e poi `q`). Se invece si vuole abbandonare temporaneamente l'ambiente TurboC/C++ si usa il comando `DOS shell` del menù `File`. Per rientrare in TurboC/C++ si usa il comando DOS `exit`.

Lo schermo iniziale è diviso in più parti, corrispondenti a *finestre* denominate: `<noname00.c/cpp>` (sempre presente), `<Output>`, `<Message>`... La finestra di output è quella in cui vengono mostrati i risultati dell'esecuzione dei programmi. La finestra `noname00.c/cpp` è quella in cui viene scritto il testo del programma, e si chiama *finestra di edit*. E' possibile ingrandire e spostare queste finestre, se ne possono inserire di nuove e chiudere quelle già aperte. Per aprire finestre vedi: menù `File` comando `Open` o `New`, per spostarle, ingrandirle, chiuderle: menù `Window` comando `Zoom`, `Size/Move`, `Close`. Tutte le finestre che l'utente apre vengono messe una sotto l'altra, e resta visibile di esse solo il nome (`noname00.c/cpp` o `Output`) ed il numero di controllo. Il numero di controllo è quel numero che sta a destra del nome della finestra (ad es. `nomame00.c/cpp` ha il numero di controllo `1` mentre `Output` ha il numero di controllo `2`) . Solo l'ultima finestra aperta è completamente visibile e tutti i comandi o le funzionalità che si attivano si riferiscono ad essa. Si dice che tale finestra è *attiva*. Per rendere attiva una finestra si usa `Alt-n`, con `n` numero di controllo della finestra che si vuole attivare. Per cambiare finestra attiva, basta premere il tasto `F6`.

1.2. Come assicurarsi che le compilazioni avvengano tramite il compilatore ANSI C (e non il compilatore C++)

1. **Memorizzare il programma sorgente C in un file avente estensione `.C` (e non `.CPP`)**. Allo scopo di rendere automatico l'uso dell'estensione `.c` si può intervenire sulla finestra `Editor options` specificando il suffisso `c` come `Default extension` (e non `CPP`). Tale finestra è accessibile selezionando dal menù `Options` la voce `Editor` e nel sottomenù risultante la voce `Editor`

2. **Verificare nella finestra `C++ options` che la scelta corrispondente a `Use C++ Compiler` sia `CPP extension` e non `C++ always`**. Tale finestra è accessibile tramite selezionando dal menù `Options` la voce `Compiler` e nel sottomenù risultante la voce `C++ options`.

3. **Infine, per fare riferimento allo standard ANSI del linguaggio C, è opportuno scegliere nella finestra `Source options` le `Keywords` ANSI**. Tale finestra è accessibile tramite selezionando dal menù `Options` la voce `Compiler` e nel sottomenù risultante la voce `Source`.

2. Realizzazione di un programma

Mostriamo le operazioni principali da effettuare per redigere, compilare, eseguire e memorizzare su disco un programma C.

Apertura della finestra di editor

Si esegue il comando `Open` del menù `File` (`F3`), si inserisce nella casella `name` il nome di un file (cone estensione `.c`). In questo modo si crea un file con il nome specificato in cui verrà memorizzato il testo del programma (vedi comando `Save` più avanti). Se il file già esiste in memoria di massa e deve essere modificato, occorre specificarne il nome, ed il direttorio se esso si trova in un direttorio diverso da quello corrente. Viene quindi aperta una finestra col nome specificato, in cui compare il testo del programma contenuto nel file.

Scrittura del programma

Per i comandi dell'editor si rimanda al menù `Edit` per le operazioni di inserimento e cancellazione del testo ed al menù `Search` per la ricerca e sostituzione di parti del testo. Si ricorda che le frecce consentono di posizionare il cursore nel punto sul quale si intende lavorare. Quando si ritiene completato il testo del programma questo può essere salvato nel file associato alla finestra tramite il comando `Save` del menù `File` (`F2`). Se si vuole salvare su un nuovo file il comando da utilizzare è `Save as`, che ne richiede il nome; in questo caso il file di partenza rimane inalterato.

Compilazione del programma

Tramite il comando `Compile` del menù `Compile` (`Alt-c` e poi `c`). Se la compilazione ha successo, il compilatore segnala `Success` e premendo un tasto ci si riporta sulla finestra di edit. Se invece ci sono errori o avvertimenti, la finestra di compilazione segnala `Errors` o `Warnings` in basso a sinistra. Premendo un tasto, appare la finestra `Messages` con i messaggi di avvertimento o errore. I messaggi possono essere evidenziati con i tasti freccia. Quando un messaggio è evidenziato, premendo `F1` si ottiene una spiegazione estesa dell'errore (`ESC` per chiudere la finestra). Premendo invio, il cursore si posiziona automaticamente sulla parte di programma in cui è stato commesso l'errore, facilitando così la correzione. Una volta corretto l'errore si deve ricompilare di nuovo il programma, e così via fino a ottenere il messaggio `Success`. A questo punto conviene salvare nuovamente il programma su file.

Esecuzione del programma

Il comando `Run` del menù `Run` (`Alt-r` seguito da `r`) compila il programma e, se non vengono individuati errori, lo esegue. L'input e l'output dei dati avvengono nella finestra `Output`. Per vedere il risultato dell'esecuzione del programma occorre quindi rendere tale finestra attiva. Per interrompere il programma (ad esempio nei casi di non terminazione) si usano i tasti `Ctrl-Pausa`.

Creazione di una versione eseguibile del programma

Se il programma viene considerato corretto si può costruire una versione eseguibile del programma una versione cioè, che si possa utilizzare senza entrare in TurboC/C++ mandandola in esecuzione direttamente dal DOS. Per fare ciò, si utilizza il comando `Make` del menù `Compile` (`Alt-c` seguito da `m`). In risposta a tale comando il compilatore memorizza nel direttorio corrente (cioè sul disco rigido o sul floppy) la versione eseguibile del programma (cioè in linguaggio macchina che è direttamente comprensibile dal calcolatore). Si può verificare che nel direttorio corrente viene incluso un file col nome del programma ed estensione `exe`.

3. Individuazione degli errori

Uno degli aspetti più importanti dell'ambiente di programmazione riguarda gli strumenti per l'individuazione degli errori logici contenuti nei programmi. A questo proposito risulta molto utile fare uso degli strumenti offerti dall'ambiente di programmazione.

3.1. Esecuzione del programma istruzione per istruzione

Per ottenere una esecuzione istruzione per istruzione del programma occorre eseguire il comando `Trace into` del menù `Run`. Il programma viene compilato e viene posta la barra di esecuzione all'inizio della funzione `main()`. Ogni volta che si preme `F7` viene eseguita l'istruzione successiva.

In questa prima lezione impareremo semplicemente come visualizzare sullo schermo delle frasi decise da noi. Consideriamo il programma seguente che somma `n` interi specificati dall'utente.

```
#include <stdio.h>

int main(void)

{ int i,m,n;
  int somma = 0;
  printf("Numero dati da sommare = ");
  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
    printf("Prossimo dato = ");
    scanf("%d", &m);
    somma = somma + m;
  }
  printf("Somma = %d\n", somma);
  return 0;
}
```

Premendo più volte `F7` si esegue il programma passo passo. Quando si arriva all'istruzione sottolineata, vengono chiesti dati di input, mostrando la finestra `Output`, e, per proseguire, occorre inserirli. Nella finiestra `Output` si può anche verificare l'effetto delle istruzioni di output.

Il comando `Step over` del menù `Run` (`F8`), consente di eseguire in un unico passo l'istruzione corrente, durante l'esecuzione istruzione per istruzione. In particolare quindi, quando si esegue la chiamata ad una funzione in questa modalità, non se ne esegue il flusso completo. Ad esempio:

```
#include <stdio.h>
int fattoriale(int x)
{int i;
  int somma = 1;
  for (i = x; i >= 1; i--)
    somma = somma * i;
  return somma;
}
int main(void)
{  int numero, fatt;
  printf("Numero = ");
  scanf("%d", &numero);
  fatt = fattoriale(numero);
  printf("Il fattoriale di %d e' %d", numero, fatt);
  return 0;
}
```

Eseguendo il programma passo passo, arrivati all'istruzione sottolineata dando il comando `Step over` il sistema restituisce il valore del fattoriale senza eseguire la funzione istruzione per istruzione come invece avrebbe fatto con `Trace into`.

3.2. Ispezione delle variabili

Durante l'esecuzione passp passo è molto utile verificare il valore delle variabili del programma. A questo scopo occorre specificare le variabili di cui si vuole controllare il valore. Per fare ciò si apre il menù `Debug` (`Alt-d`;) e si esegue il comando `Watches`. Così facendo si attiva un sottomenù nel quale si deve scegliere `Add watch`. `Add watch` chiede le `Watch Expression`, cioè il nome delle variabili di cui si vuole controllare il flusso (si può attivare direttamente l'opzione `Add watch` premendo `Ctrl-F7`). Facendo riferimento al programma precedente che somma n interi, può essere utile controllare il valore che assumeranno istruzione dopo istruzione le variabili `somma` ed `n`: queste sono le `Watch Expression` che vanno inserite. Per vedere il valore di tali variabili occorre attivare la finestra `Watch`, tramite il menù `Windows` (`Alt-w`) e si esegue il comando `Watch`. In tale finestra compaiono i valori che assumono le variabili specificate in precedenza. Se il programma ancora non è stato mandato in esecuzione nella finestra `Watch` accanto al nome della generica variabile ci sarà: `Unknown symbol`. Non appena si manda in esecuzione passo passo il programma le variabili assumono i valori determinati dall'esecuzione del programma. Si noti che quando ci sono due variabili con lo stesso nome in due blocchi diversi, viene mostrato il valore della variabile visibile nella parte di programma in esecuzione.

Durante l'esecuzione passo passo, si può anche effettuare un controllo più fine sulle variabili mediante il comando `Inspect...` dal menù `Debug`. Questa modalità di ispezione, che funziona solo quando il programma è in esecuzione, permette di visualizzare, oltre al *valore* di una variabile, anche il *tipo* dichiarato, e l'*indirizzo* a partire dal quale essa è memorizzata. Questo è molto utile quando si hanno assegnazioni inaspettate, dovute ad indirizzi fuori controllo. Si consideri il seguente programma, che contiene un errore molto comune in C:

```
#include <stdio.h>

int main(void)
{
  int a = 5;
  int b[3];
  int i;

  for (i = 0; i <= 3; i++)
     b[i] = 1;
  printf("valore di a = %d; valori di b = [%d, %d, %d]",
       a,    b[0], b[1], b[2]);
  return 0;
}
```

il programma stampa: `valore di a = 1; valori di b = [1, 1, 1]`
e quindi ha cambiato anche il valore di `a`; perché?

La risposta si può trovare usando l'esecuzione passo passo, e, dopo aver passato le dichiarazioni, facendo `Inspect...` sia di `a`, che di `b`. Osservate che ispezionare un vettore significa ispezionare tutte le sue componenti. L'indirizzo di `a` è visibile in alto a sinistra nella finestra di `Inspect`. Nell'installazione in Laboratorio, esso è DS:FFF4; (DS sta per Data Segment; si veda lo Help per maggiori dettagli) quello di `b[0]`, è DS:FFEE (entrambi in notazione esadecimale). Poiché un intero in TurboC occupa due byte, `b[1]` ha indirizzo DS:FFF0, `b[2]` DS:FFF2. Quando `i` giunge a 3, `b[3]` ha indirizzo DS:FFF4, che è lo stesso di `a`, e quindi, assegnando 1 a `b[3]`, in realtà assegna 1 ad `a`.

3.2.1 Finestre durante l'esecuzione passo passo con ispezione

In questa fase è utile posizionare le finestre in modo da avere sempre sotto gli occhi la finestra `Watch`, e quella del programma da controllare. Per ridimensionare una finestra si usa il comando `Tile` del menù `Windows`. In alternativa si può usare il comando `Size/Move` del menù `Windows`: muovendo il mouse o usando le frecce la finestra attiva si sposta e si può organizzare lo schermo a proprio piacimento. Le finestre di `Inspect` sono invece visibili durante l'esecuzione passo passo, e vengono automaticamete chiuse alla fine dell'esecuzione.

3.3. Punti di arresto

L'esecuzione istruzione per istruzione risulta impraticabile non appena le dimensioni dei programmi e dei dati diventano significative. Si può per questo eseguire il comando `Go to cursor` del menù `Run`, che provoca l'esecuzione di tutte le istruzioni da quella corrente fino al punto in cui è posizionato il cursore. Nella finestra `Watch` compaiono i valori delle variabili aggiornate fino all'ultima istruzione eseguita.

Si può infine richiedere l'arresto dell'esecuzione del programma solo in determinati punti usando i comandi `Toggle breakpoint` e `Breakpoint` del menù `Debug`.

`Toggle Breakpoint` attiva e disattiva punti di arresto incondizionato, mentre `Breakpoint` permette di definire e controllare i punti di arresto. All'esecuzione di tale comando si apre una finestra in cui c'è l'elenco dei punti di arresto, la condizione ed il passaggio a cui tale punto di arresto si attiva. Se sono presenti dei punti di arresto incondizionati, cioè inseriti col comando `Toggle breakpoint` non c'è né la condizione che li determina nè il passaggio in cui vengono attivati. Se si vuole inserire un punto di arresto, si deve scegliere l'opzione `Edit` con `Alt-e` : si apre così una finestra di dialogo in cui si possono inserire le seguenti informazioni:

`Condition` - condizione
`Pass Count` - numero di passaggio prima dell'attivazione
`File Name` - nome del file
`Line Number` - numero della riga del punto d'arresto

3.4. La pila delle attivazioni

La pila delle attivazioni (chiamate) di funzione può essere molto utile per esaminare l'esecuzione del programma. Per vedere la pila delle chiamate di funzione si deve esegue il comando `Call stack` nel menù `Debug`, durante l'esecuzione passo passo. Questo comando apre una finestra in cui è mostrata la pila delle attivazioni nello stato corrente del programma.

Si consideri il seguente programma che contiene una procedura ricorsiva per il calcolo del fattoriale.

```
#include<stdio.h>

void fattoriale(int* totale, int numero)
{
  if (numero > 1) {
    *totale = numero * (*totale);
    fattoriale(totale, numero-1);
  }
}
```

```
int main(void)
{
  int fatt = 1;
  int n;
  printf("Inserisci un numero:  ");
  scanf("%d", &n);
  fattoriale(&fatt, n);
  printf("Il fattoriale e' %d\n", fatt);
  return 0;
}
```

Supponendo che al programma venga fornito il valore `4`, eseguendo il comando `Call stack` subito dopo la quarta chiamanta alla funzione `fattoriale()` si ottiene:

```
fattoriale(1,DS:FFF4)
fattoriale(2,DS:FFF4)
fattoriale(3,DS:FFF4)
fattoriale(4,DS:FFF4)
main()
```

Si noti che l'ordine dei parametri è invertito rispetto a come essi sono dichiarati nell'intestazione della funzione. Si noti inoltre che tutte attivazioni della funzione `fattoriale()` hanno il parametro `totale` pari a `DS:FFF4` perchè tutte le chiamate fanno riferimento alla stessa locazione di memoria (la locazione memorizzata nel parametro `totale`, che è un puntatore).

4. I comandi dei menù

4.1. Menù File

- `New`
  Apre una nuova finestra di edit con il nome fittizio `nomamexx.c/cpp` (`xx` è un numero da `00` a `99`) ed automaticamente rende la finestra attiva.

- `Open`
  Questo comando introduce ad una "*finestra di dialogo*". In essa troviamo una casella, `Name`, in cui va inserito il nome del file da caricare. Premendo il tasto freccia in giù compare una finestra in cui ci sono tutti i file che sono stati aperti dall'inizio della sessione di lavoro, e che quindi possono essere facilmente richiamati. Sotto la casella `Name` c'è la finestra `Files`. Essa contiene il nome dei file nel direttorio corrente e che quindi è possibile caricare. Per entrare nella finestra si deve premere `Alt-f`, e poi spostarsi sul file che interessa attraverso le frecce. Premendo invio si apre il file selezionato. Le operazioni possibili in questa "*finestra di dialogo"* sono: `Open` apre una nuova finestra di edit e mette il file selezionato in questa finestra. `Replace` invece di aprire una nuova finestra sostituisce il file nella finestra attiva con il file selezionato.

- `Save`
  Salva il file che è nella finestra attiva su disco. Se il file ha il nome fittizio `nonamexx.c/cpp` allora viene automaticamente aperta la finestra (`Save as`).

- `Save as`
  Non appena viene aperto questo sottomenù, il cursore si trova nella casella `Save File As` in cui si deve inserire il nome con il quale si vuole registrare il programma. Al di sotto di questa casella c'è una finestra `Files` in cui appare la lista dei file presenti nel direttorio corrente. Ancora al di sotto c'è un'ultima finestra (blu) nella quale vengono date informazioni sui file già esistenti nel direttorio corrente (ad es. nome, data di esecuzione, memoria occupata) e sulla maschera di

visualizzazione (se ad esempio appare la maschera `A:\*.CPP`, si visualizzeranno solo i file che terminano con `CPP`).

- `Save all`
  Salva tutti i file nelle finestre aperte.

- `Change dir`
  Il comando `Change dir` conduce ad un sottomenù nel quale è possibile cambiare il direttorio corrente.

- `Print`
  Stampa il contenuto della finestra attiva.

- `Dos Shell`
  Questo programma permette di abbandonare temporaneamente il TurboC/C++ ed eseguire comandi MS-DOS e programmi. Digitando `Exit` si rientra nell'ambiente TurboC/C++.

- `Quit`
  Serve ad uscire dal TurboC/C++.

4.2.Menù Edit

- `Undo`
  Serve una volta modificata una linea a tornare allo stato precedente. Ad esempio, se si modifica la linea

  ```
  int a,b,c;
  ```

  introducendo nuove variabili

  ```
  int a,b,c,d; char y;
  ```

  usando `Undo` torno ad avere la linea iniziale.

- `Redo`
  Serve per annullare il comando `Undo`.

- `Show clipboard`
  Apre la finestra `Clipboard`, contiene ogni testo che si porta in essa attraverso i comandi `Cut` e `Copy`. I testi vengono inseriti uno di seguito al' altro. Il testo che compare evidenziato è quello che viene usato dal TurboC/C++ attraverso il comando `Paste`.

- `Cut`
  Si può usare solo dopo avere selezionato un testo. Il testo si seleziona in questo modo: si posiziona il cursore all'inizio del testo da selezionare, si preme `Ctrl-k` poi `b`, si posiziona il cursore alla fine del testo e si preme `Crtl-k` poi `k`. Un modo molto più comodo è selezionare il testo premendo la freccia verso l'alto (`Shift`) e contemporaneamente spostare il cursore con le frecce sul testo da selezionare. Il comando `Cut` (taglia) cancella il testo selezionato nella finestra attiva e lo colloca nella `Clipboard`.

- Copy
  E' simile al Cut: come Cut fa una copia del testo selezionato sulla Clipboard tuttavia a differenza del Cut tale comando lascia intatto il testo selezionato. E' possibile copiare testi dalle finestre Help: si seleziona il testo nei modi precedentemente descritti e poi premendo Ctrl-Ins si copia il testo nella Clipboard.

- Paste
  Trasferisce il testo selezionato nella Clipboard sulla finestra di edit attiva nella posizione indicata dal cursore. Si può ripetere questa operazione tutte le volte che si desidera.

- Clear
  Cancella il testo selezionato; è una funzione irreversibile.

- Copy example
  I quadri di Help contengono degli esempi di particolari funzioni che è possibile copiare nella Clipboard e quindi trasferire nelle finestre di Edit attraverso il comando Paste.

4.3. Menù Search

- Find
  Nella riga Text to Find va inserita la stringa da ricercare. Per iniziare la ricerca va scelto OK altrimenti va scelto Cancel per tornare alla finestra attiva. Se si vuole inserire una stringa che si è già ricercato si preme la freccia verso il basso e viene mostrata l'History list, cioè la lista di tutte le ricerche fatte da inizio sessione di lavoro. Attraverso le frecce è possibile selezionare la stringa desiderata, premendo invio si avvia la ricerca. Se non si vuole ricercare una stringa già cercata basta digitarla e premere invio. Va notato che inizialmente nella riga del Text to Find compare la parola su cui si trova il cursore nella finestra di Edit attiva. In Text to find ho varie opzioni:
  - Options
    - Case sensitive
      Selezionandola il TurboC/C++ fa differenza tra maiuscole e minuscole.
    - Whole word only
      Cerca solo le parole intere (e non sotto-stringhe).
    - Regular expression
      Riconosce i caratteri []*.$^.
  - Scope
    - Global
      Ci si riferisce al testo globale.
    - Selected text
      Ci si riferisce al testo selezionato.
  - Direction
    - Forward
      La direzione della ricerca va dall'origine alla fine.
    - Backward
      La direzione della ricerca è opposta.
  - Origin
    - From cursor
      La ricerca inizia da dove è il cursore fino alla fine.
    - Entire scope
      La ricerca copre l'intero testo.

- `Replace`
  Mostra una finestra di dialogo che consente di introdurre un testo da ricercare ed il testo da sostituirgli. Ha tutte le opzioni del comando text to find ,in più c'è un'opzione `Prompt on replace` la quale se attivata fa sì che TurboC/C++. prima di operare la sostituzione chieda un'ulteriore conferma. Si può inoltre sfruttare la funzionalità `Change all` che consente di cambiare tutte le corrispondenze trovate (e non solo la prima).

- `Search again`
  Ripete l'ultimo comando `Find` o `Replace` eseguito.

- `Go to line number`
  Sposta il cursore sulla linea di cui si è indicato il numero.

- `Previous error`
  Sposta il cursore sulla linea in cui si è commesso l'errore precedente.

- `Next error`
  Sposta il cursore sulla linea in cui si è commesso l'errore successivo.

- `Locate function`
  Attivo soltanto durante una fase di debugging, apre una finestra di dialogo in cui si può inserire il nome della funzione da ricercare.

4.4. Menù Run

- `Run`
  Esegue il programma fino al successivo punto d'arresto, o fino alla fine se non ce ne sono.

- `Program reset`
  Ferma l'attuale sessione di debug, rilascia la memoria allocata, chiude ogni file aperto che il programma stava usando.

- `Go to cursor`
  Serve per far eseguire al TurboC/C++ tutte le istruzioni dal inizio della funzione `main()` fino al punto in cui è posizionato il cursore.

- `Trace into`
  Esegue il programma una riga per volta.

- `Step over`
  Consente di eseguire in un unico passo la chiamata ad una procedura, o funzione, senza cioè seguire il flusso completo della procedura.

- `Arguments`
  Permette di passare argomenti al programma da eseguire come se fossero scritti nella riga di comando DOS.
-

4.5. Menù Compile

- `Compile`
  Compila il programma nella finestra di `Edit` attiva.

- `Make`
  Compila il programma nella finestra di `Edit` attiva, generando un file eseguibile, con il nome della finestra e l'estensione `exe`. Tale file viene memorizzato nel direttorio corrente.

- `Link`
  Non trattato (serve se definiamo un progetto - programma C/C++ diviso su più file).

- `Build all`
  Non trattato (serve se definiamo un progetto - programma C/C++ diviso su più file).

4.6. Menù Debug

- `Inspect`
  Permette di visualizzare l'indirizzo, il tipo, il valore in decimale ed in esadecimale di una variabile. Se la variabile è strutturata, si visualizzano tutte le componenti. Funziona solo nel mentre il programma è in esecuzione, quindi si può usare solo nell'esecuzione passo passo o con punti d'arresto.

- `Evaluate/modify`
  Calcola il valore di una variabile, o di un'espressione, lo visualizza e permette all' utente di cambiarlo.

- `Call stack`
  Apre una finestra che mostra la pila di chiamate a funzioni con i relativi parametri dalla più recente alla più vecchia. In fondo c'è quindi la funzione `main()`. Ovviamente la finestra è visibile solo se si sono predisposti punti di arresto, o si sta eseguendo il programma passo passo.

- `Watches`
  Apre un menù per la gestione dei punti di osservazione, cioè delle variabili di cui si vuole controllare l'evoluzione.
  - `Add watches`
    Inserisce un punto di osservazione; attivando questo comando compare una finestra di dialogo in cui bisogna inserire l' identificatore del punto di osservazione e premere invio. Inizialmente nella finestra compare la parola su cui è posizionato il cursore; è anche disponibile un elenco delle watches inserite dall'inizio della sessione di lavoro (si attiva premendo la freccia verso il basso).
  - `Delete watch`
    Cancella un punto di osservazione.
  - `Edit watch`
    Serve per modificare l'espressione attiva nella finestra Watch. Selezionando questo comando compare una finestra di dialogo in cui c'è l'espressione che si vuole modificare.
  - `Remove all watches`
    Cancella tutte le espressioni di controllo.

- `Toggle breakpoint`
  Attiva e disattiva punti di arresto incondizionato.

- `Breakpoint`
  Permette di gestire i punti di arresto.

4.7. Menù Project

Non trattato (serve per definire un progetto - programma C/C++ diviso su più file).

4.8. Menù Options

Non trattato. Qui ricordiamo solo che per lavorare con il compilatore ANSI C si devono selezionare le seguenti opzioni:

- Per rendere automatico l'uso dell'estensione `.c` per i file contenente i programmi sorgenti, (estensione necessaria per compilare con il compilatore C), specificare nella finestra `Editor options` il suffisso `c` come `Default extension` (e non `CPP`). Tale finestra è accessibile selezionando dal menù `Options` la voce `Editor` e nel sottomenù risultante la voce `Editor`

- Per evitare che venga chiamanto indipendentemente dall'estensione il compilatore C++, selezionare nella finestra `C++ options` alla voce `Use C++ Compiler` l'opzione `CPP extension` e non `C++ always`. Tale finestra è accessibile tramite selezionando dal menù `Options` la voce `Compiler` e nel sottomenù risultante la voce `C++ options`.

- Per fare riferimento allo standard ANSI del linguaggio C, è opportuno selezionare nella finestra `Source options` alla voce `Keywords` l'opzione `ANSI`. Tale finestra è accessibile tramite selezionando dal menù `Options` la voce `Compiler` e nel sottomenù risultante la voce `Source`.

4.9. Menù Window

- `Size/Move`
  Usando le frecce si può cambiare la posizione della finestra attiva (la cui cornice diventa azzurra). Scelta la posizione si preme invio.

- `Zoom`
  Consente di ingrandire al massimo la finestra attiva. Se essa è già stata zoomata tale comando la riporta allo stato precedente.

- `Cascade` E' il modulo attivo quando si entra in TurboC/C++: le finestre sono accatastate ed è visibile solo la finestra attiva, delle altre si vede solo il nome e il numero.

- `Tile`
  Serve a vedere contemporaneamente tutte le finestre di editing aperte. Tutte le finestre sono grandi uguali e messe una accanto all'altra senza sovrapposizioni.

- `Next`
  Rende attiva la finestra successiva.

- `Close`
  Chiude la finestra attiva.

- `Close all`
  Chiude tutte la finestre.

- `Message`
  Apre la finestra Message e la rende attiva.

- `Output`
  Apre la finestra Output e la rende attiva.

- `Watch`
  Apre la finestra Watch e la rende attiva.

- `User screen`
  Consente di vedere l'output di un programma a schermo pieno (tasti `Alt-F5`).

- `Register`
  Apre la finestra Register che mostra i registri della CPU.

- `List all`
  Serve per ottenere un elenco di tutte le finestre aperte.

4.10. Menù Help

- `Index`
  Visualizza l'elenco di tutte le parole chiave, le librerie, gli operatori, le direttive, etc., del TurboC. Molto utile come manuale di riferimento in linea, per richiamare la sintassi delle istruzioni, le precedenze tra operatori, e approfondire aspetti particolari del TurboC, (ad es., l'intervallo dei valori di `int`). Il cursore si posiziona sulle parole le cui lettere sono premute da tastiera (ad es., per trovare il while, basta immettere "wh").

- `Altri comandi`
  Non trattati.

5. Ambiente TurboC/C++: Domande poste frequentemente (FAQ)

Indice FAQ

5.1. Generalità

- **D:** *Come entro nell'ambiente TurboC/C++?*
  R: Cliccando sull'icona presente sullo schermo. In alternativa si può lanciare il comando `tc` da una finestra DOS. Se tale comando non viene riconosciuto, ci si deve prima portare nel sotto-direttorio `bin` del direttorio di istallazione del TurboC/C++.
- **D:** *Come esco dall'ambiente TurboC/C++?*
  R: Usando il menu a tendina `File`, e scegliendo `Quit`.
- **D:** *Come esco temporaneamente al DOS?*
  R: Usando il menu a tendina `File`, e scegliendo `DOS`.
- **D:** *Sono in DOS. Come ritorno all'ambiente TurboC/C++?*
  R: Con il comando DOS `exit`.
- **D:** *Come faccio a visualizzare la finestra del TurboC/C++ a schermo intero?*
  R: Con il comando `Alt-invio`. Per farla ritornare di dimensioni normali premere `Alt-invio` nuovamente.
- **D:** *Come faccio a visualizzare la finestra del TurboC/C++ con 43/50 linee invece di 25?*
  R: Selezionando dal menù `Options` la voce `Enviroment` e nel menù risultante la voce `Preference`, e ponendo l'opzione `Screen Size` a `43/50 lines`.

5.2. Interagire con l'editor

- **D:** *Come attivo i menu a tendina?*
  R: Premendo il tasto `Alt` e l'iniziale del menù richiesto (ad es. `f` per `File`, `w` per `Windows`, etc.). Nella tendina sono attivi i tasti cursore.
- **D:** *Non trovo sulla tastiera un carattere (ad esempio {). Come lo posso digitare?*
  R: E' possibile comporre un carattere eseguendo in sequenza i seguenti passi:
  - verificare che l'indicatore "Num Lock" sia acceso, se non lo fosse, accenderlo premendo il tasto "Num Lock"
  - premere il tasto `Alt`, o il tasto `Alt Gr`
  - (senza lasciare il tasto `Alt`/`Alt Gr`) comporre il codice ASCII del carattere con il tastierino numerico sulla destra
  - rilasciare il tasto `Alt`/`Alt Gr`.

  I codici ASCII più usati sono i seguenti:

  - 35 per `#`
  - 91 per `[`
  - 93 per `]`
  - 123 per `{`
  - 125 per `}`
  - 126 per `~`
- **D:** *Come passo da una finestra all'altra?*
  R: Il menu `Windows` (`Alt-w`) fornisce tutti i comandi per le finestre. I più usati sono:
  - `Alt-F3` per chiudere una finestra
  - `F5` per lo "zoom" (allargare se è piccola, rimpicciolire se è grande)
  - `F6` per passare dall'una all'altra

5.3. Caricare e salvare file

- **D:** *Come carico un file?*
  R: Tramite il menu a tendina `File` (in alternativa, tasto `F3`).
- **D:** *Non vedo il file `nomefile.txt`. Come mai?*
  R: Per default il TurboC/C++ assume che i file abbiano estensione `.c/.cpp` (ciò si può verificare notando il contenuto della finestra dove si inserisce il nome del file). Per scegliere file con estensione diversa, va modificata tale estensione (si ricorda che l'estensione `.*` è valida per tutti i file).
- **D:** *Dove devo salvare i miei file?*
  R: Conviene crearsi un direttorio personale, dove salvare tutti i propri file. Ad esempio, si può creare il direttorio `C:\MIO_DIR`, tramite il comando DOS `mkdir`.
  Conviene inoltre fissare come direttorio corrente del TurboC/C++ il direttorio così creato, scegliendo `Current directory` dal menu a tendina `File`.

5.4. Compilare ed eseguire un programma

- **D:** *Come si compilano e si eseguono i programmi?*
  R: Si possono usare i menu a tendina. In alternativa, si possono usare i tasti `Alt-F9` per la compilazione e `Ctrl-F9` per l'esecuzione.
- **D:** *Come posso vedere i risultati dell'elaborazione del mio programma?*
  R: La maniera più semplice è aprire una finestra per l'output, scegliendo `Output` nel menu a tendina `Windows`. A questo punto posso passare dalla finestra del programma a quella dove viene visualizzato l'output, e viceversa.
- **D:** *Esiste un modo per "pulire" la finestra di output?*
  R: Occorre includere nel programma C/C++ la direttiva `#include<conio.h>`. La funzione per pulire lo schermo è `clrscr()`.
- **D:** *Ho fatto eseguire un programma ed il computer si è bloccato. Cosa devo fare?*
  R: Probabilmente si è acceduta un'area di memoria riservata (ad esempio attraverso un puntatore), oppure il programma è entrato in un ciclo infinito. Si possono provare i seguenti metodi (nell'ordine):
  - Premere `Ctrl-Pausa` o `Ctrl-z` per abortire il programma.
  - Premere `Ctrl-Alt-Canc` per fare ripartire il calcolatore.
  - Spegnere l'interruttore del calcolatore, attendere 10 secondi, e riaccenderlo.

LEZIONE I

Un programma creato in C per funzionare deve essere composto in vari campi ovvero:

1. Librerie
2. Dichiarazione delle variabili
3. Il programma principale

Le librerie sono delle piccole aggiunte che permettono di utilizzare alcune funzioni senza doverle ricreare all'interno del nostro programma.

Le variabili serviranno nello svolgimento del programma quando dobbiamo acquisite per esempio un dato da tastiera.

Il programma principale invece è tutto quello che noi vogliamo creare.

LIBRERIE

Le librerie più comuni sono 3:

1. **#include<stdio.h>**

   che serve per permettere al programma di stampare a video le scritte a di acquisire dati da tastiera.

2. **#include <windows.h>**

   che permette di mantenere, tramite un comando che vedremo più avanti, le stampe a video.

3. **#include<conio.h>**

   che serve per utilizzare la maggior parte dei comandi.

VARIABILI

Possono essere di vario genere, servono per dichiarare se i dati sono numeri oppure caratteri.

Servono al compilatore per associare i dati ottenuti da tastiera ad un qualcosa che possa convertire e riutilizzare più di una volta

I tipi di dati sono:

Parola	Tipo	Memoria
char	Carattere	1 byte
int	Intero	2 byte
float	Virgola mobile	4 byte
double	Doppia	8 byte
void	Indefinito, vuoto	0 byte

PROGRAMMA PRINCIPALE

Ogni programma in C e in C++ deve iniziare con una serie di comandi standard cioè:

main()

{

}

dove all'interno delle parentesi verrà inserito il programma che vogliamo trattare.

STAMPARE FRASI A VIDEO

Per stampare il C si poggia su una funzione che è:

printf ("….");

oppure

cout<<"……"<< variabile

dove al posto dei puntini va inserita la frase che vogliamo stampare.

IL PRIMO PROGRAMMA

Creiamo adesso il primo programma e faremo scrivere al compilatore una frase semplicissimo come ad esempio: Ciao mondo!

In questo caso non sono necessarie variabili ma dovremo aggiungere una funzione che serve per mantenere stampati a video i messaggio cioè: **getch()** oppure **while(!kbhit());**

```
#include<stdio.h>
#include<conio.h>
main()
{ printf ("Ciao mondo!");
 getch();
}
```

LEZIONE 2

Questa lezione ci permetterà di far acquisire al nostro programma alcuni dati, per esempio numeri o lettere, immessi da tastiera cioè dati che scegliamo noi.

Questa operazione è estremamente semplice in quanto basta imparare ad usare una sola funzione che è lo scanf ("…");

all'interno delle parentesi verrà posizionato il parametro in base alle nostre scelte;

I parametri possono essere:

1. **%d** se il dato è un numero intero reale
2. **%f** se il dato è un numero a virgola mobile
3. **%c** se il dato è un singolo carattere
4. **%s** se il dato è una stringa di caratteri

Dopo aver scelto il tipo di dato da inserire bisogna associarlo ad una variabile (vedi lezione 1) preceduta dal segno and **(&).**

Esempio di acquisizione di un numero:

scanf ("%d", & numero);

dove: %d indica il tipo di dato che vogliamo inserire;

& comanda al programma di associare il numero alla variabile;

numero è la variabile che può avere qualsiasi nome basta che la dichiariamo all'inizio.

(Il nome però deve essere differente dalle parole chiave del programma es. case, while, default…)

Potremmo anche utilizzare la funzione

cin>>variabile

la quale risulta essere ancora più avanzata della funzione **scanf.**

Vediamo ora un programma che utilizza sia la funzione di printf che scanf e ristampa il numero inserito:

```
#include<stdio.h>

#include<conio.h>

int a;

main()

{ printf ("Quanto fa 2+2?");
```

```
  scanf ("%d", & a);

  printf ("Il risultato inserito e':%d",a);

getch();

}
```

[N.B. Il programma sopra creato non fa il controllo sul dato inserito quindi la risposta che date è arbitraria nel senso che questo programma serve solo per far capire la funzione di acquisizioni dati.]

Per fare riscrivere al programma il dato che abbiamo inserito noi da tastiera è stato sufficiente aggiungere dopo le virgolette di chiusura la variabile, in questo caso "a", che racchiude le informazioni necessarie.

Il "%d" è stato inserito prima delle virgolette di chiusura perché il programma deve stampare a video il numero inserito.

Tutto quello racchiuso nelle virgolette viene stampato mentre quello fuori viene ignorato.

Se noi volessimo scrivere qualcosa su più di una riga dovemmo semplicemente scrivere uno **\n** (new line) nel punto in cui desideriamo che il programma vada a capo o invece, se volessimo allontanare due parole, dovremmo inserire uno **\t** (tab).

Esempio:

```
#include<stdio.h>
#include<conio.h>
int a;
main()
{printf ("1-Somma\n");
  printf ("2-Sottrazione\n");
  printf ("Quale operazione vuoi effettuare?");
  scanf("%d", & a);
  printf ("L'operazione scelta e':\t%d", a);
 getch();
}
```

LEZIONE 3

Eccoci giunti alla terza lezione di questo corso che speriamo sia produttivo per voi che lo state seguendo.

Come promesso nella lezione seconda ora vi parleremo di come effettuare un controllo sui dati inseriti attraverso un ciclo che prende il nome di "*do…while*" data la sua struttura.

```
Do{…

   }while(…);
```

Esempi

```
i = 1; n = 1;

do {

   n *= i;

   i++;

  } while (i <= factorial);
```

oppure

```
while (*p == ' ') p++;
```

Questo ciclo permette di svolgere un'operazione se non se ne verifica un'altra…spieghiamoci meglio…

Il ciclo si apre con un "**do**" che tradotto dall'inglese all'italiano vuol dire "fare" e si chiude con un "**while**" che possiamo tradurre in "altrimenti".

Quindi il programma svolgerà le operazioni presenti nel "**do**" ma se esse vanno contro a determinate condizioni poste il programma scarterà tutto questo blocco svolgendo solo quelle del "**while**".

Un esempio potrebbe essere su un controllo su un numero inserito da tastiera;

```
#include<conio.h>
#include<stdio.h>
int numero;
main()
{do{printf ("Inserisci un numero compreso tra 3 e 10:\t");
      scanf ("%d", & numero);
     }while (numero<3 || numero>10);
  getch();
}
```

Cosa fa questo programma?

Come già note all'inizio ci sono le librerie e poi le variabili (vedi lezione 1) e di seguito il **printf** e lo **scanf** (vedi lezione 1 e 2).

La cosa nuova rispetto a prima è quel ciclo "**do…while**".

Il programma quando giunge al comando "do" svolge tutte le operazioni inserite al suo interno ma confronta i dati dello scanf con le condizioni inserite nel "while", in questo caso le condizioni sono che il numero inserito deve essere maggiore di 3 ma nello stesso tempo minore di 10.

A questo punto il programma confronta i dati e se rileva che il numero inserito è compreso in quel determinato intervallo.

Se invece il numero fosse stato ad esempio 11 il programma avrebbe chiuso il ciclo e avrebbe ricominciato tutte le operazioni da capo ignorando così i dati inseriti precedentemente.

Solitamente all'interno di questo ciclo c'è una funzione che fino a questo punto non abbiamo ancora trattato; si tratta della funzione cancella schermo che si indica con l'espressione: **clrscr();**

solitamente questa espressione viene posta subito dopo l'apertura della parentesi graffa del "do".

Inserendo questa funzione il programma, quando trova una discordanza tra numero inserito e condizioni, torna all'inizio del do e cancella tutto quello presente nello schermo e ristampa da capo tutto il ciclo rendendo così più funzionale il programma.

Adesso utilizziamo lo stesso ciclo non per un controllo ma per far ripetere una seconda volta l'intero programma.

Vediamo il tutto prima con un esempio: (programma 4)

```
#include <conio.h>
#include<stdio.h>
int scelta;
main()
{do{ printf ("questa e' solo una prova\n");
     printf ("vuoi ripetere il programma? s/n");
     scanf ("%c", & scelta);
}while (scelta == 's');
getch();
}
```

Come avrete notato in questo programma incontriamo ben due novità.

La prima è il %c che sostituisce il classico %d in quanto il dato che dobbiamo inserire non è più un numero ma un carattere (vedi lezione 1).

La seconda è nel "while" e a dir la verità non è proprio una novità in quanto l'abbiamo già incontrata nel programma precedente a questo ma non l'abbiamo spiegata.

Procediamo con ordine…

Questo programma è molto semplice ma è utile per far capire le varie possibilità di utilizzo del ciclo do…while.

In questo caso infatti il ciclo è usato per ripeter il programma se il dato inserito corrisponde alle condizioni del "while".

Quindi se il dato inserito sarà uguale a "s" il programma ricomincerà da capo.

Passiamo a spiegare i diversi simboli incontrati nel "while"

Quando al suo interno troviamo:

Simbologia	Significato	Spiegazione
||	Oppure	Controlla se un dato è questo o un altro, permette di fare due controlli contemporanei.
= =	Uguale a	Indica che l'operazione verrà effettuata solo se il dato inserito sarà uguale a quello che si trova nella condizione del "while" il quale dovrà essere inserito tra apici ‘’.
! =	Diverso da	Per continuare col programma il dato inserito deve essere diverso da quello contenuto nel "while".
< =	Minore o uguale di	Accetta il dato solo se è minore o uguale a quello inserito come condizione.
> =	Maggiore o uguale di	Accetta il dato solo se è maggiore o uguale a quello inserito come condizione.

LEZIONE 4

in questa quarta lezione ci occuperemo del ciclo if…else ma prima diamo una veloce spiegazione di alcuni simboli usati nelle lezioni precedenti e una spiegazione di quelli che utilizzeremo nelle lezioni successive.

Qua di seguito mostreremo le "sequenze di escape" e le "direttive" di formattazione dei dati:

Sequenza di escape		Direttive	
\a	Attivazione segnale acustico	**%d**	Un intero
\b	Spazio indietro	**%c**	Un carattere
\n	Nuova riga	**%f**	Numero a virgola mobile
\r	Ritorno carrello	**%u**	Un intero senza segno
\t	Tabulazione orizzontale	**%e**	Un numero a virgola mobile con esponente
\v	Tabulazione verticale	**%s**	Una stringa
		%x	Un numero esadecimale

Adesso che abbiamo velocemente ricapitolato questi concetti possiamo passare alla spiegazione del ciclo **if…else**.

Questo ciclo non è molto differente dal ciclo **do…while** presentato precedentemente (vedi lezione 3).

La struttura è leggermente differente ma la funzione di questi due cicli è pressoché la medesima.

Anche l'if…else, come il do…while, viene chiamato in causa solo se si verifica una particolare condizione logica posta dal programmatore.

La struttura è la seguente:

```
 if (condizione)
{sequenza 1
}
else
{sequenza 2
}
```

Vediamo il tutto in un esempio pratico così da capire meglio ciò che stiamo dicendo: (Programma 5)

```
#include <conio.h>
#include <stdio.h>
int  a;
main()
{printf ("Inserisci un numero maggiore di 2:\t");
 scanf("%d" , & a);
if (a>2)
{printf ("Il numero e' valido");
}
else
{printf ("Il numero inserito non e' corretto");
}
getch();
}
```

Questo esempio mostra come il ciclo if…else sia un valido sostituto del ciclo do…while per quanto riguarda il controllo sui dati inseriti.
Nonostante questa somiglianza in genere i programmatori preferiscono utilizzare il do…while in quanto restringe di molto la lunghezza dei programmi più complessi.
Nel nostro caso l'utilizzo di uno o dell'altro non è significativo perché in questo corso ci limiteremo a creare piccoli programmi.
La particolarità di questo ciclo è che è possibile utilizzare cicli if all'interno di altri cicli if come se fossero una catena.
Avrete sicuramente notato che nell'esempio precedente dopo la scritta "if" ci sono delle parentesi tonde; beh è in quelle parentesi che vanno inserite la condizioni che noi vogliamo che il programma rispetti.
Nel caso precedente la nostra unica condizione era quella che il numero fosse maggiore di 2 ma avremmo anche potuto mettere più di una condizione magari inserendo (a>2 && a<5).
Inserendo queste condizioni richiediamo al programma di accettare solo numeri che siano contemporaneamente maggiori di 2 e minori di 5.

Ecco l'esempio pratico: (Programma 6)

```
#include <conio.h>
#include <stdio.h>
int  a;
main()
{printf ("Inserisci un numero maggiore di 2:\t");
 scanf("%d" , & a);
if (a>2 && a<5)
{printf ("Il numero e' valido");
}
else
{printf ("Il numero inserito non e' corretto");
}
getch();
}
```

LEZIONE 5

Prima di iniziare con la spiegazione ci teniamo a elencare le tipologie di variabili principali utilizzabili con il linguaggio c++:

int valori interi

float valori in virgola mobile

char caratteri

Oltre a variabili singole è possibile creare verrori, o array, e matrici. Cosa sono vettori e matrici? Essenzialmente sono dei gruppi omogenei di variabili contraddistinte da un nome comune e da un indice del tipo [i]. Dichiarando un vettore **int** x[8] si creano 8 variabili contraddistinte ciascuna da un differente indice: x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]; dichiarando una matrice **int** m[2][2] si ottiene un oggetto simile ad vettore, e il numero di variabili contenute è pari al prodotto degli indici dichiarati, comunque di questo ci occuperemo nelle lezioni seguenti, ora focalizziamoci sulla struttura del ciclo for…

Il ciclo for non è altro che un ciclo do… while opportunamente modificato, la struttura della sintassi è però abbastanza diversa e ha delle funzioni molto utili all'interno di ogni tipo di algoritmo. Ecco qui di seguito la struttura:

```
for (condizione)
{
    sequenza
}
```

La sintassi della condizione del ciclo for è insolita rispetto a tutti gli altri cicli, in quanto non è una serie di condizioni logiche legate da concetti di AND (&&), OR (||), UGUALE (==) o DIVERSO (!=) [vedi lezioni precedenti], ma è sempre strutturata in 3 distinte dichiarazioni: un inizio, un termine e un incremento di una precisa variabile, come mostrato di seguito:

```
for ( a = 0; a< 8; a++)
{
    scanf ("%d", x[a]);
}
```

Tale dicitura implica di far assumere alla variabile intera a il valore 0 (zero) e di eseguire le istruzioni all'interno delle graffe. Una volta terminate le istruzioni contenute nelle parentesi la variabile a viene incrementata di un valore (a++) e quindi assumerà il valore 1. Come avrete già capito il processore eseguirà le stesse operazioni per un certo numero di volte, che è proprio uguale al numero scelto nella seconda parte della condizione (a<8). Una volta compiute le operazioni volute per il determinato numero di volte il ciclo termina e il processore prosegue leggendo le istruzioni successive.

Ma a cosa serve il ciclo **for** ? Come avrete notato può essere usato per acquisire un certo numero di variabili all'interno di un vettore, ed è proprio facendo variare l'indice del vettore di volta in volta che si può con una sola istruzione assegnare dei valori ad 8 variabili!

Inoltre, ogni qual volta dovrete effettuare delle operazioni che si ripetono con pochi parametri che variano tra l'una e l'altra, il ciclo for è la soluzione più efficace.

PROGRAMMA SVOLTO:

```
#include <stdio.h>    // libreria per le funzioni di stampa e acquisizione dati
#include <conio.h>    // funzione di pausa fino alla pressione di un tasto
#include <windows.h>  // funzione random (); e Sleep ();
Int i;
main ()
{
   for (i=0; i<8; i++)
    {
                printf ("%d stampa", i+1);
                Sleep(1000);          //tempo di attesa prima di continuare l'operazione
                 }
        getch ();
}
```

Passiamo ora a spiegare quanto fatto nel programma precedente…

Dopo aver dichiarato le librerie necessarie abbiamo dichiarato le variabili, in questo caso usiamo una variabile i intera.

Come di consueto il programma inizia con la dicitura main(){…}

Scorrendo il programma troviamo il ciclo for con delle condizioni interne.

La **i=0** indica il valore da cui partiamo; **i<8** indica quante volte verrà ripetuto il ciclo; i++ indica l'incremento del valore ad ogni passaggio del ciclo.

All'interno delle parentesi {} del for troviamo un printf che indica la volontà di stampare a video dei valori numerici %d seguiti dalla parola "stampa".

Dopo la virgola troviamo la dicitura i+1…con questa artificio grafico indichiamo al programma di stampare dei numeri in sequenza partendo dallo zero per arrivare all'8 ottenendo:

1 stampa 2 stampa…8 stampa;

la funzione Sleep(1000) indica il tempo che dovrà passare tr la stampa di un numero e l'eltro…ovvero 1 secondo.

La funzione getch(), già incontrata nelle lezioni precedenti,permetta la stampa a video del programma.

LEZIONE 6

Costrutto **Switch :**

Ci permette la selezione di una delle possibili opzioni dato il valore di una variabile :

```
  switch (operand) {
        case MULTIPLY:   x *= y; break;
        case DIVIDE:     x /= y; break;
        case ADD:        x += y; break;
        case SUBTRACT:   x -= y; break;
        case INCREMENT2: x++;
        case INCREMENT1: x++;    break;
        case EXPONENT:
        case ROOT:
        case MOD:        printf("Not done\n"); break;
   default:        printf("Bug!\n");

  exit(1);
}
```

Nell'esempio, *operand*, potrebbe assumere uno dei valori (Multiply,Divide,Add,…), a secondo della scelta verrà eseguito il percorso definito che terminerà con l'istruzione *break* ; Nel caso nessuno dei casi rientri all'inetrno delle scelte si finisce nell'opzione di *default*.

LEZIONE 7

Le Strutture di Record

Il presente programma crea una struttura "**Dipendente**" formata da **n=2 Record** a differenza dei vettori mono e bidimensionali le strutture sono arra di dati non omogenei cioè di tipo diversi organizzati in riga e dove a ciascun rigo corrisponde una informazione relativa ad un ben determinato Record

```
// In una azienda lavorano 2 operai, di ogni operaio inserire :
// N. matricola, il livello, il nome, indirizzo, importo stipendio
// Ordinare alfabeticamente i dipendenti che hanno uno stipendio compreso tra 1200 e 1500.

#include<iostream.h>
#include<io.h>
#include<conio.h>
#include<math.h>

int i, n;

void main (void)
{ const n=2;
 clrscr();

  struct dipendente{
     int matricola;
     char livello[20];
     char nome[20];
     char indirizzo[30];
     float stipendio;    };

 dipendente azienda[n];

 for(i=0;i<n;i++)
    {
      cout<<"dammi la matricola: ";
      cin>>azienda[i].matricola;
      cout<<"dammi il livello: ";
      cin>>azienda[i].livello;
      cout<<"dammi il nome: ";
      cin>>azienda[i].nome;
      cout<<"dammi l'indirizzo: ";
      cin>>azienda[i].indirizzo;
      cout<<"dammi il valore dello stipendio: ";
      cin>>azienda[i].stipendio;
    }
 for(i=0;i<n;i++)
     {    if(azienda[i].stipendio>1000&&azienda[i].stipendio<1500)
          cout<<"l'operaio "<<azienda[i].nome<<" riceve uno stipendio di: "<<azienda[i].stipendio<<endl;}
  while(!kbhit());
}
```