SOMMARIO

- Sviluppo ed esecuzione di un programma:
  - Compilatore.
  - Interprete.
  - Compilatore e interprete.
- Sviluppo di un progetto.
- Ambiente integrato di programmazione:
  - Visual Studio.

ESECUZIONE DI UN PROGRAMMA

- Il codice sorgente scritto dal programmatore non è direttamente eseguibile dalla CPU (Central Processing Unit), è necessario *tradurlo* in linguaggio macchina: un *file binario* contenente microistruzioni gestibili dalla CPU.
  - Diverse architetture di processori prevedono diversi linguaggi macchina.
- Un **traduttore** "*elabora*" un codice sorgente (un programma) per trasformarlo in uno equivalente ma in un diverso linguaggio.

ESECUZIONE DI UN PROGRAMMA

- Il **compilatore** traduce un linguaggio ad alto livello in un linguaggio più vicino alla macchina (di solito per poter essere direttamente eseguito dal processore).
- L'**interprete** invece si occupa dell'esecuzione diretta del codice sorgente.
- Si possono distinguere diversi approcci per ottenere l'esecuzione di un programma: basati solo sulla compilazione, solo sull'interpretazione, oppure un approccio misto basato su una fase di compilazione e una fase di interpretazione.

COMPILATORE

- I **compilatori** leggono il programma sorgente (per esempio in linguaggio C) e lo traducono in linguaggio macchina generando un *programma oggetto*. La fase di compilazione è accompagnata dalla rilevazione degli errori. Vediamo le fasi del processo:
  - *Preprocessing*: semplici operazioni di editing automatiche (rimozione commenti e espansione di definizioni) e inclusione di file.
  - *Scanning* (Analisi lessicale): verifica le regole di aggregazione dei caratteri dell'alfabeto in simboli del linguaggio (verifica identificatori, simboli).

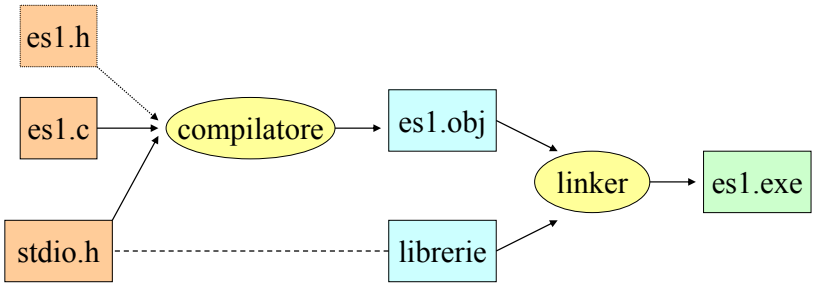
COMPILATORE

- *Parsing* (Analisi sintattica): verifica la correttezza del concatenamento delle parole del linguaggio per formare "frasi": le operazioni vengono riordinate in una struttura gerarchica.
- *Analisi semantica*: determina la compatibilità dei tipi, dei parametri delle funzioni: il significato da attribuire ad ogni "frase".

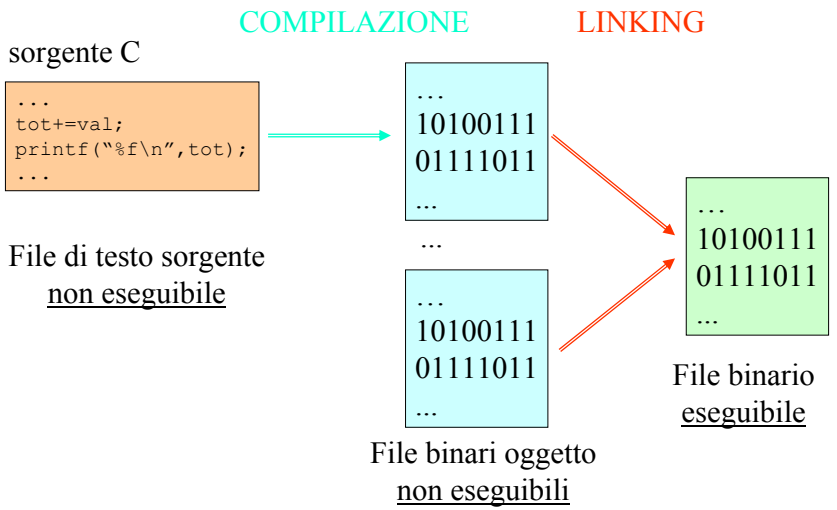
- Per produrre il file eseguibile è necessario *collegare* (utilizzando il **linker***)* tra loro i diversi *file oggetto* e le *librerie di funzioni standard* che sono file forniti insieme al programma compilatore.

COMPILATORE

- Le *librerie di funzioni standard* sono raccolte di piccoli programmi *oggetto* che forniscono particolari funzionalità: per esempio funzioni matematiche, apertura di file, creazione di finestre. I prototipi di tali funzioni standard sono nei *file header*.



COMPILATORE



LIBRERIE DINAMICHE

- Le librerie dinamiche (in Windows i file DLL) vengono caricate in memoria solo quando necessarie al file eseguibile, pertanto non al momento della creazione del file eseguibile.

- Il vantaggio è risparmiare memoria non dovendo caricare tutti i moduli del sistema e di solito non dovendo ricompilare i file eseguibili dopo l'aggiornamento delle librerie.

FILE ESEGUIBILI

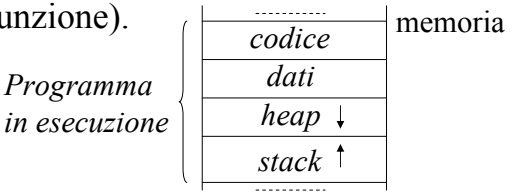
- Gli strumenti di sviluppo sono un insieme di programmi (editor, traduttore, linker, debugger) che consentono la scrittura, la verifica e l'esecuzione di *nuovi programmi* per applicazioni specifiche.

- Il risultato dell'attività di sviluppo di un programma è il *file eseguibile*: un file binario che viene *caricato* in *memoria* e svolge i compiti per cui è stato progettato.

FILE ESEGUIBILI

Il file eseguibile dispone delle seguenti aree di memoria:

- *Area codice*: contiene il codice (come microistruzioni di CPU) del programma.
- *Area dati*: contiene le variabili globali e statiche.
- *Area heap*: disponibile per allocazioni dinamiche.
- *Area stack*: contiene i *record di attivazione* delle funzioni (tutti i dati necessari alla chiamata, esecuzione e ritorno di una funzione).



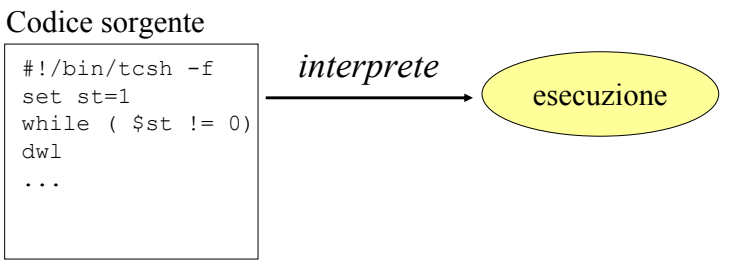
FILE ESEGUIBILI

- Un file eseguibile è specifico per una particolare *architettura hardware e software*, pertanto può risultare non utilizzabile per due ragioni: (1) la CPU ha un set di microistruzioni differenti, (2) il sistema operativo fornisce chiamate a sistema differenti (anche con la stessa CPU).

- I moduli oggetto (moduli rilocabili cioè con riferimenti relativi) di un progetto devono contenere una sola funzione `main()` che richiama tutte le altre funzioni (se una funzione è stata richiamata ma non collegata, il linker genera un errore).

INTERPRETE

- Il codice sorgente viene direttamente tradotto ed eseguito (gli interpreti traducono il programma sorgente sequenzialmente dall'inizio, alternando la fase di traduzione di ciascuna istruzione all'esecuzione effettiva della stessa). Tipici linguaggi sono il BASIC e gli script di shell.



INTERPRETE

- *Vantaggi*: semplicità, portabilità e nessuna perdita di tempo per la compilazione.
- *Svantaggi*: lentezza di esecuzione.

- Linguaggi adatti per programmi di piccole dimensioni e per compiti specifici. Presentano funzionalità ad alto livello di solito specifiche per particolari ambiti applicativi.

INTERPRETE

- Vediamo un esempio di *script di shell* per il *prompt dei comandi*, i file **.bat**: spostarsi nella directory `c:\temp`, salvare su file (`dati.txt`) i nomi di tutti i file presenti, comprimere tale file e spostarlo nella directory `c:\backup`.

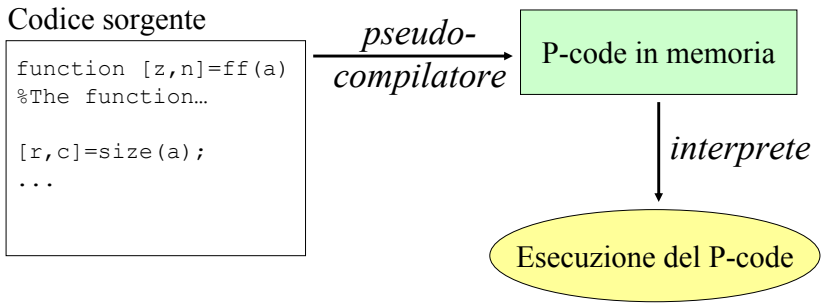
esempio.bat

```
set file=dati
c:
cd c:\temp
dir >%file%.txt
zip  %file%.zip %file%.txt
move %file%.zip c:\backup
```

- Si edita il file `esempio.bat` come un file sorgente C. Al prompt si deve scrivere `esempio.bat` e premere invio come per un file eseguibile.

COMPILATORE + INTERPRETE

- Il sorgente è *compilato* in pseudo-codice intermedio (*P-code*) e caricato in memoria. In seguito è eseguito dall'*interprete* (in modo più veloce che il sorgente originale).



COMPILATORE + INTERPRETE

- Le prestazioni di questi linguaggi sono buone: anche se non si ottengono prestazioni paragonabili a quelle ottenibili con la compilazione. L'idea è quella di simulare una "CPU generica".
- Il vantaggio è la possibilità di avere funzionalità di livello più alto e portabilità.

- Un tipico esempio di linguaggio pseudo-compilato è MATLAB (un ambiente di sviluppo interattivo per il calcolo scientifico).

COMPILATORE + INTERPRETE

- Un caso particolare è il linguaggio Java.
- La differenza fondamentale è che il risultato della pseudo-compilazione viene memorizzato in un *file bytecode* (`.class`), che quindi può essere trasferito su diversi sistemi.
- L'interpretazione è svolta da interpreti (Java Virtual Machine) specifici per la piattaforma su cui è trasferito il P-code: utile quindi per applicazioni portabili come quelle per *Internet.*

LINGUAGGI

- Non esiste un linguaggio ottimo in senso assoluto: bisogna tenere conto dell'ambito in cui si lavora e scegliere il linguaggio da utilizzare caso per caso. Per esempio, con riferimento a questo corso:
  - Programmazione di sistema, calcolo numerico: *C.*
  - Calcolo numerico e grafica: *MATLAB.*
  - Applicazioni Windows: *Visual C++.*
  - Programmazione WEB: *Java.*
  - Gestione di sistemi: *script di shell.*

ATTIVITÀ DI SVILUPPO DI UN PROGRAMMA

- Per realizzare un programma sono necessari almeno due strumenti fondamentali: un *editor* e un *programma di traduzione.*
- L'*editor* è lo strumento che permette di digitare il testo di un *programma sorgente* (il codice simbolico di un linguaggio), di visualizzarlo, modificarlo e memorizzarlo in un file su disco. Gli editor orientati alla programmazione includono funzionalità legate ai linguaggi.

ATTIVITÀ DI SVILUPPO DI UN PROGRAMMA

- Nel caso di programmi di una certa complessità e dimensione, si suddivide il programma sorgente in *moduli* distinti, contenuti in file distinti, più facilmente manipolabili. Sono compilati separatamente, ma alla fine sono collegati in un solo programma eseguibile.
- La suddivisione su più file del programma e la loro gestione prende il nome di *progetto.*

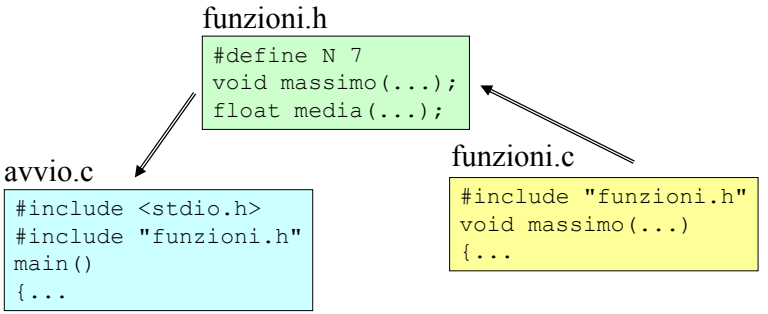
PROGETTO

- Consideriamo un esempio di progetto, utilizzando un compilatore a riga di comando.
- Noi faremo riferimento a un particolare compilatore: il DJGPP, un compilatore C/C++ a 32 bit ANSI C. Documentato e supportato via Internet (*www.delorie.com*).

- Consideriamo un programma che elabori dei dati relativi ai giorni di una settimana: calcoli il valore massimo, il giorno in cui si ha il massimo e il valore medio dei dati inseriti.

PROGETTO

- Sviluppiamo su più file il programma: *avvio.c* che contiene il `main()` e si occupa della gestione di I/O; *funzioni.c* che contiene le definizioni delle funzioni usate da avvio.c; *funzioni.h* che contiene i prototipi delle funzioni e informazioni comuni.



AVVIO.C

```
#include <stdio.h>
#include "funzioni.h"

main()
{
  float dati[N],resv,resm;
  int i,resi;

  printf("Inserire i 7 valori settimanali:\n");
  for(i=0;i<N;i++)
    scanf("%f",&dati[i]);

  massimo(dati,&resv,&resi);
  resm=media(dati);

  printf("\nIl val. max e` %f con ind. %d\n",resv,resi);
  printf("Il val. medio e` %f\n",resm);
}
```


FUNZIONI.H

```
#define N 7

void massimo(float *vett,float *v,int *ind);
float media(float *vett);
```

FUNZIONI.C

```
#include "funzioni.h"
float media(float *vett)
{
  float temp=0;
  int i;
  for(i=0;i<N;i++)
    temp+=vett[i];

  return (temp/N);
 }
```

FUNZIONI.C

```
void massimo(float *vett,float *v,int *ind)
{
  float tempv;
  int i,tempi;

  tempv=vett[0];
  for(i=0;i<N;i++)
    if (vett[i]>=tempv)
      {
        tempv=vett[i];
        tempi=i;
       }
   *v=tempv;
   *ind=tempi;
 }
```


COMPILAZIONE

- Si possono compilare i singoli file per rilevare gli errori e generare i relativi file oggetto:
  gcc -c funzioni.c (crea il file **funzioni.o**)
- Si utilizza *gcc* per collegare i diversi file oggetto in un file eseguibile:
  gcc -o avvio.exe avvio.o funzioni.o
- Si può utilizzare *gcc* per gestire file oggetto e file di codice:
  gcc -o avvio.exe avvio.c funzioni.o

ESECUZIONE

```
G:\fabio\informatica1\Prog_avanzata\es_gcc_files>avvio.exe
Inserire i 7 valori settimanali:
1  2  3  4  5  6  7

 Il valore massimo e` 7.000000 con indice 6
 Il valore medio e` 4.000000

G:\fabio\informatica1\Prog_avanzata\es_gcc_files>avvio.exe
Inserire i 7 valori settimanali:
0  12  3  4  20  3  5

 Il valore massimo e` 20.000000 con indice 4
 Il valore medio e` 6.714286
```


AMBIENTE INTEGRATO DI SVILUPPO

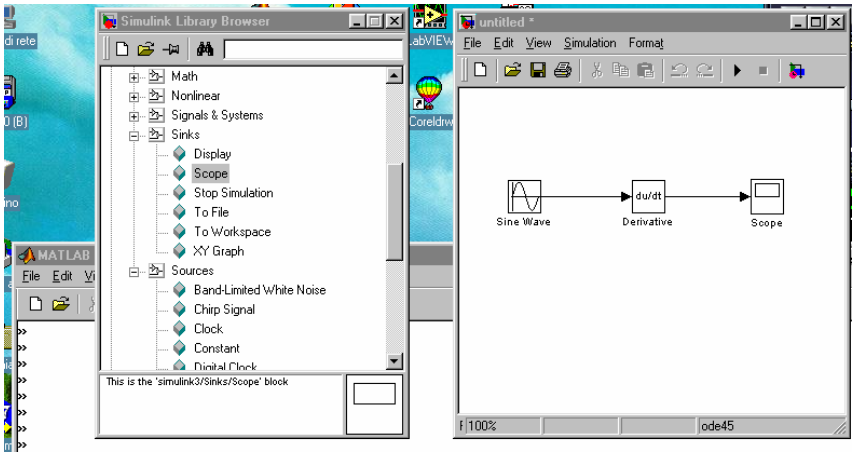
- La gestione di un progetto può essere svolta con utility a linea di comando (*make* e *Makefile*), o con applicazioni che integrano in un solo prodotto i diversi strumenti: si parla di *ambienti integrati di sviluppo* (IDE, *Integrated Development Environment)*.
- In tali ambienti sono forniti editor sensibile al linguaggio, compilatore, linker e debugger; inoltre sono disponibili molti strumenti per la gestione dei diversi file del progetto.
- Noi faremo riferimento ad ambienti Windows: *Microsoft Visual Studio*.

AMBIENTE INTEGRATO DI SVILUPPO

- Il paradigma di *programmazione visuale*, invece, implica la possibilità di sviluppare applicazioni semplicemente spostando icone e riempendo forme: disegnare nello spazio lavoro i diversi blocchi del programma e collegarli insieme (per esempio: LabVIEW, gestione strumentazione elettronica; Simulink, analisi sistemi dinamici).

- Da un lato vi è la semplicità di utilizzo e la forte specificità delle funzionalità disponibili, dall'altro la difficoltà di sfruttare tali software in ambiti diversi e prestazioni non ottimizzate.

AMBIENTE INTEGRATO DI SVILUPPO



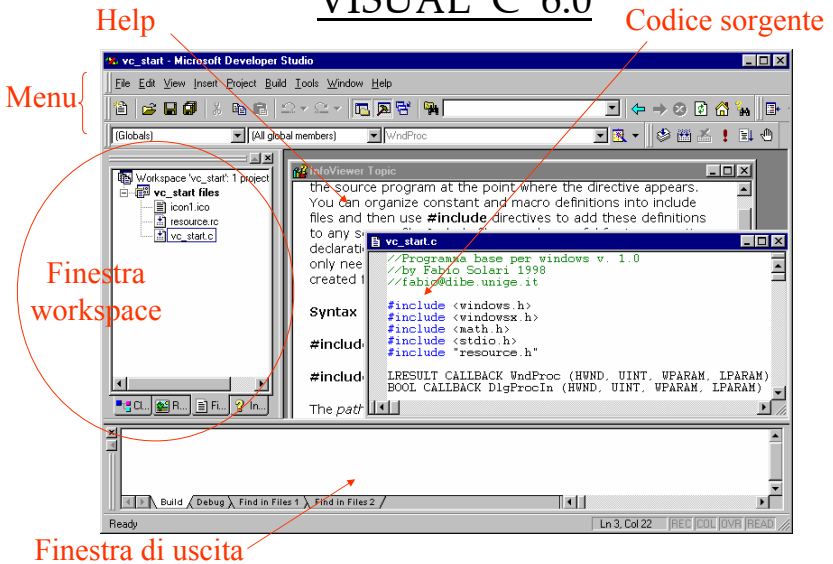
VISUAL STUDIO

- L'IDE di riferimento in ambiente Windows si chiama Microsoft Visual Studio (6.0; o 2005 .NET).
- Vediamo alcuni termini utili:
  - *Build* (costruire): significa compilare e linkare i file sorgenti di un progetto in un eseguibile.
  - *Project*: ha due significati, (i) il prodotto finale, cioè l'applicazione, o (ii), più correttamente, l'insieme di file che compongono l'applicazione (file sorgenti, file precompilati, file grafici), tutto ciò che serve per costruire l'applicazione.

VISUAL STUDIO

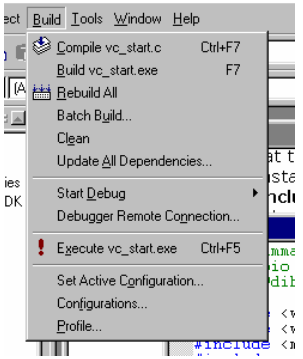
- *Target*: indica il tipo di applicazione che si sta sviluppando e se è una versione *debug* o *release.*
- *Configuration e options*: tutte le opzioni del progetto (dalla directory di lavoro ai colori dei caratteri).
- *Workspace file o Solution Explorer*: contiene informazioni sui diversi progetti in corso.
- *Project file*: contiene informazioni specifiche al singolo progetto (file sorgenti, risorse, configurazioni, informazioni di debug).

VISUAL C 6.0



VISUAL C 6.0

- Alle diverse funzionalità si accede attraverso i menu o le toolbar.

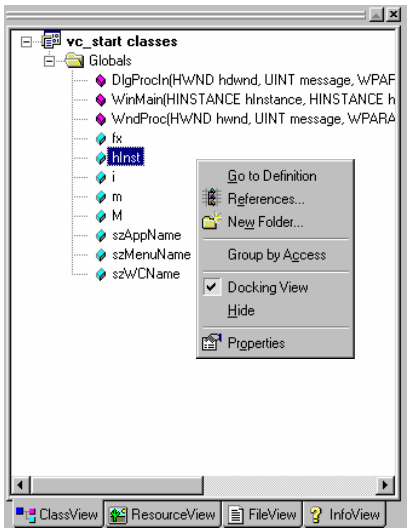


VISUAL C 6.0

- Vi è la zona in cui si aprono le finestre di help e dei file sorgenti. L'editor è sensibile al linguaggio.
- L'area relativa al workspace fornisce informazioni su quattro diversi aspetti del progetto: class view, resource view, file view e info view.
- La finestra di output descrive lo stato della compilazione e del debug.

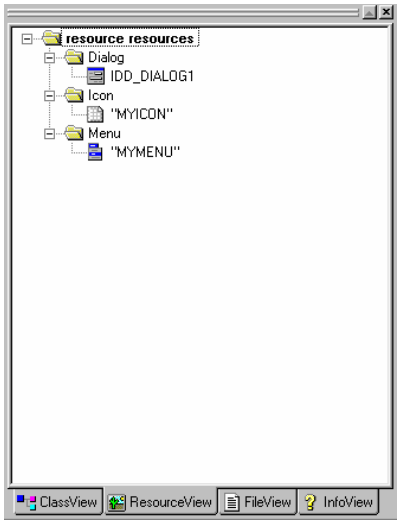
VISUAL C 6.0

Class view: visualizza e fornisce informazioni (sulle classi), sulle funzioni e variabili globali del progetto.
Cliccando col pulsante destro del mouse si ottengono ulteriori informazioni (come negli altri *tab* della finestra di workspace).



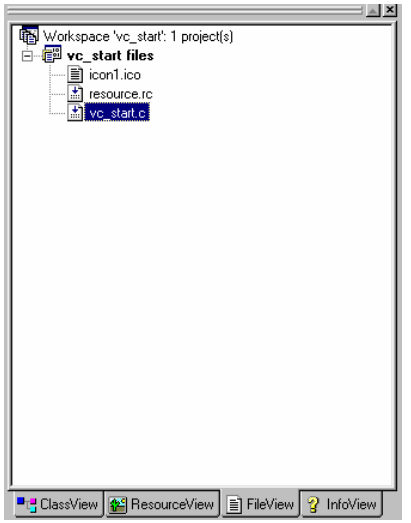
VISUAL C 6.0

Resource view: visualizza e permette di gestire le risorse del progetto, come box di dialogo, icone e menu.



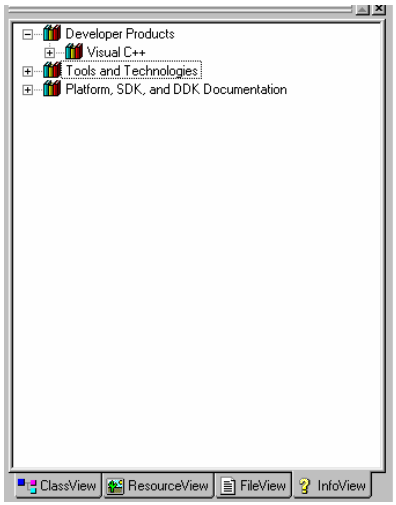
VISUAL C 6.0

File view: visualizza e gestisce i file sorgente e risorse del progetto.



VISUAL C 6.0

Info view: permette di accedere all'ampio *help in linea* fornito con l'ambiente di sviluppo.

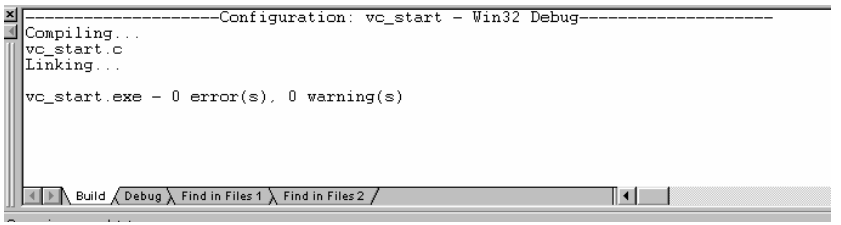


VISUAL C 6.0

•Il *build tab* fornisce informazioni sullo stato della compilazione e collegamento dei file del progetto.

•Il *debug tab* notifica informazioni sul debug del programma.

•Gli altri due *tab* sono relative alla ricerca di occorrenze nei file ottenibile attraverso *Find in Files* del menu *edit.*



WINDOWS

- Per poter programmare in un ambiente Windows, è necessario conoscere in modo generale come opera.

- Le caratteristiche principali di tale sistema operativo sono:
  - *32 bit,* quindi può utilizzare 4 gigabyte di memoria virtuale con spazio di *indirizzamento lineare* (attualmente si sta passando ai *64 bit*, seguendo *l'evoluzione hardware* delle CPU).
  - *Multitasking,* può eseguire "contemporaneamente" due o più programmi. I programmi *condividono la CPU* e non vengono eseguiti effettivamente insieme, (in caso di sistemi *multi-core* o *multi-processore* più programmi sono effettivamente eseguiti in parallelo).

WINDOWS

- Windows supporta due forme di multitasking: il multitasking basato sul *processo* o quello basato sul *percorso* (*thread*).
- Un *processo* è costituito da un programma in esecuzione, quindi possono essere eseguiti più programmi alla volta.
- Un *thread* è un'unità di codice eseguibile di un programma che può essere passata all'esecuzione. Tutti i processi hanno almeno un *thread*, o più di uno. Quindi un programma può avere più porzioni di codice eseguite contemporaneamente (si ottengono applicazioni molto efficienti).

WINDOWS

- L'interfaccia al S.O. Windows è *basata sulle chiamate.*
- Per accedere alle caratteristiche del S.O. si utilizzano un insieme di *funzioni di sistema predefinite* (allocazione della memoria, output a video, creazione di finestre).
- L'insieme di queste funzioni viene detto *Application Program Interface* (API).
- Le API sono contenute in librerie di collegamento dinamiche (DLL), quindi non sono unite ad ogni programma eseguibile, ma sono caricate durante l'esecuzione.

WINDOWS

- Windows supporta un tipo speciale di finestra, detta *console*, che fornisce un'interfaccia in modalità testo di tipo standard, basata su *prompt.*

- Questo permette di scrivere applicazioni in modo simile allo stile degli ambienti Unix/Linux o DOS, senza dover gestire *esplicitamente* i problemi relativi alla gestione delle finestre.

ESEMPI

- Sviluppiamo due diversi tipi di applicazioni:
  - *console*: quindi una programmazione con interfaccia testuale, ma gestita con l'IDE. Inoltre in questo esempio si affronta lo sviluppo di un progetto su più file e il relativo debug.
  - *Win32*: l'attenzione è rivolta allo sviluppo di un programma che crea una finestra e gestisce alcune funzionalità grafiche.

ESEMPIO: CONSOLE

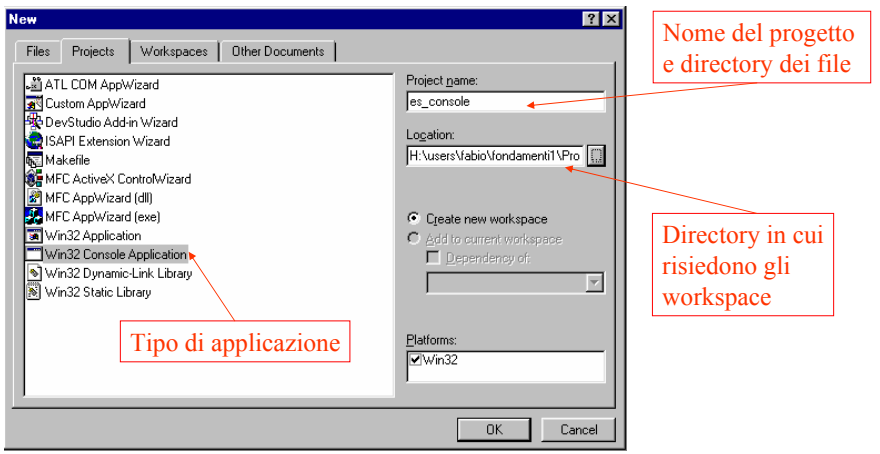
- Consideriamo un programma che fornisca la conversione di una temperatura da gradi Celsius a Fahrenheit e viceversa: si passa da tastiera il valore numerico e un carattere che indica la scala di rappresentazione.

$$T_{Celsius} = \left(\frac{5}{9}\right)\left(T_{Fahrenheit} - 32\right)$$

- Sviluppiamo il programma come progetto: un file che gestisce l'I/O (contiene il `main()`), un file *header* (contiene costanti e dichiarazioni di funzioni) e un file di elaborazione (contiene le definizioni delle funzioni).

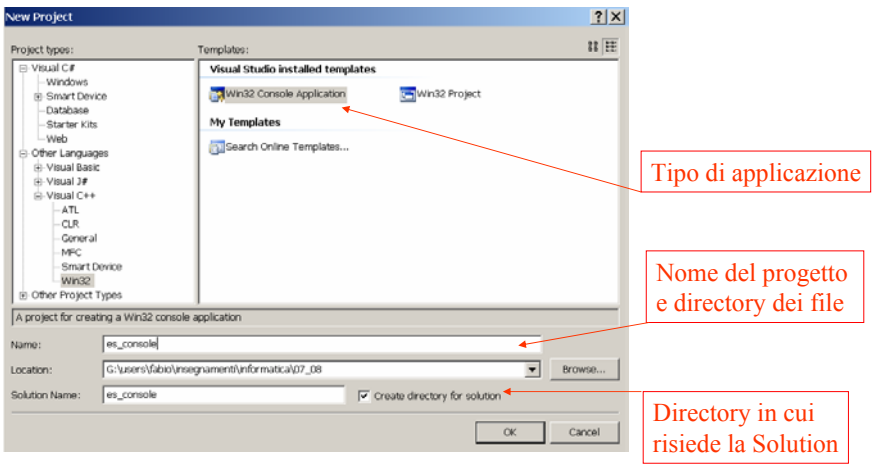
PROGETTO: VISUAL C 6.0

Dal menu *File* del Visual C scegliere *New,* e dal box di dialogo cliccare sul *tab Project.*



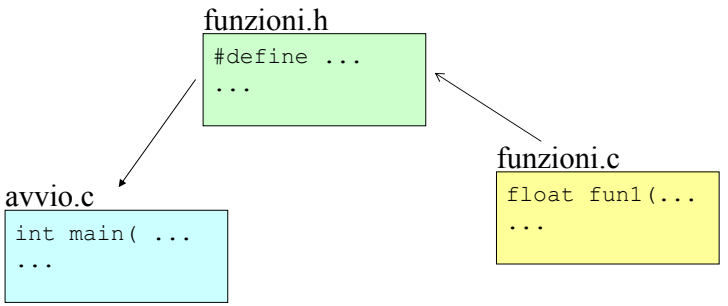
PROGETTO: VISUAL STUDIO

Dal menu *File* del Visual Studio 2005 scegliere *New* e *Project…*

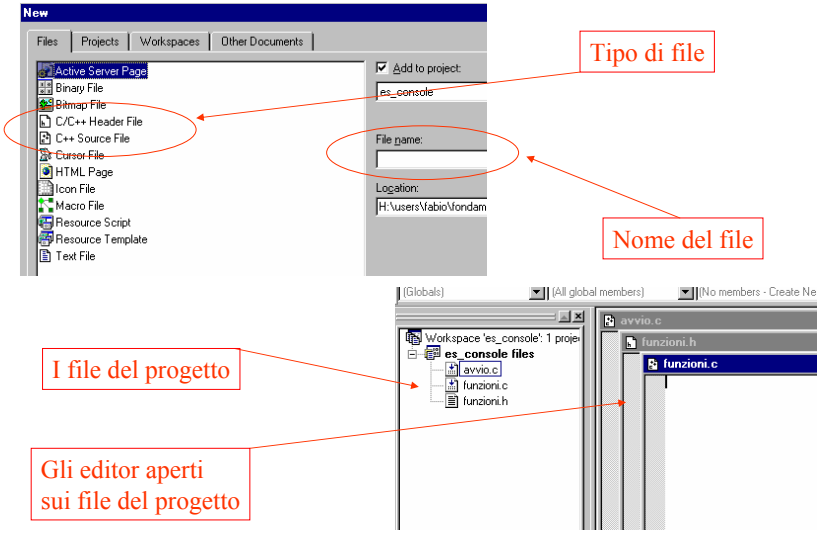


PROGETTO : VISUAL C 6.0

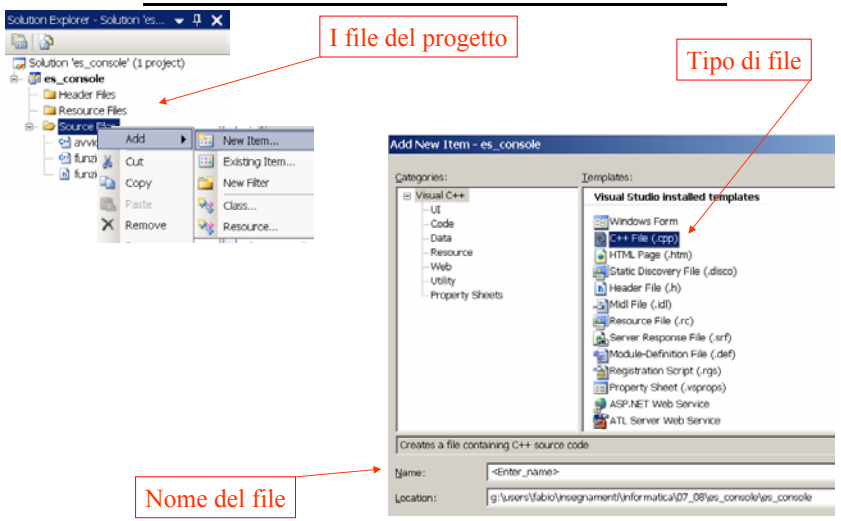
- Dal menu *File* del Visual C 6.0 scegliere *New,* e dal box di dialogo cliccare su *Files* e scegliere il tipo di file per un progetto strutturato come segue:



PROGETTO : VISUAL C 6.0



PROGETTO : VISUAL STUDIO



AVVIO.C

```
#include <stdio.h>
#include <stdlib.h>
#include "funzioni.h"

int main( ){
  float T, res;
  char s;

  printf("Ins. un val. di temp. (e.g. 10c o 12f):\n");
  scanf("%f%c",&T,&s);

  if (s=='c'){
    res=cel2fah(T);
    printf("%.2f Celsius = %.2f Fahrenheit\n",T,res);
  } else{
    res=fah2cel(T);
    printf("%.2f Fahrenheit = %.2f Celsius\n",T,res);}
return 0;}
```

FUNZIONI.H

```
/*file header…*/

#define C1 5.
#define C2 9.
#define TH 32.

float cel2fah(float T);
float fah2cel(float T);
```


FUNZIONI.C

```
/*file elaborazione…*/
#include <math.h>
#include "funzioni.h"

float cel2fah(float T)
{
      double out;
      out=(C2/C1)*T + TH;
      return (float)(out);
}

float fah2cel(float T)
{
      double out;
      out=(C1/C2)*(T - TH);
      return (float)(out);
}
```

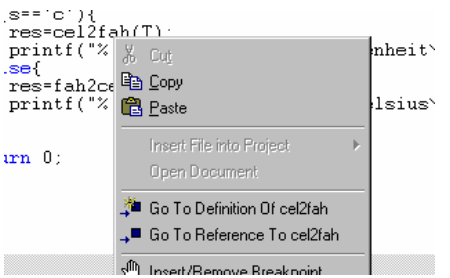

COMPILAZIONE

- Compilare e linkare il progetto (*Build*). Lanciare (*Start*) il programma. Oppure aprire un "Prompt dei comandi", spostarsi nella directory del progetto e lanciare l'eseguibile:

```
Z:\fabio>cd Z:\fabio\fondamenti1\Prog_avanzata\es_console
Z:\fabio\fondamenti1\Prog_avanzata\es_console>./Debug/es_console
Inserire un valore di temperatura (e.g. 10c o 12f):
32f
32.00 Fahrenheit = 0.00 Celsius
Z:\fabio\fondamenti1\Prog_avanzata\es_console>.\Debug\es_console
Inserire un valore di temperatura (e.g. 10c o 12f):
0c
0.00 Celsius = 32.00 Fahrenheit
Z:\fabio\fondamenti1\Prog_avanzata\es_console>.\Debug\es_console
Inserire un valore di temperatura (e.g. 10c o 12f):
98.6f
98.60 Fahrenheit = 37.00 Celsius
```

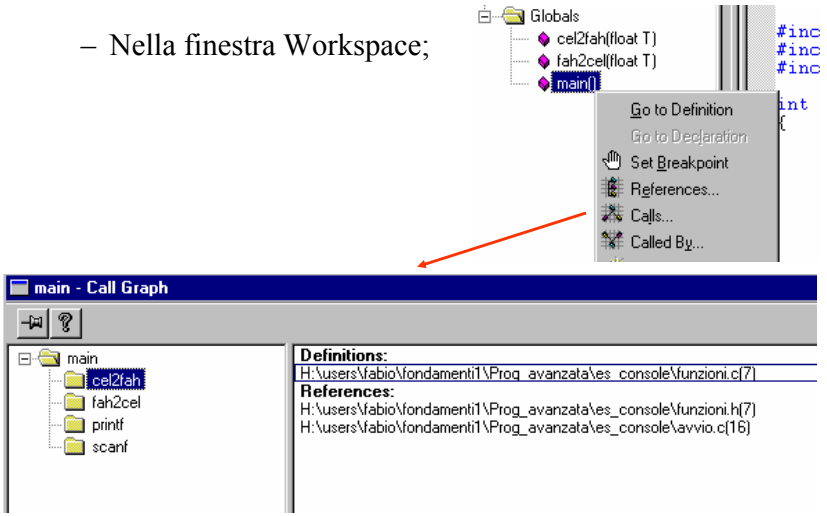

STRUMENTI

- Sono forniti molti strumenti per gestire lo sviluppo del progetto: in generale cliccando col pulsante destro si attivano menu contestuali:
  - nell'editor;



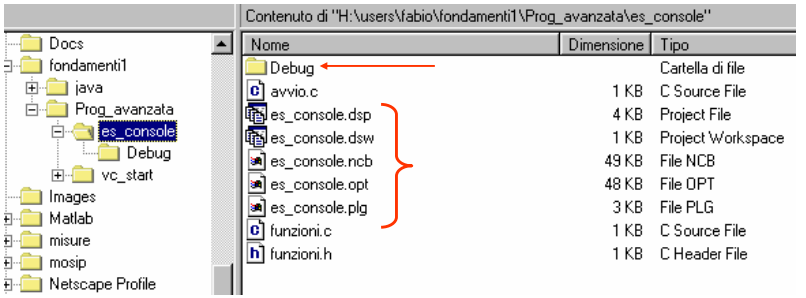
STRUMENTI

– Nella finestra Workspace;



STRUMENTI

- Questo insieme di funzionalità è gestito automaticamente dall'IDE attraverso un insieme di file (non direttamente editati dal programmatore).



ERRORI E DEBUG

- Esistono due tipi principali di errori: quelli rilevati durante la fase di *Build* e quelli, più insidiosi, di *esecuzione*, cioè risultati non attesi (fase di test dell'applicazione).

- Inseriamo alcuni errori nel codice visto e consideriamo come risolverli (*debug*).

ERRORI IN FASE DI BUILD

- Quando vi è un errore, questo compare nella finestra di output dell'IDE.

```
Compiling...
avvio.c
H:\users\fabio\fondamenti1\Prog_avanzata\es_console\avvio.c(15):
                        error C2143: syntax error : missing ';' before 'if'
H:\users\fabio\fondamenti1\Prog_avanzata\es_console\avvio.c(16) :
                        warning C4020: 'cel2fah' : too many actual parameters
Creating browse info file...
es_console.exe - 1 error(s), 1 warning(s)
```

ERRORI IN FASE DI BUILD

- Cliccando sull'errore il controllo passa sulla riga corrispondente del codice sorgente.

Errore

```
scanf("%f%c",&T,&s)
if (s=='c'){
```

Warning

```
if (s=='c'){
    res=cel2fah(T,1);
    printf("%.2f Celsius =
```


DEBUG

- Supponiamo di eseguire il programma e di ottenere:

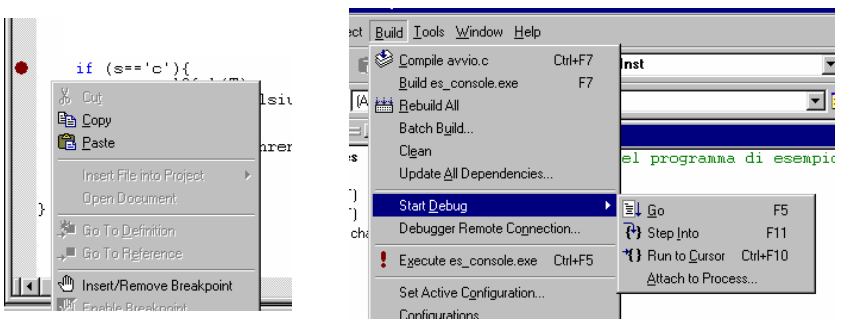
```
Z:\fabio\fondamenti1\Prog_avanzata\es_console>.\Debug\es_console
Inserire un valore di temperatura (e.g. 10c o 12f):
32c
32.00 Celsius = 64.00 Fahrenheit

Z:\fabio\fondamenti1\Prog_avanzata\es_console>.\Debug\es_console
Inserire un valore di temperatura (e.g. 10c o 12f):
64f
64.00 Fahrenheit = 0.00 Celsius
```

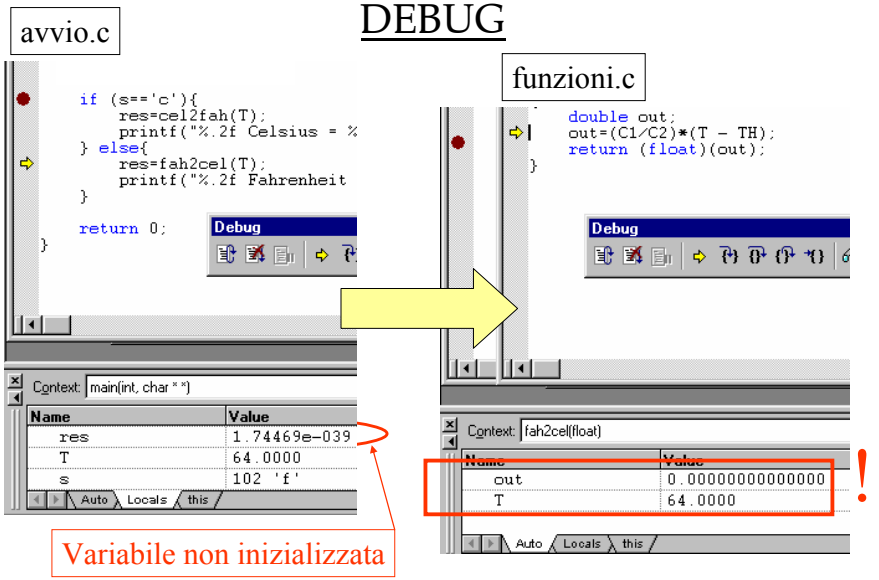
- Questo è un errore che non è stato rilevato nella fase di *Build*. E sembra che sia prodotto nella conversione da Fahrenheit a Celsius e non viceversa.

DEBUG

- Attiviamo la sessione di *debug*: si inserisce un *breakpoint* nel codice sorgente. Da quel punto si interagisce con un insieme di strumenti che permettono di seguire passo passo ogni riga di codice e verificare i valori delle espressioni mentre il programma è in esecuzione.



DEBUG



DEBUG

- Procedendo dal `main()` si entra nella funzione `fan2cel()` e si trova che l'espressione assegnata a `out` vale zero, considerando che la variabile `T` ha il valore corretto, si controllano le costanti:

```
#define C1 5
#define C2 9
#define TH 32

float cel2fah(float T);
float fah2cel(float T);
```

- L'errore è aver dichiarato `C1` e `C2` intere: in tal modo `5/9` vale `0` (anche `9/5` non è corretto!).