

COSTRUZIONE DI UN'APPLICAZIONE

Per costruire un'applicazione occorre:

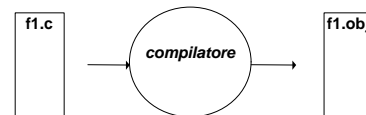
- compilare il file (o i file se più d'uno) che contengono il testo del programma (file *sorgente*)
Il risultato sono uno o più file *oggetto*.
- collegare i file oggetto l'uno con l'altro e con le librerie di sistema.

1

COMPILAZIONE DI UN'APPLICAZIONE

1) Compilare il file (o i file se più d'uno) che contengono il testo del programma

- File *sorgente*: estensione **.c**
- File *oggetto*: estensione **.o** o **.obj**



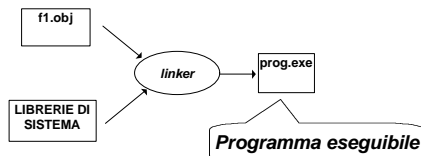
f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

2

COLLEGAMENTO DI UN'APPLICAZIONE

2) Collegare il file (o i file) oggetto fra loro e con le librerie di sistema

- File *oggetto*: estensione **.o** o **.obj**
- File *eseguibile*: estensione **.exe** o nessuna



3

COLLEGAMENTO DI UN'APPLICAZIONE

LIBRERIE DI SISTEMA:

insieme di componenti software che consentono di interfacciarsi col sistema operativo, usare le risorse da esso gestite, e realizzare alcune "istruzioni complesse" del linguaggio

4

COSTRUZIONE "MANUALE"

In passato, la costruzione si faceva "*a mano*", attivando *compilatore* e *linker* dalla *linea di comando* del sistema operativo (DOS, Unix, ...)

C:\PROVA> gcc -c f1.c
(genera f1.obj)

C:\PROVA> ld -o prog.exe f1.obj -lc
(genera prog.exe)



5

AMBIENTI INTEGRATI

Oggi, gli ambienti di lavoro integrati automatizzano la procedura:

- compilano i file sorgente (se e quando necessario)
 - invocano il linker per costruire l'eseguibile
- ma per farlo devono sapere:
- quali file sorgente costituiscono l'applicazione
 - il nome dell'eseguibile da produrre.

6

PROGETTI

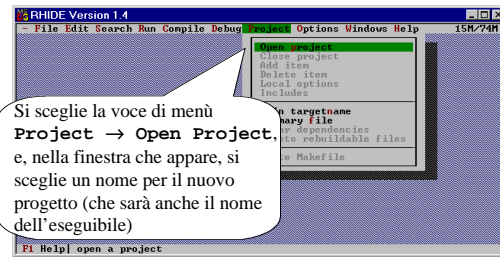
È da queste esigenze che nasce il concetto di **PROGETTO**

- un *contenitore concettuale (e fisico)*
- che *elenca i file sorgente in cui l'applicazione è strutturata*
- ed *eventualmente altre informazioni utili.*

Oggi, *tutti* gli ambienti di sviluppo integrati, *per qualunque linguaggio*, forniscono questo concetto e lo supportano con idonei strumenti.

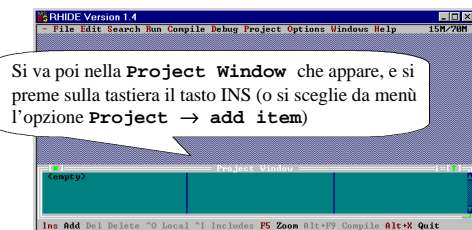
7

PROGETTI IN DJGPP/RHIDE



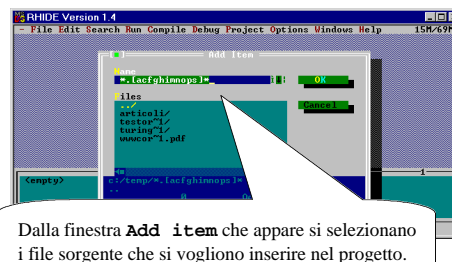
8

PROGETTI IN DJGPP/RHIDE



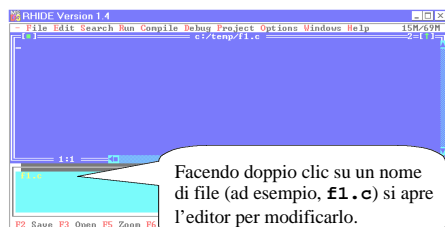
9

PROGETTI IN DJGPP/RHIDE



10

PROGETTI IN DJGPP/RHIDE



11

PROGETTI IN DJGPP/RHIDE



12

IL DEBUGGER

Una volta scritto, compilato e collegato il programma (ossia, costruito l'eseguibile)

occorre uno strumento che consenta di

- eseguire il programma passo per passo
- vedendo le variabili e la loro evoluzione
- e seguendo le funzioni via via chiamate.



Debugger

13

ESEGUIRE IL PROGRAMMA

Sia **Rhide** sia altri ambienti di sviluppo incorporano un *debugger* con cui eseguire il programma,

- riga per riga
 - entrando anche dentro alle funzioni chiamate (*Run* → *trace into*; F7)
 - oppure considerando le chiamate di funzione come una singola operazione (*Run* → *step over*; F8)
- oppure fino alla riga desiderata (*Run* → *Go to cursor*; F4)

14

ESEGUIRE IL PROGRAMMA

... e con cui inoltre è possibile:

- controllare istante per istante quanto vale una variabile
 - *Debug* → *Watch an expression* (CTRL+F7)
- vedere istante per istante le funzioni attive (*record di attivazione nello stack*)
 - *Debug* → *Call stack* (CTRL+F3)

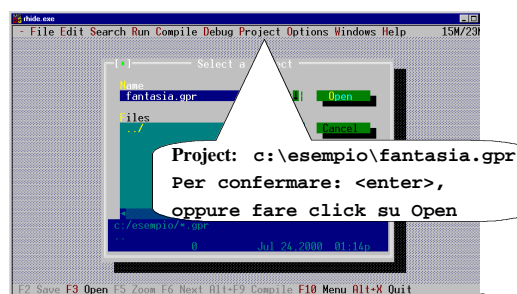
15

In sintesi

Run	esegue il programma dalla prima istruzione all'ultima
Step over	esegue la funzione come istruzione singola
Trace into	esegue passo-passo tutte le istruzioni di una funzione
Go to the cursor	esegue il programma fino alla posizione del cursore e poi passo-passo
Program reset	interrompe la fase di esecuzione (in debugging)

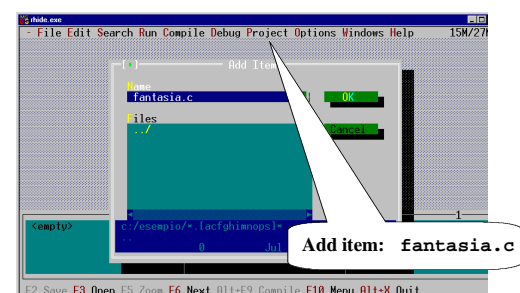
16

UN ESEMPIO COMPLETO

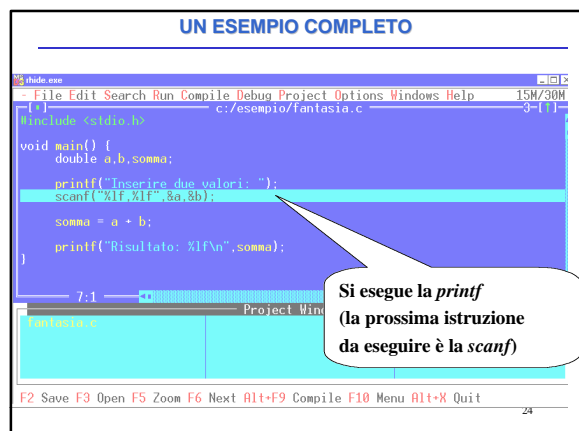


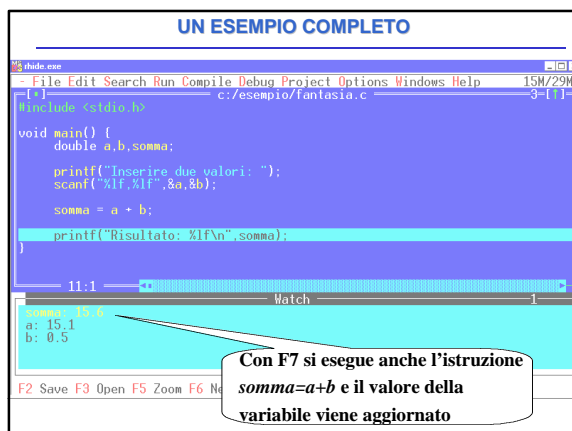
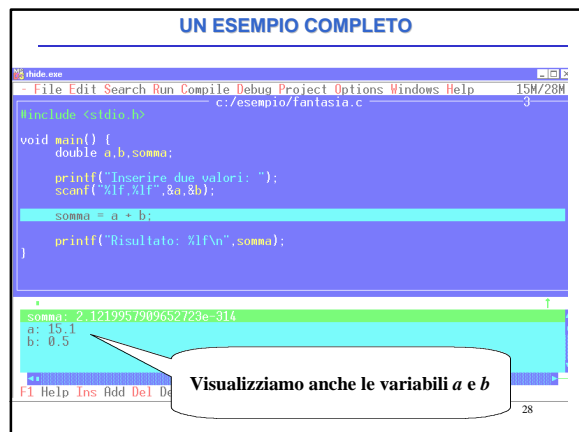
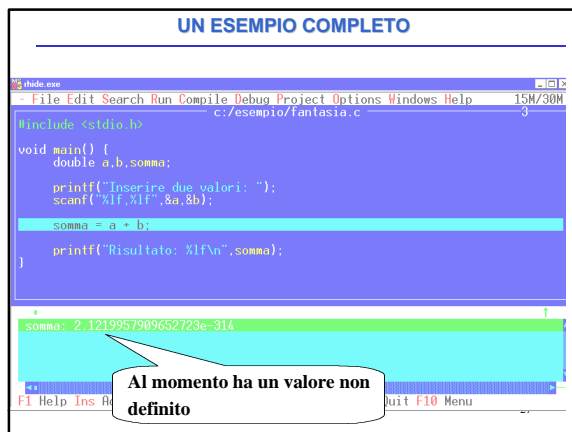
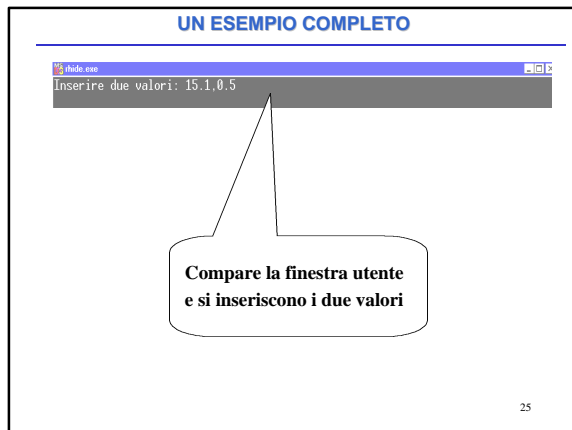
17

UN ESEMPIO COMPLETO



18





UN ESEMPIO COMPLETO



```
thede.exe
Inserire due valori: 15.1 0.5
Risultato: 15.600000
```

Per vedere l'effetto della *printf*
si passa alla finestra utente
(ALT+F5)

31