Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni e
Ingegneria dell'Automazione
a.a. 2009/2010

Lab 01
Introduzione a LCC

Costruzione di un'Applicazione

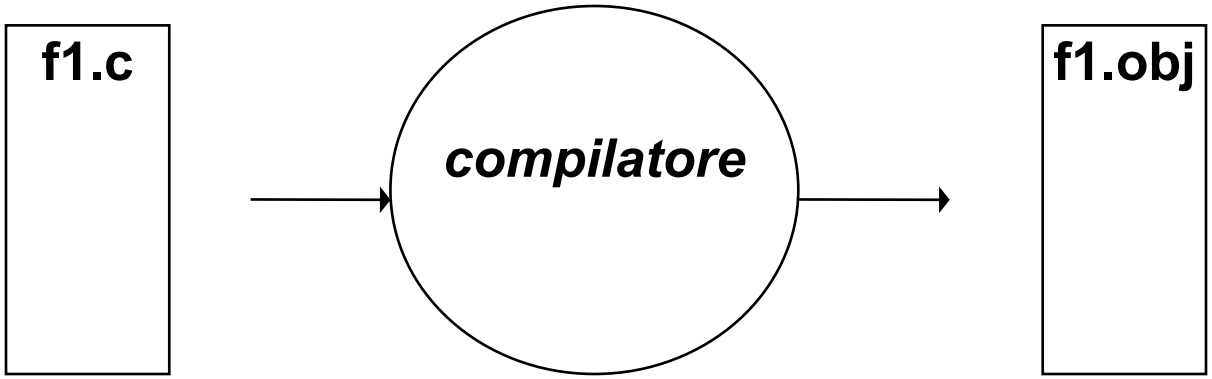
Per costruire un'applicazione occorre:

- **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.***

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`**

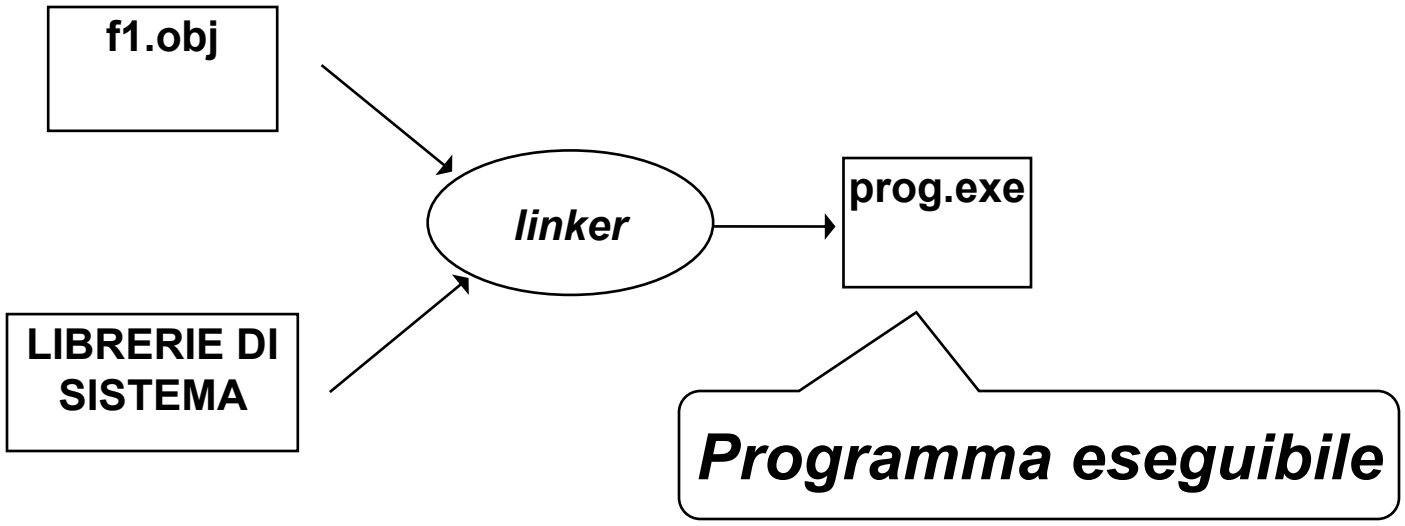


f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

Collegamento (Linking) 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**



Collegamento (Linking) 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

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.**

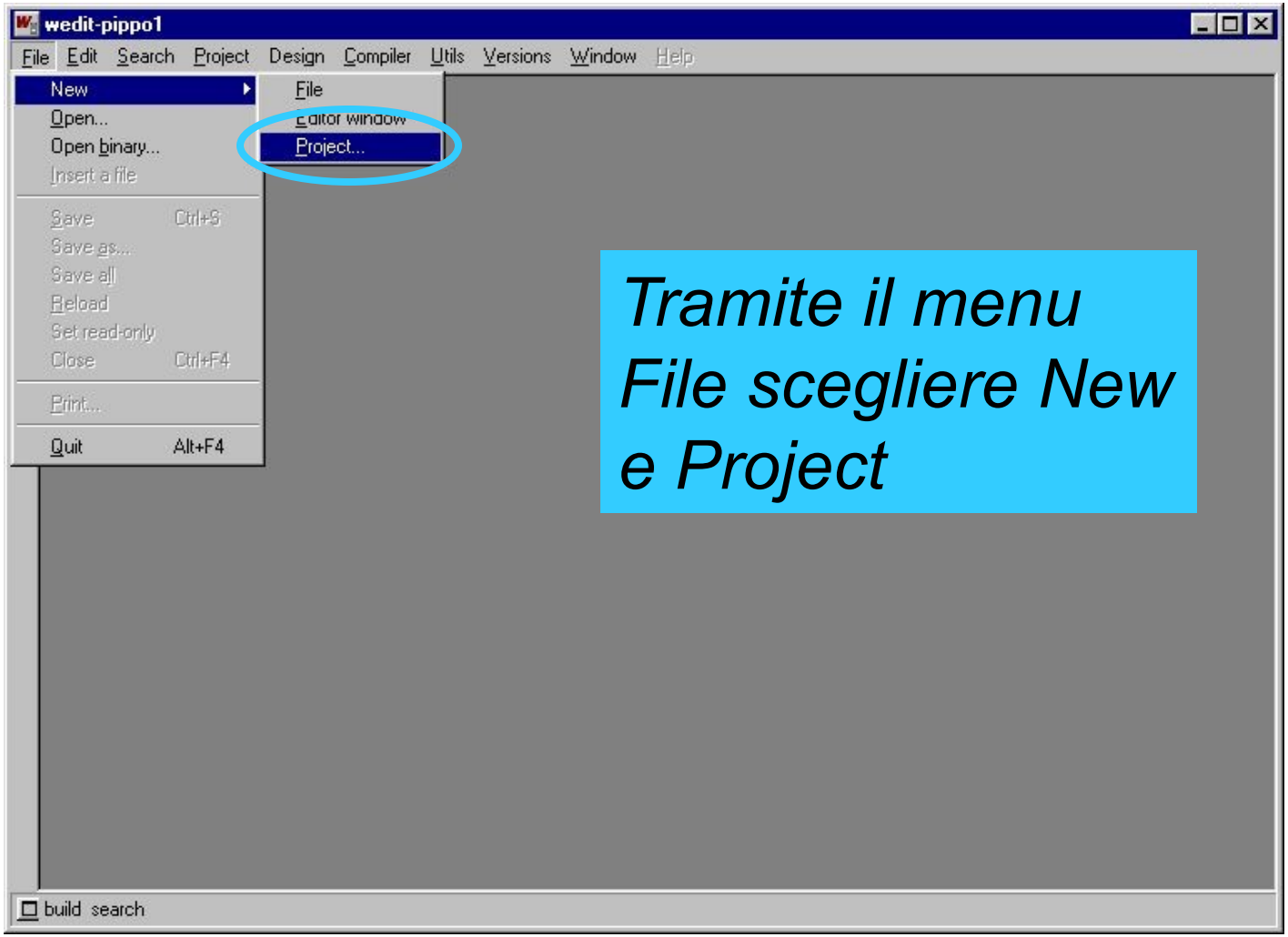
Progetti

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

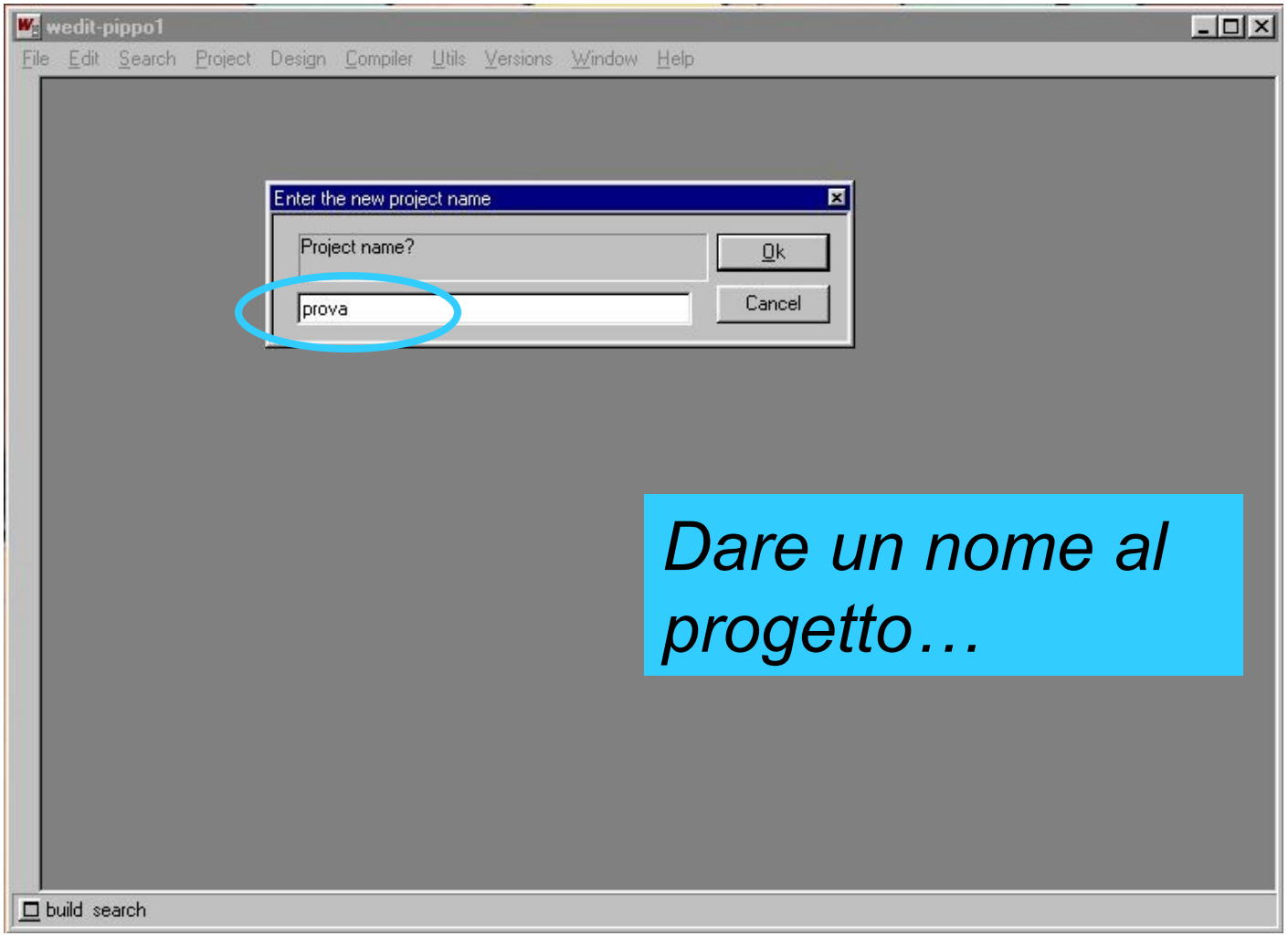
- **un *contenitore concettuale (e fisico)***
- **che *elenca i file sorgente* in cui l'applicazione è strutturata**
- **ed eventualmente altre informazioni utili.**

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

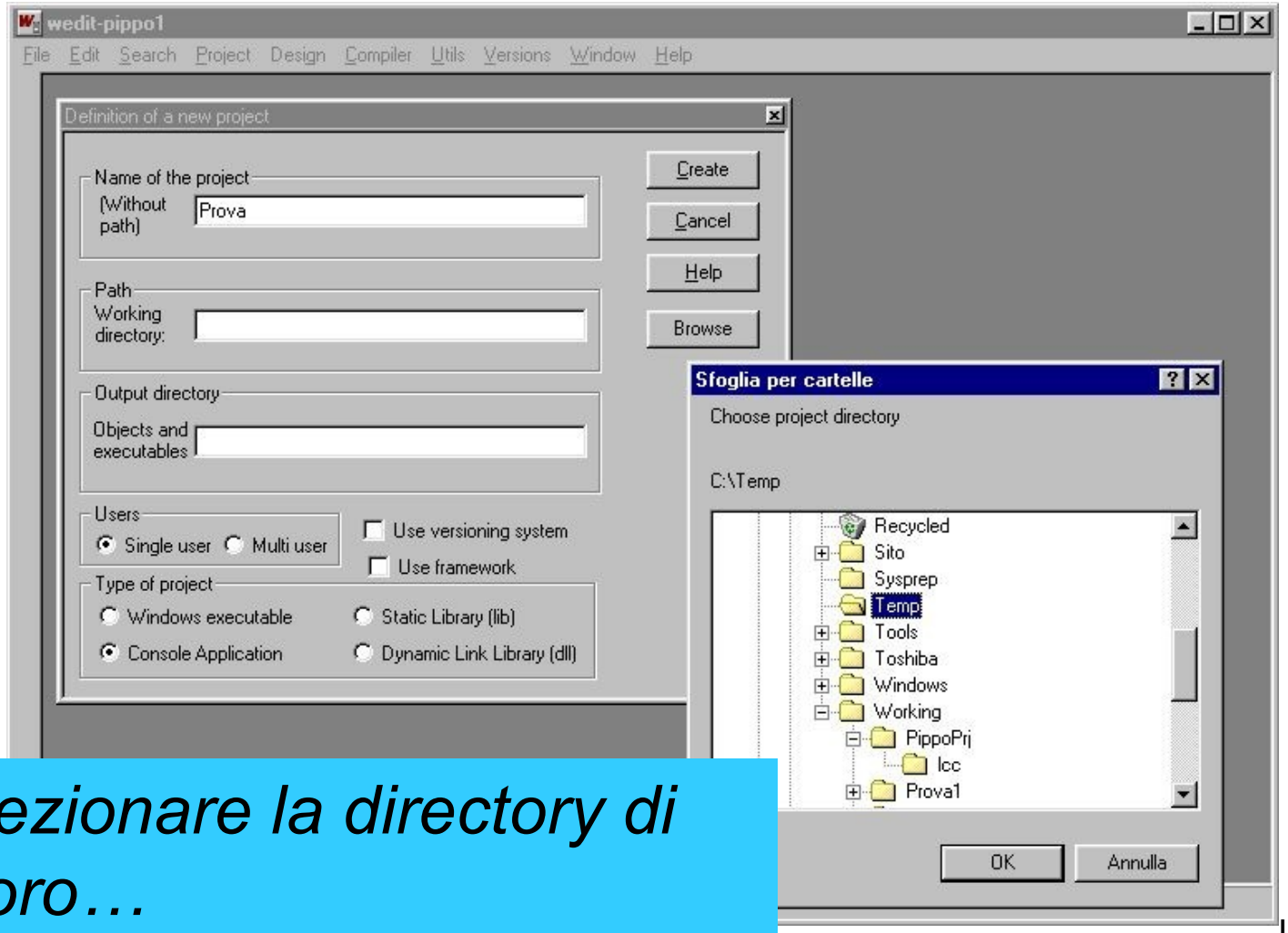
Progetti in LCC



Progetti in LCC

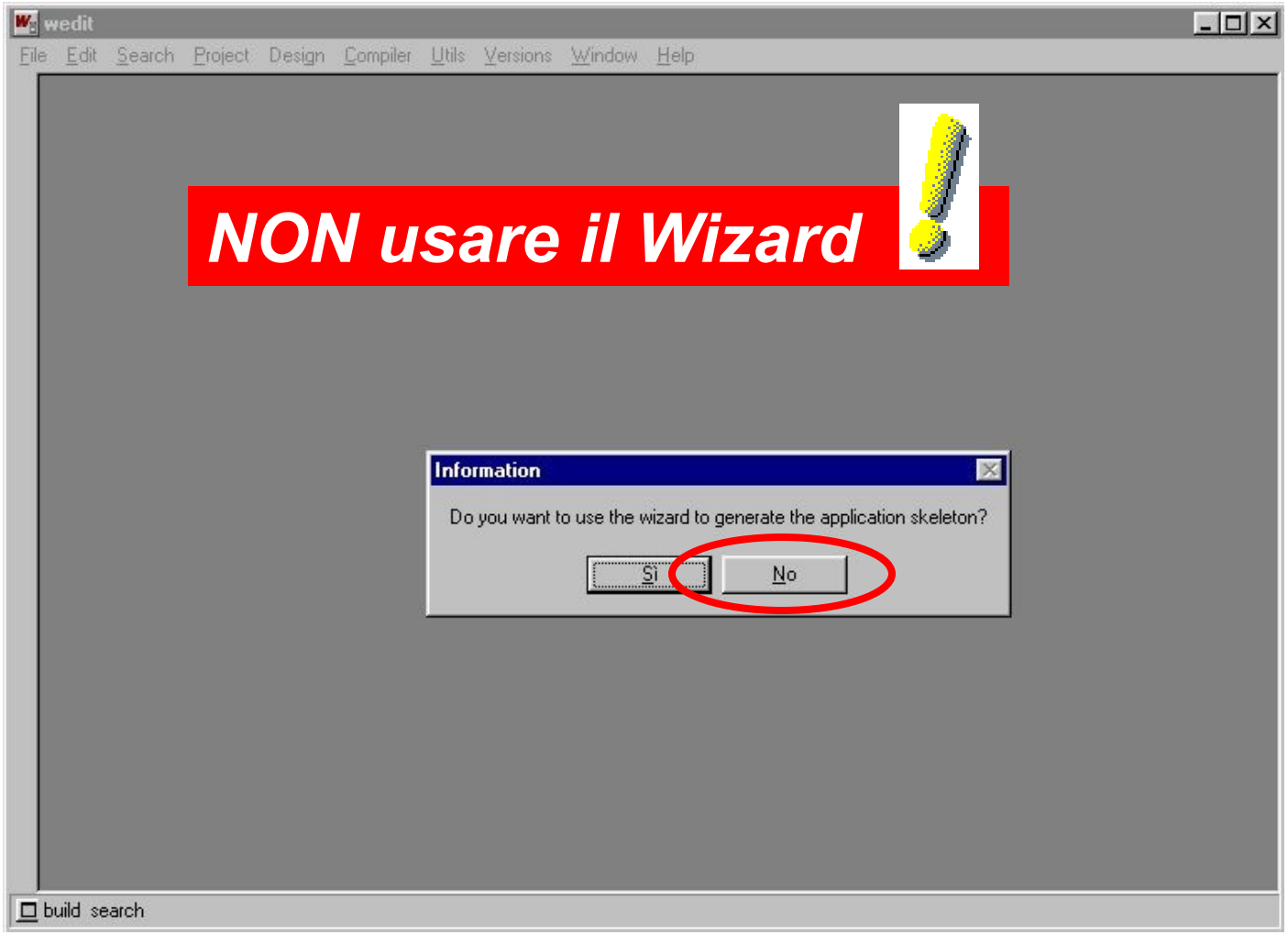


Progetti in LCC

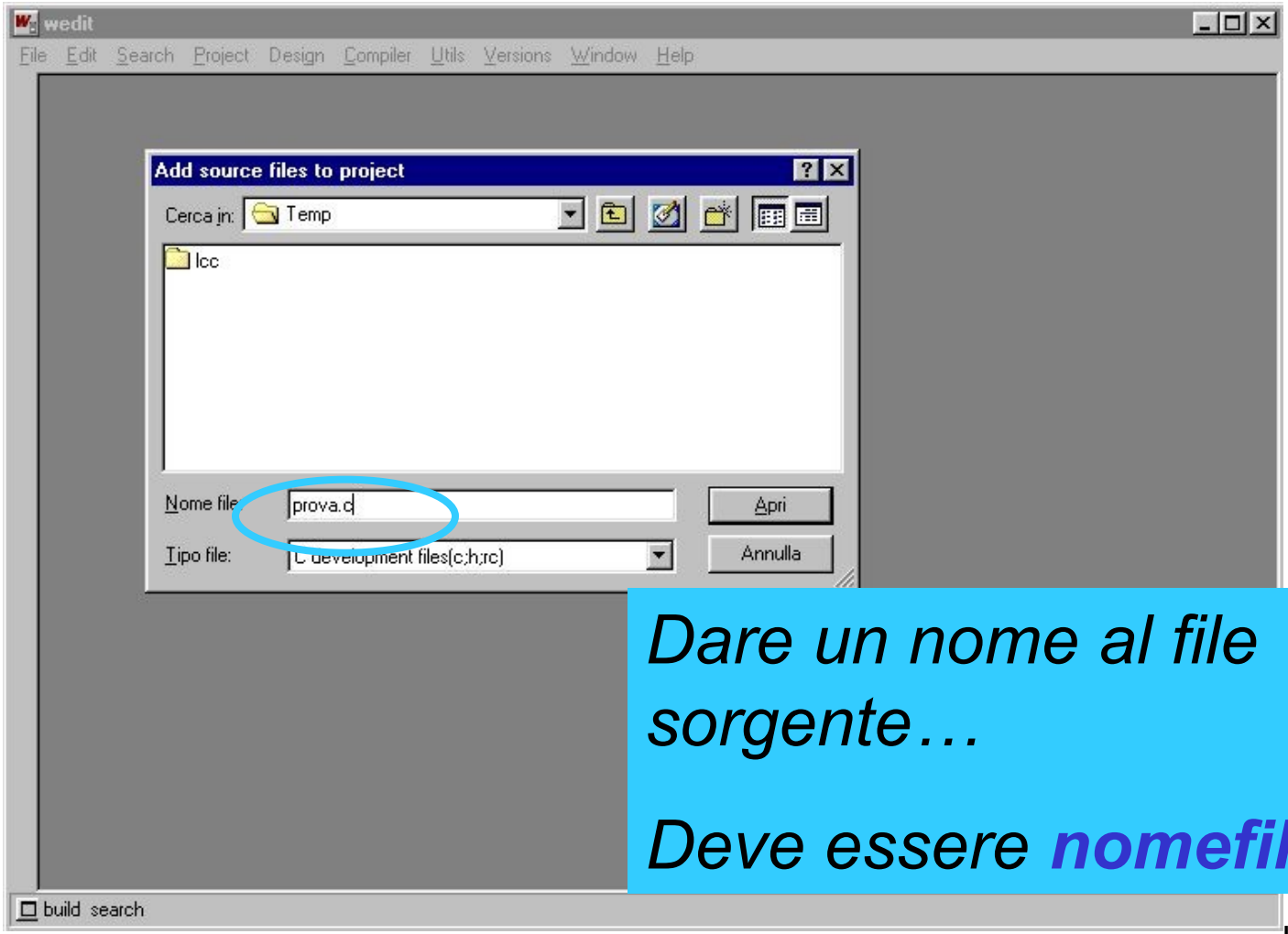


Selezionare la directory di lavoro…

Progetti in LCC



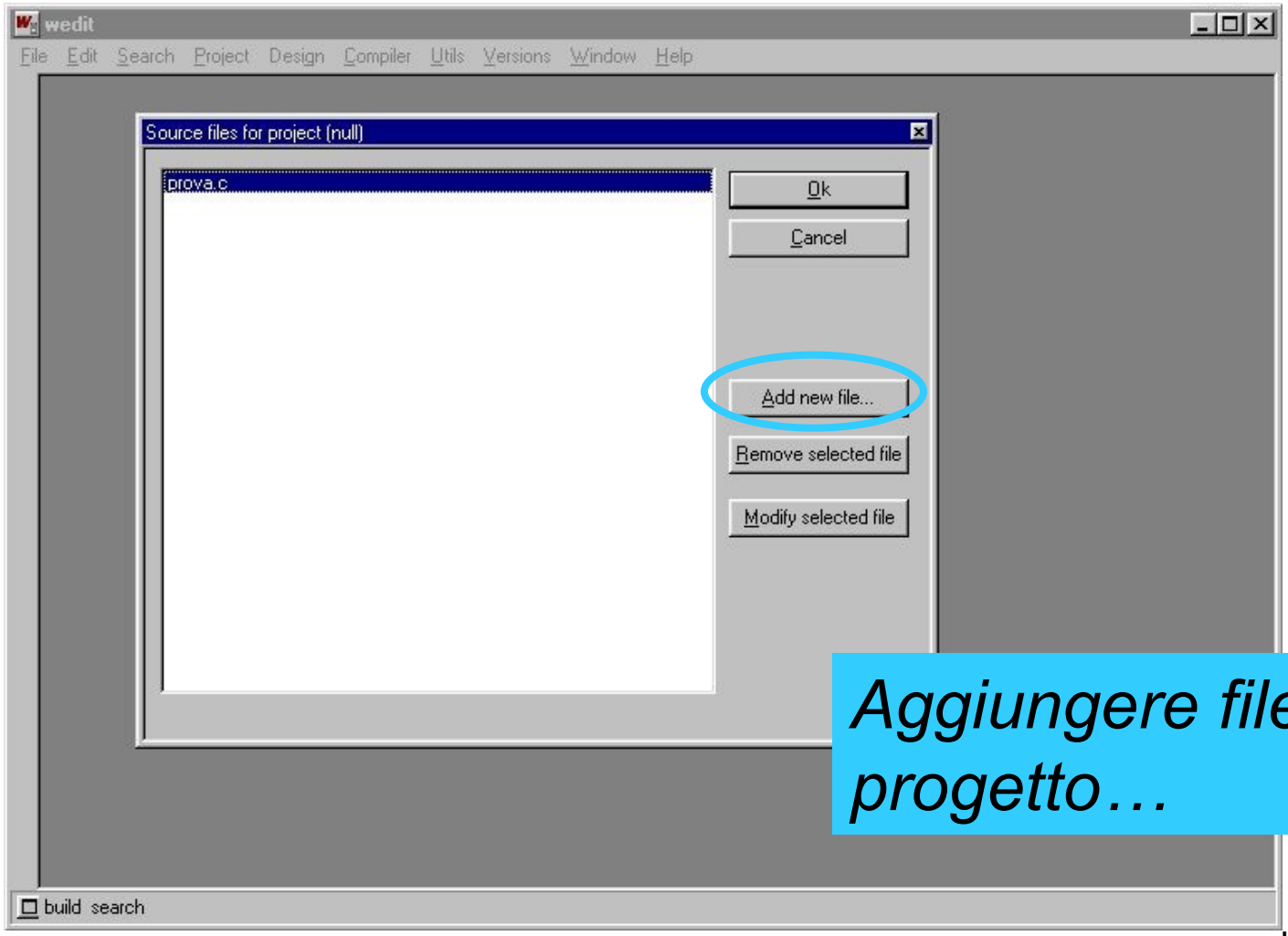
Progetti in LCC



Dare un nome al file sorgente…

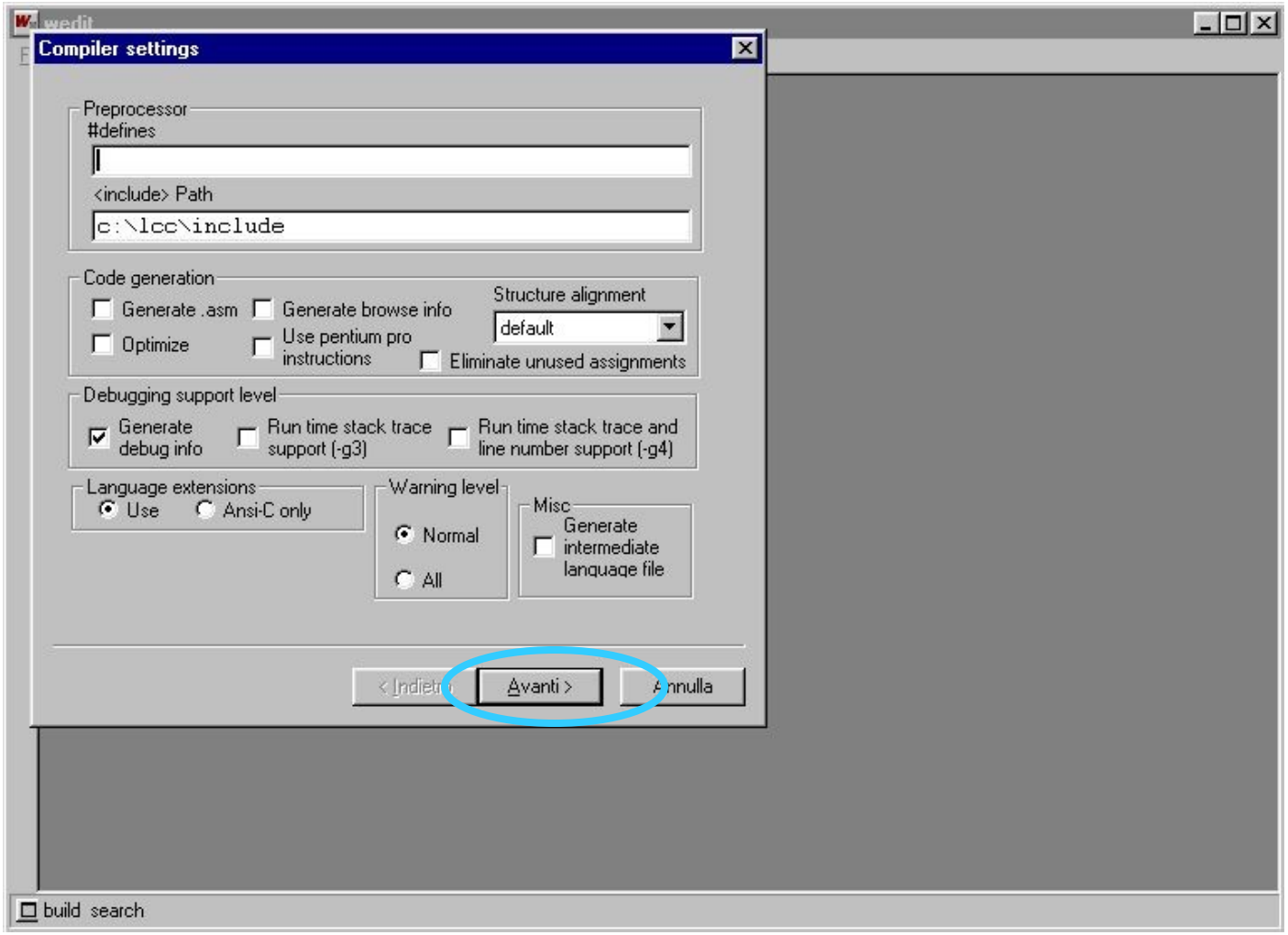
Deve essere ***nomefile.c***

Progetti in LCC



Aggiungere file al progetto…

Progetti in LCC

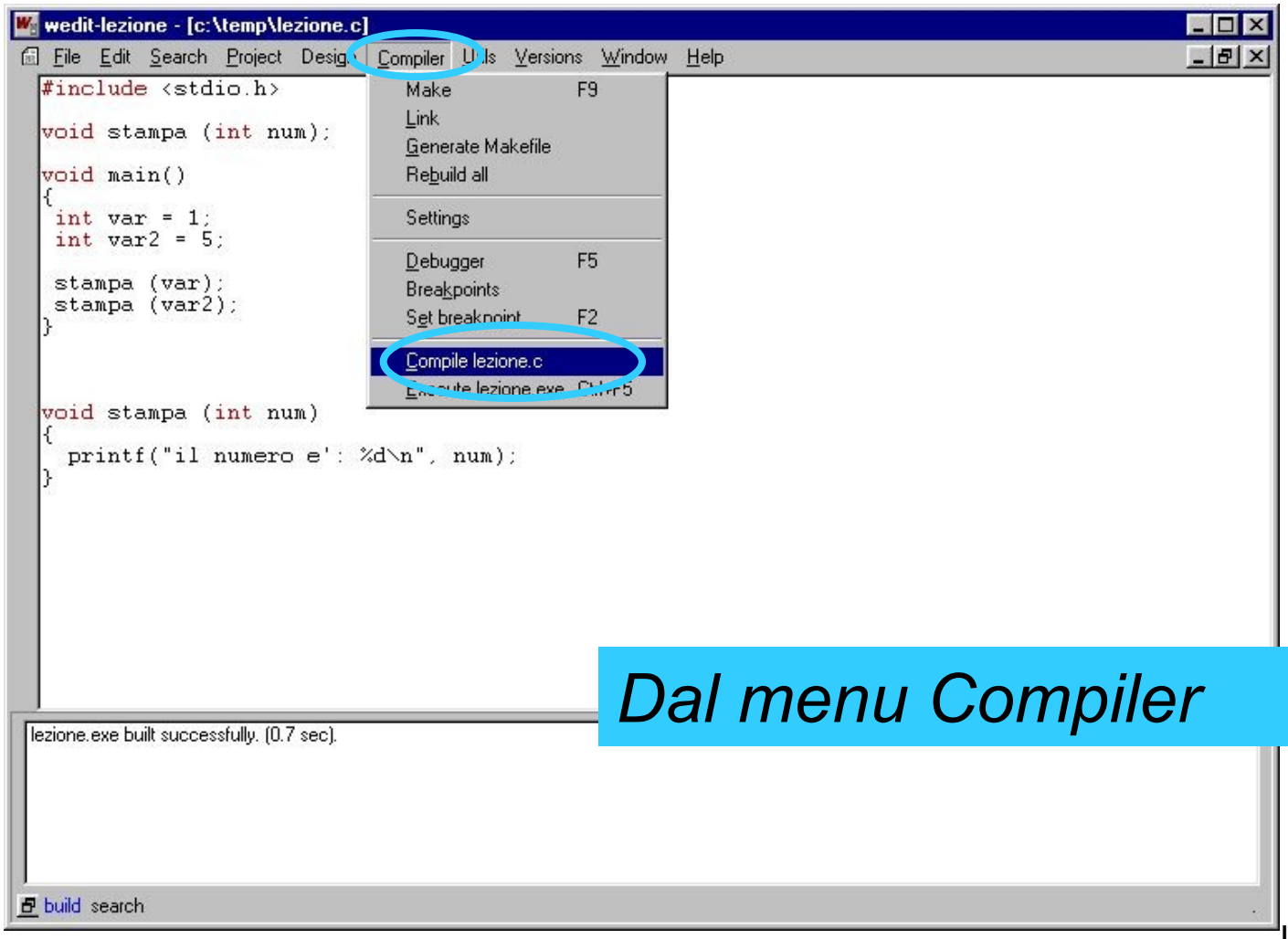


Editare e Salvare

Editare il file sorgente.

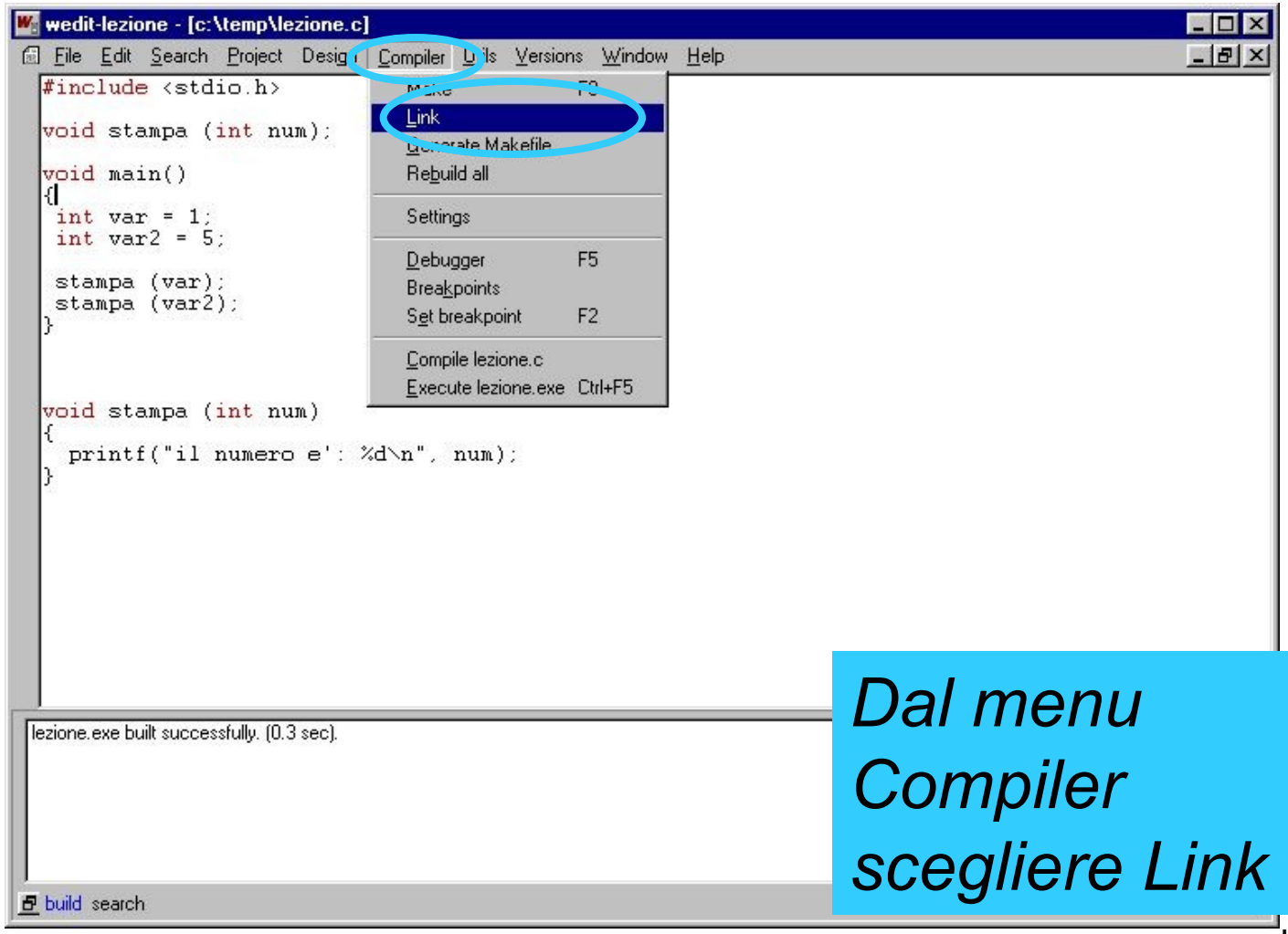
Salvare tramite Save di menu File

Compilare



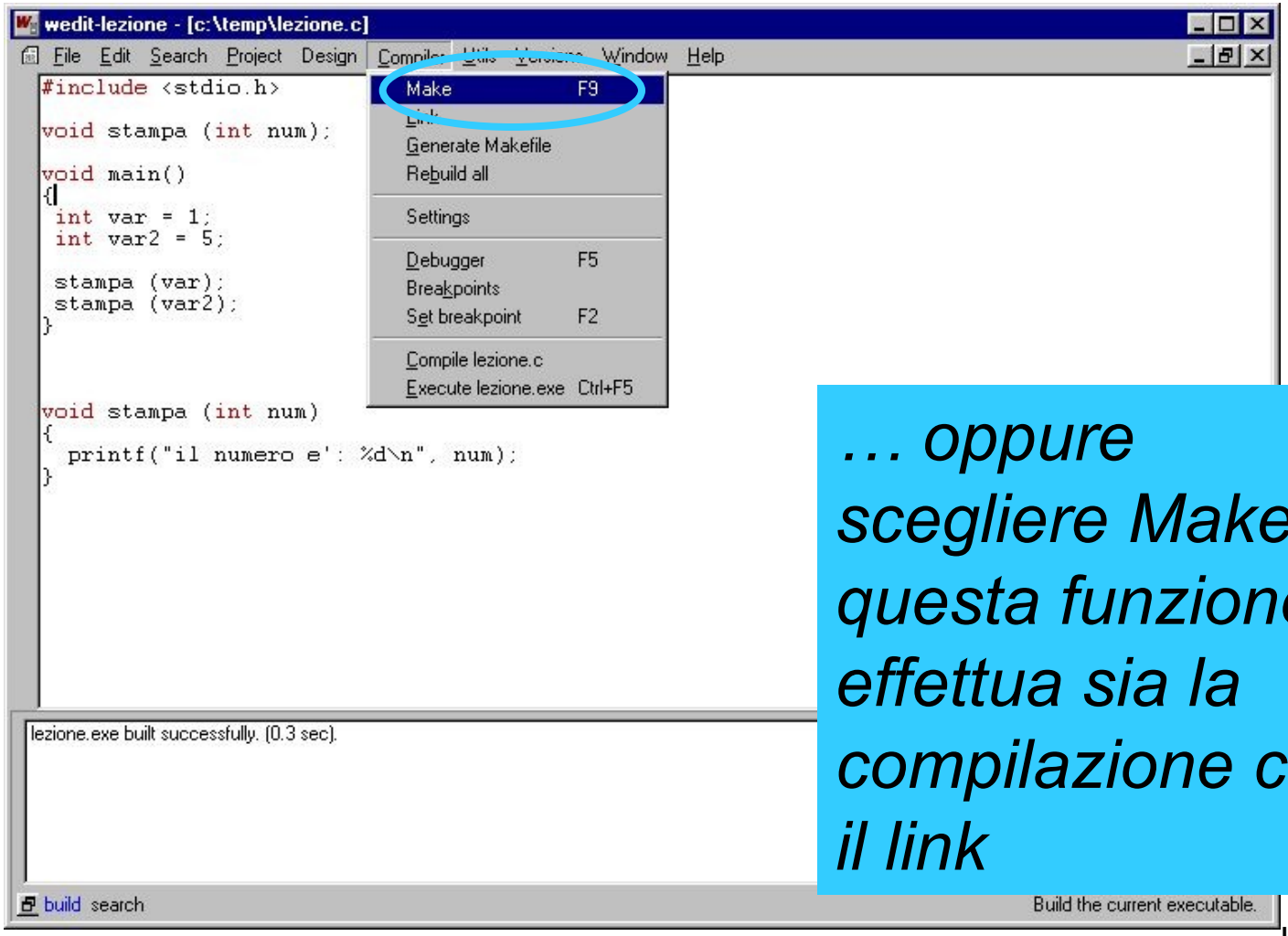
Dal menu Compiler

Link



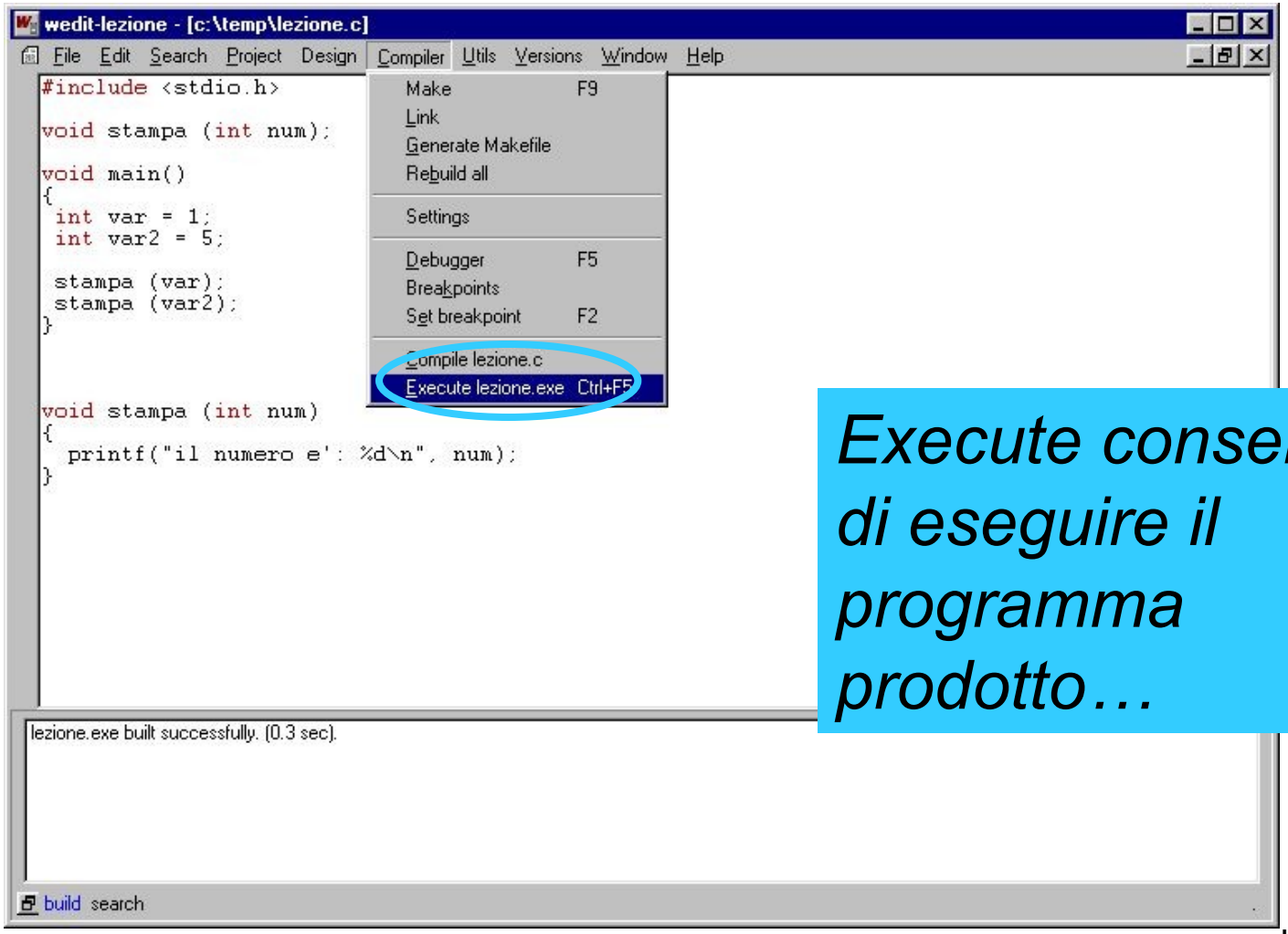
Dal menu Compiler scegliere Link

Make



… oppure scegliere Make: questa funzione effettua sia la compilazione che il link

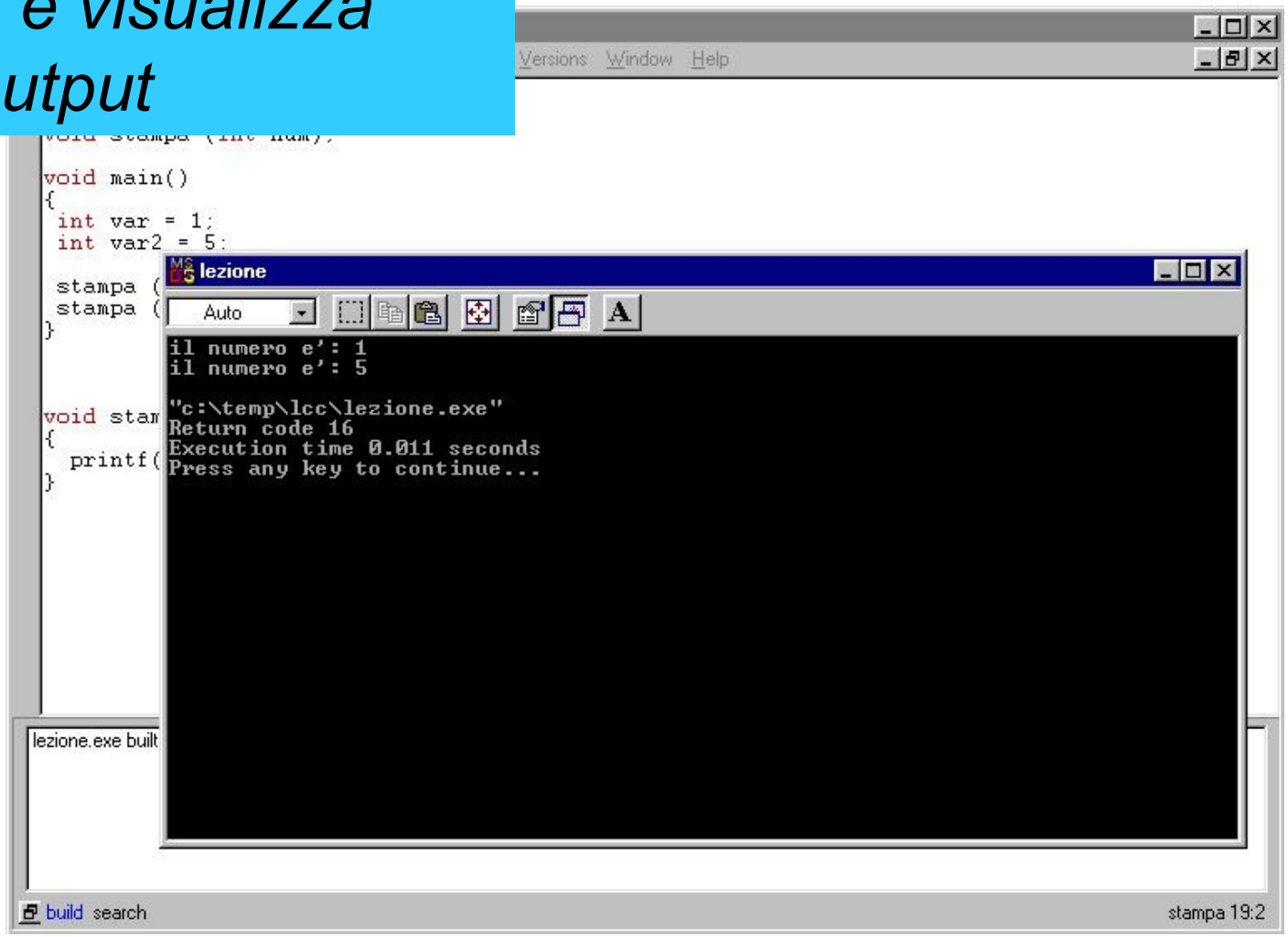
Execute



Execute consente di eseguire il programma prodotto…

Execute

… e visualizza l'output

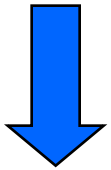


Il Debugger

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

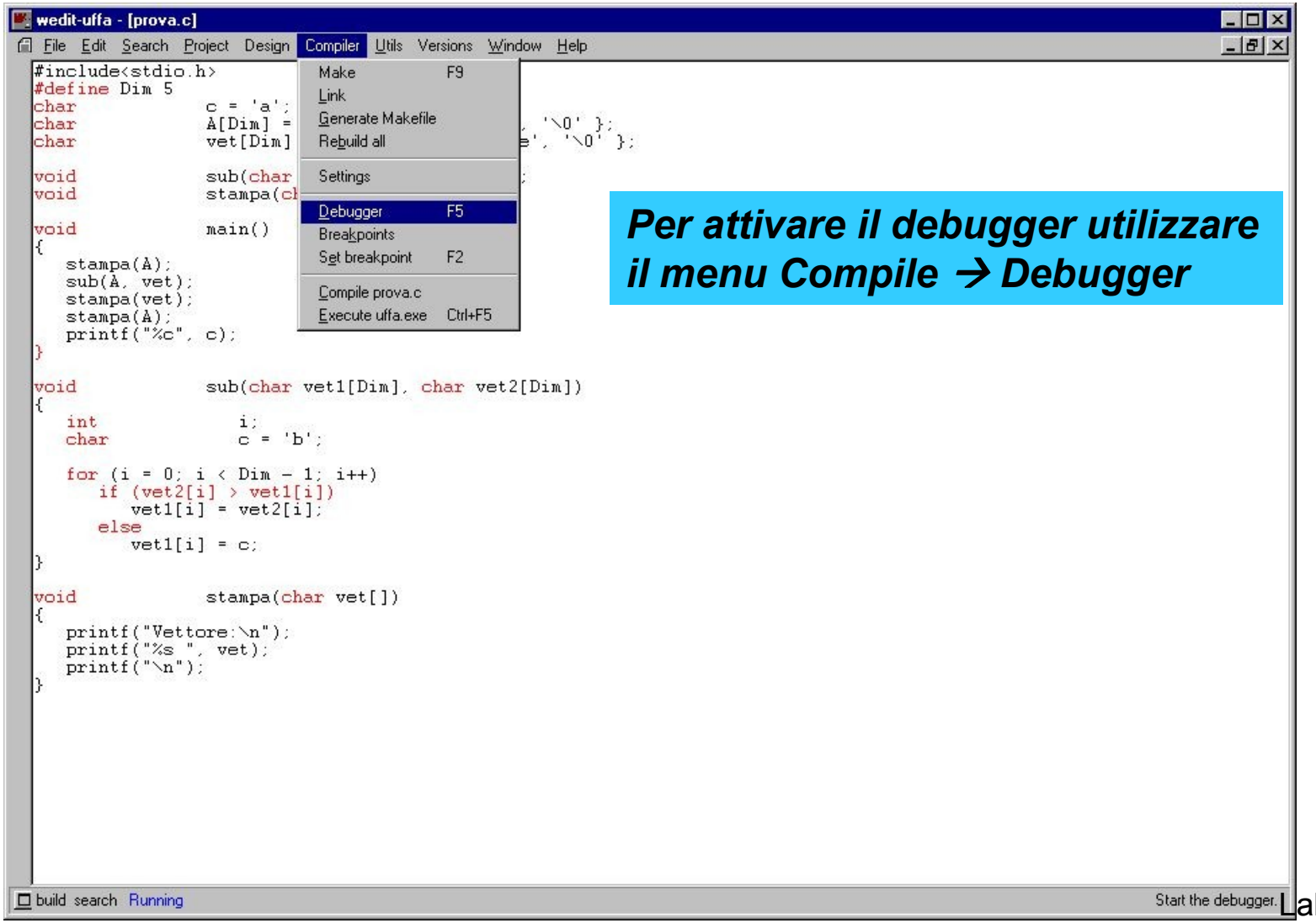
Debugger

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

- ***riga per riga***
  - **entrando anche dentro alle funzioni chiamate**
  - **oppure considerando le chiamate di funzione come una singola operazione**

- **oppure *inserendo breakpoints***

Debugger



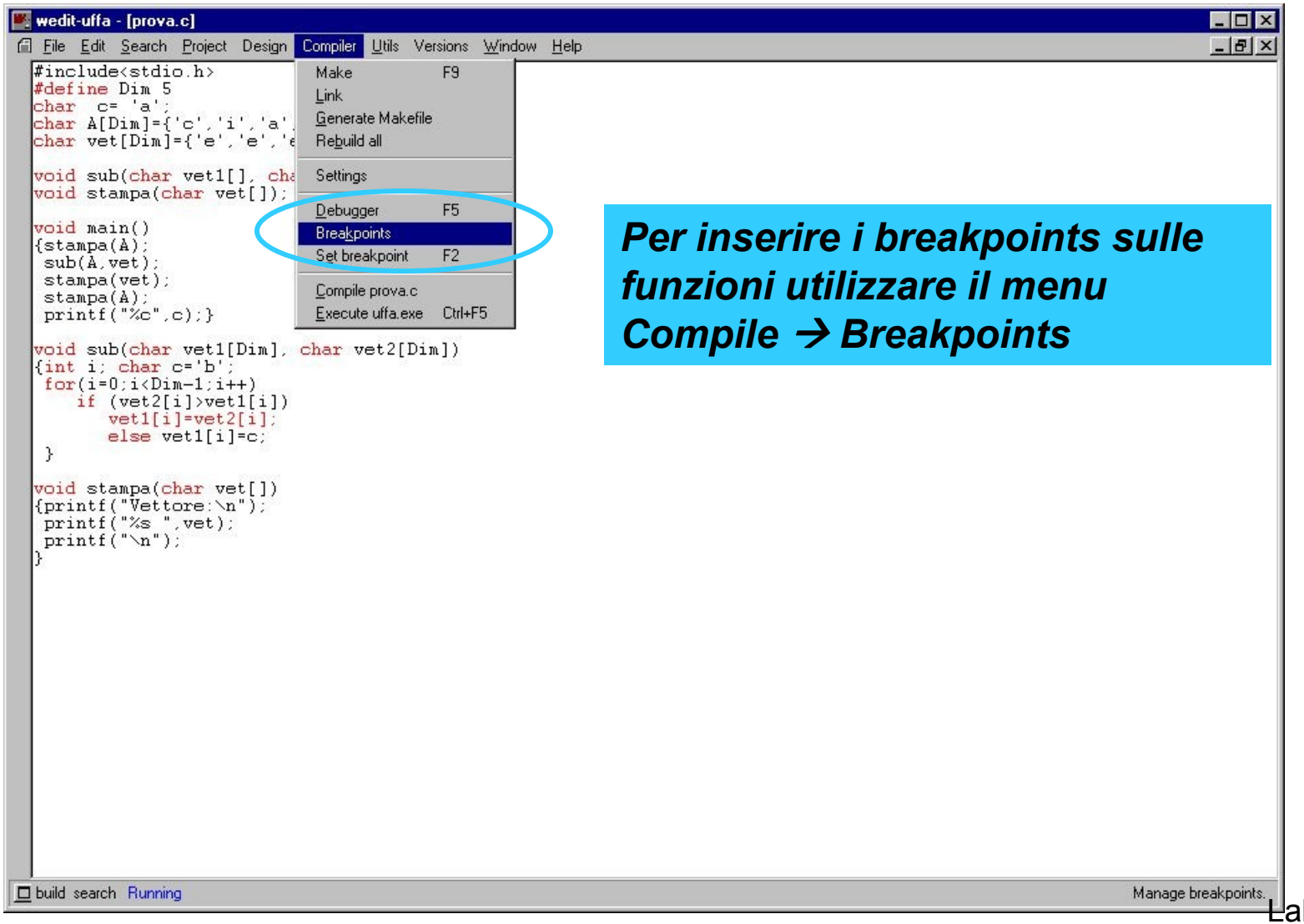
Per attivare il debugger utilizzare il menu Compile → Debugger

Fase di Debugging

- **Prima di iniziare la sessione di debugging e' possibile inserire i cosiddetti *breakpoints***
  - ***punti di interruzione nell'esecuzione del programma in cui il debugger fornisce una "fotografia" dello stato delle variabili***

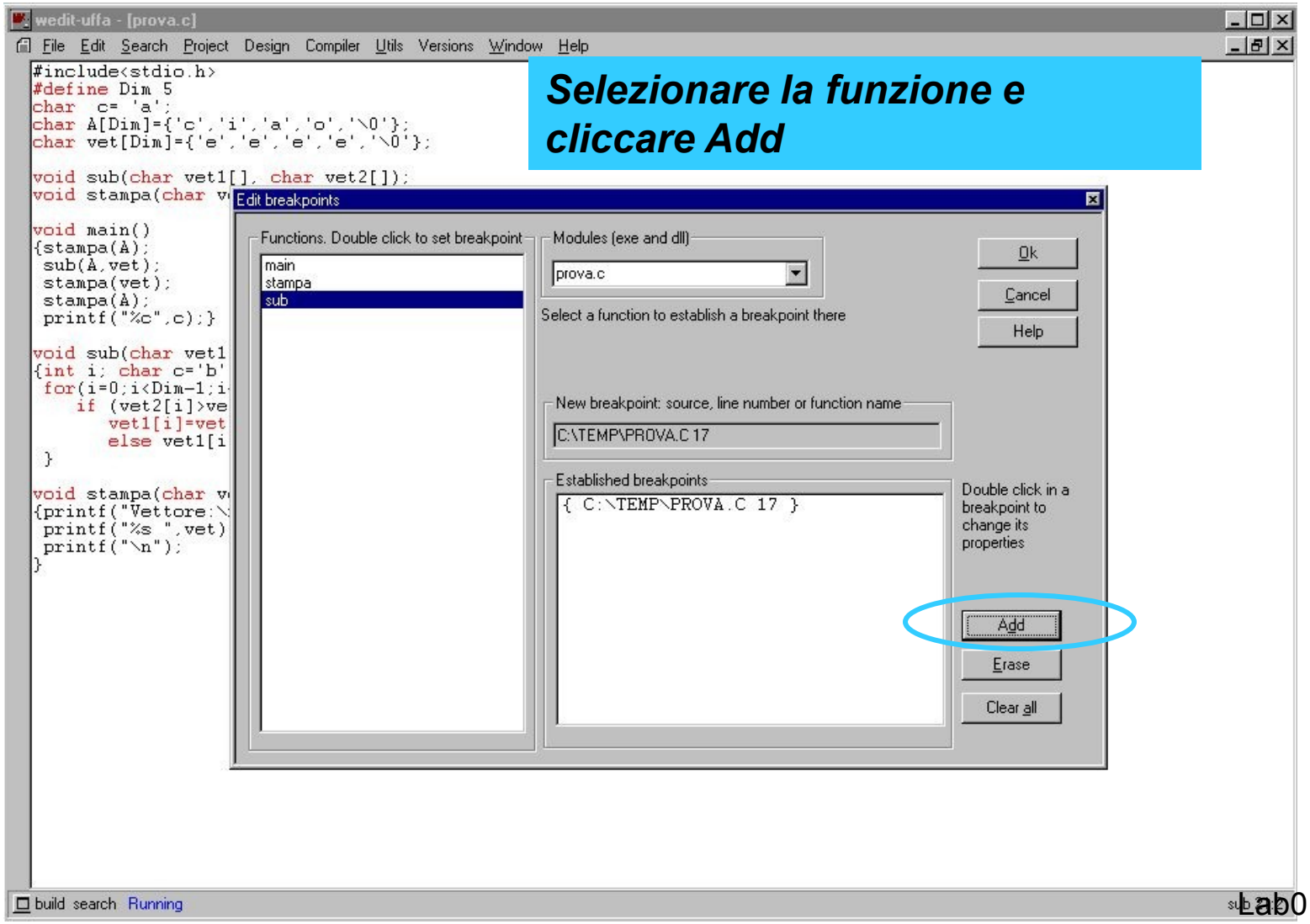
- **Due modi per inserirli:**
  - ***sulle funzioni***
  - ***sulle singole istruzioni***

Debugger

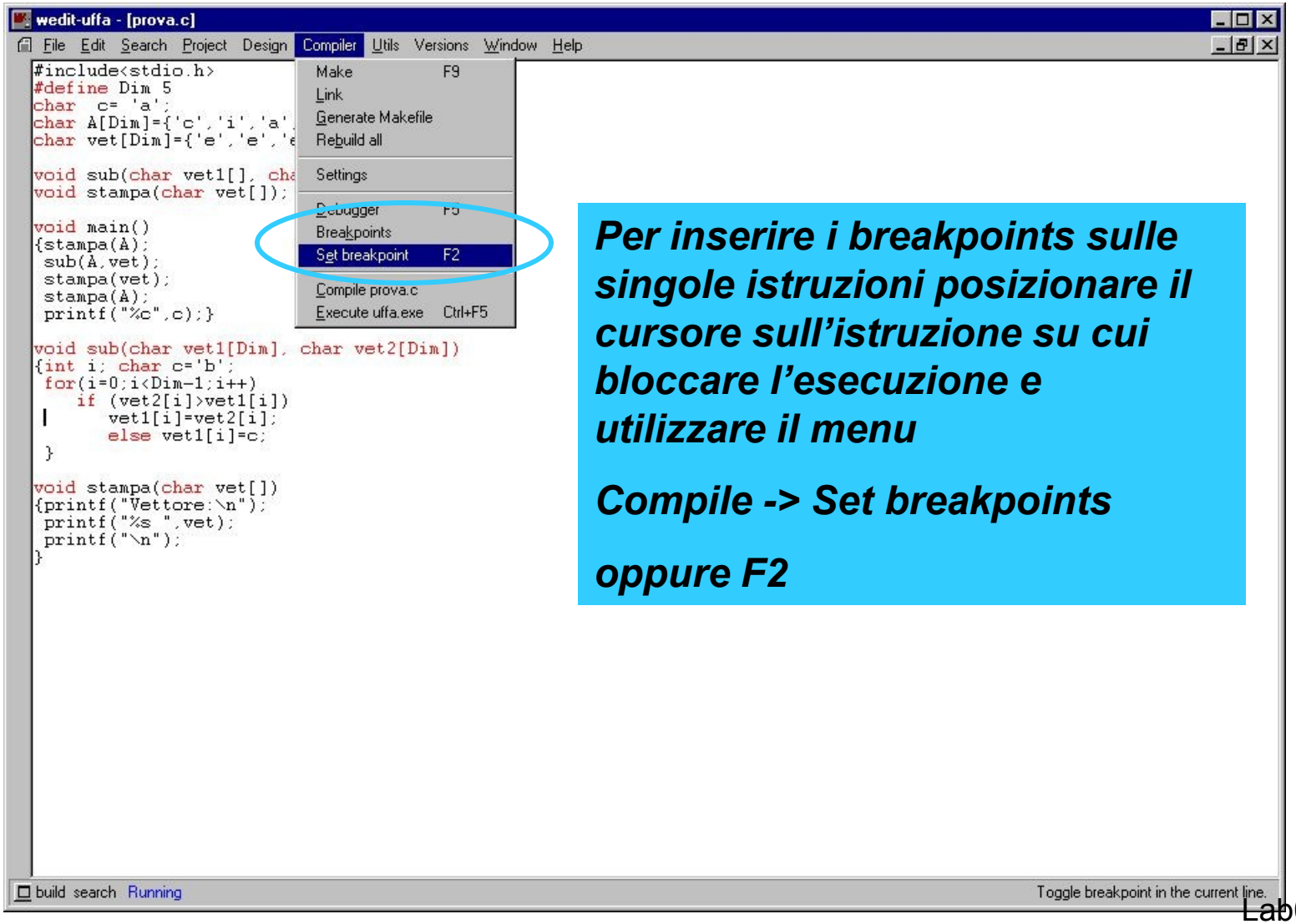


Per inserire i breakpoints sulle funzioni utilizzare il menu Compile → Breakpoints

Debugger


wedit-uffa - [prova.c]
File Edit Search Project Design Compiler Utils Versions Window Help
#include<stdio.h>
#define Dim 5
char c= 'a';
char A[Dim]={'c','i','a','o','\0'};
char vet[Dim]={'e','e','e','e','\0'};
void sub(char vet1[], char vet2[]);
void stampa(char v
void main()
{stampa(A);
sub(A,vet);
stampa(vet);
stampa(A);
printf("%c",c);}
void sub(char vet1
{int i; char c='b'
for(i=0;i<Dim-1;i
if (vet2[i]>ve
vet1[i]=vet
else vet1[i
}
void stampa(char v
{printf("Vettore:\
printf("%s ",vet)
printf("\n");
}
build search Running
Selezionare la funzione e cliccare Add
Edit breakpoints
Functions. Double click to set breakpoint
main
stampa
sub
Modules (exe and dll)
prova.c
Select a function to establish a breakpoint there
Ok
Cancel
Help
New breakpoint: source, line number or function name
C:\TEMP\PROVA.C 17
Established breakpoints
{ C:\TEMP\PROVA.C 17 }
Double click in a breakpoint to change its properties
Add
Erase
Clear all

Debugger

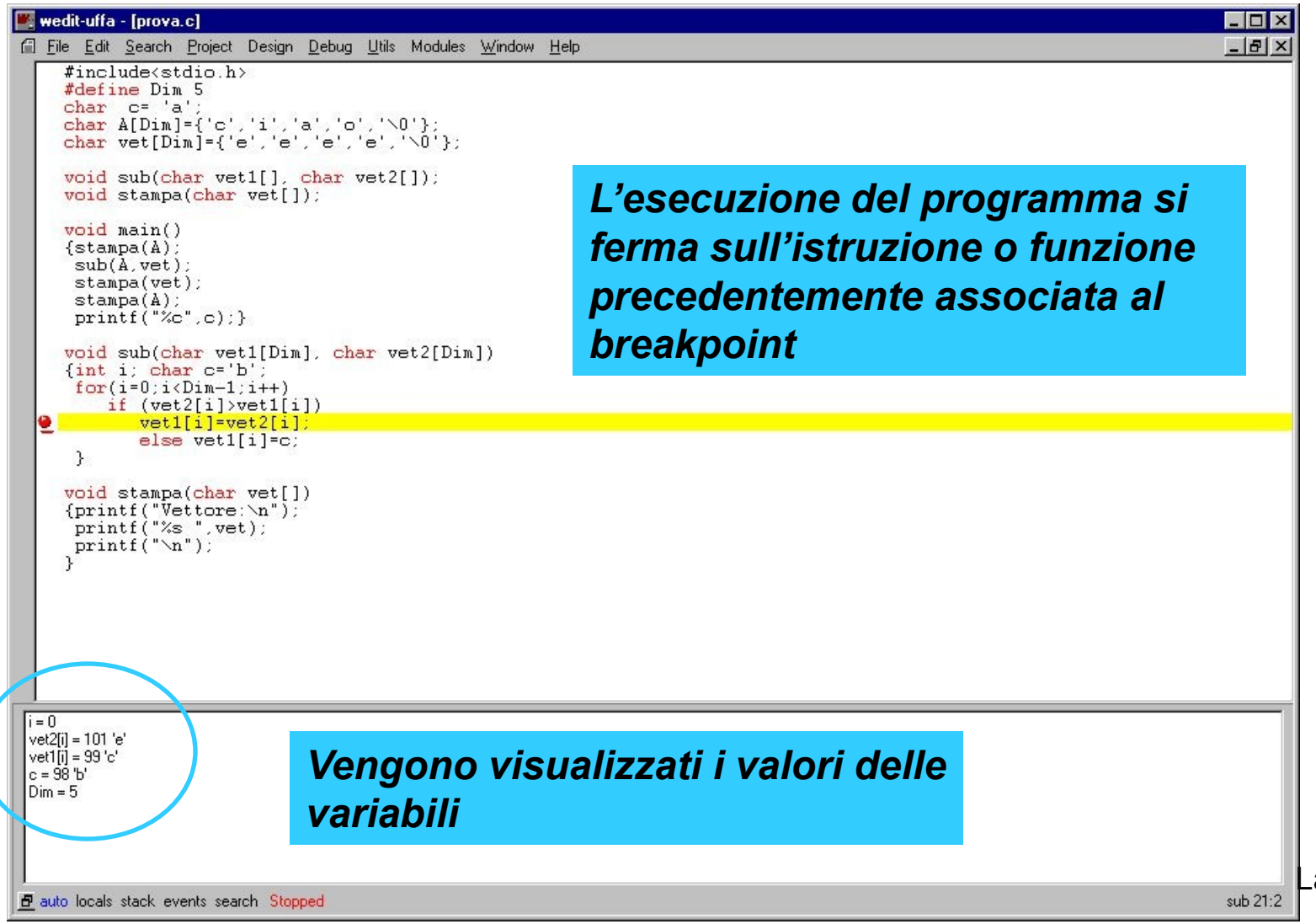


Per inserire i breakpoints sulle singole istruzioni posizionare il cursore sull'istruzione su cui bloccare l'esecuzione e utilizzare il menu

Compile -> Set breakpoints

oppure F2

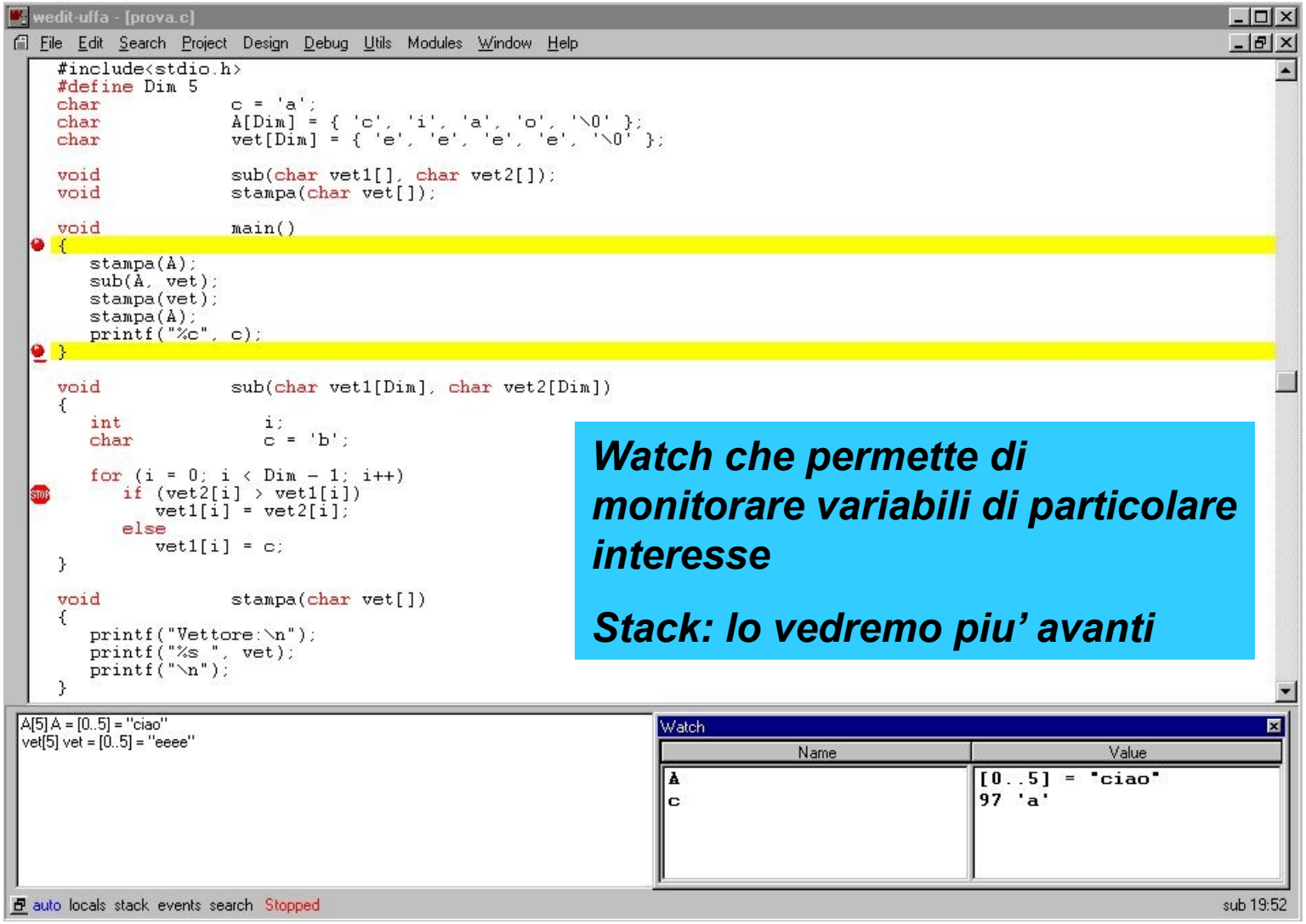
Debugger


wedit-uffa - [prova.c]
File Edit Search Project Design Debug Utils Modules Window Help
#include<stdio.h>
#define Dim 5
char c= 'a';
char A[Dim]={'c','i','a','o','\0'};
char vet[Dim]={'e','e','e','e','\0'};
void sub(char vet1[], char vet2[]);
void stampa(char vet[]);
void main()
{stampa(A);
sub(A,vet);
stampa(vet);
stampa(A);
printf("%c",c);}
void sub(char vet1[Dim], char vet2[Dim])
{int i; char c='b';
for(i=0;i<Dim-1;i++)
if (vet2[i]>vet1[i])
vet1[i]=vet2[i];
else vet1[i]=c;
}
void stampa(char vet[])
{printf("Vettore:\n");
printf("%s ",vet);
printf("\n");
}
i = 0
vet2[i] = 101 'e'
vet1[i] = 99 'c'
c = 98 'b'
Dim = 5
auto locals stack events search Stopped
sub 21:2
L'esecuzione del programma si ferma sull'istruzione o funzione precedentemente associata al breakpoint
Vengono visualizzati i valori delle variabili

Debugger: Come Procedere

- **Nel menu Debug che compare quando il Debugger e' attivo ci sono alcune voci importanti:**
  - **Execute: esegue il programma fino alla fine senza interruzioni**
  - **Step in: esegue passo passo le istruzioni di una funzione**
  - **Same level: esegue la funzione come istruzione singola**
  - **Run to cursor: permette di posizionare il cursore in una determinata posizione nel sorgente e esegue tutte le istruzioni fino ad arrestarsi al cursore.**

Debugger: Come Procedere



Watch che permette di monitorare variabili di particolare interesse

Stack: lo vedremo piu' avanti