

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 / file se più d'uno) che contengono il testo del programma (file *sorgente*)**

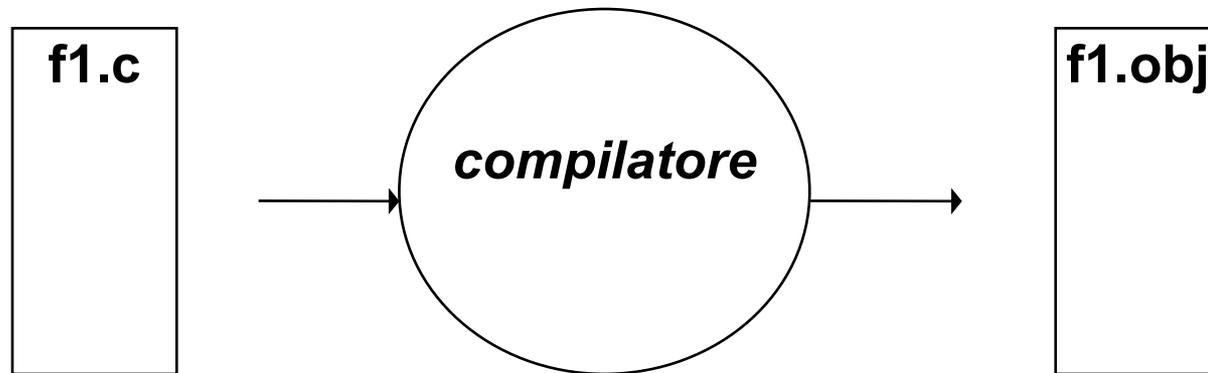
Il risultato sono uno o più file *oggetto*.

- **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**

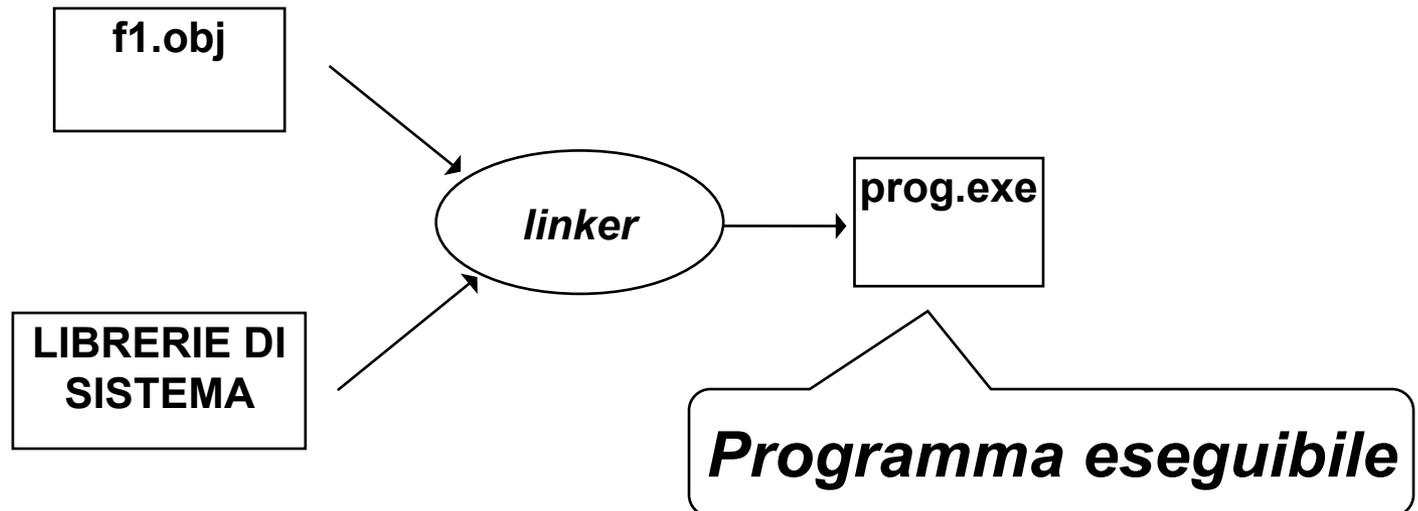


f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

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.***

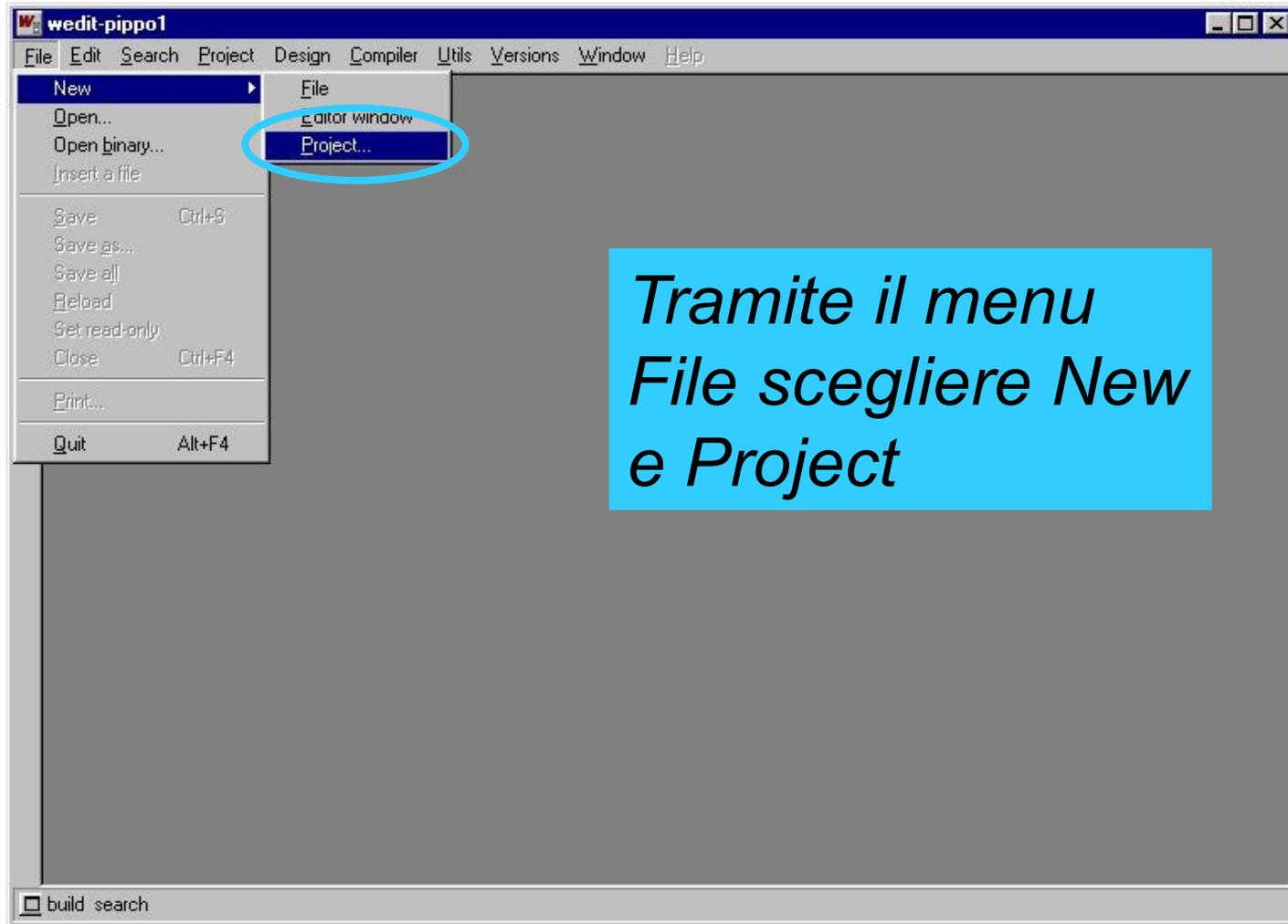
Progetti

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

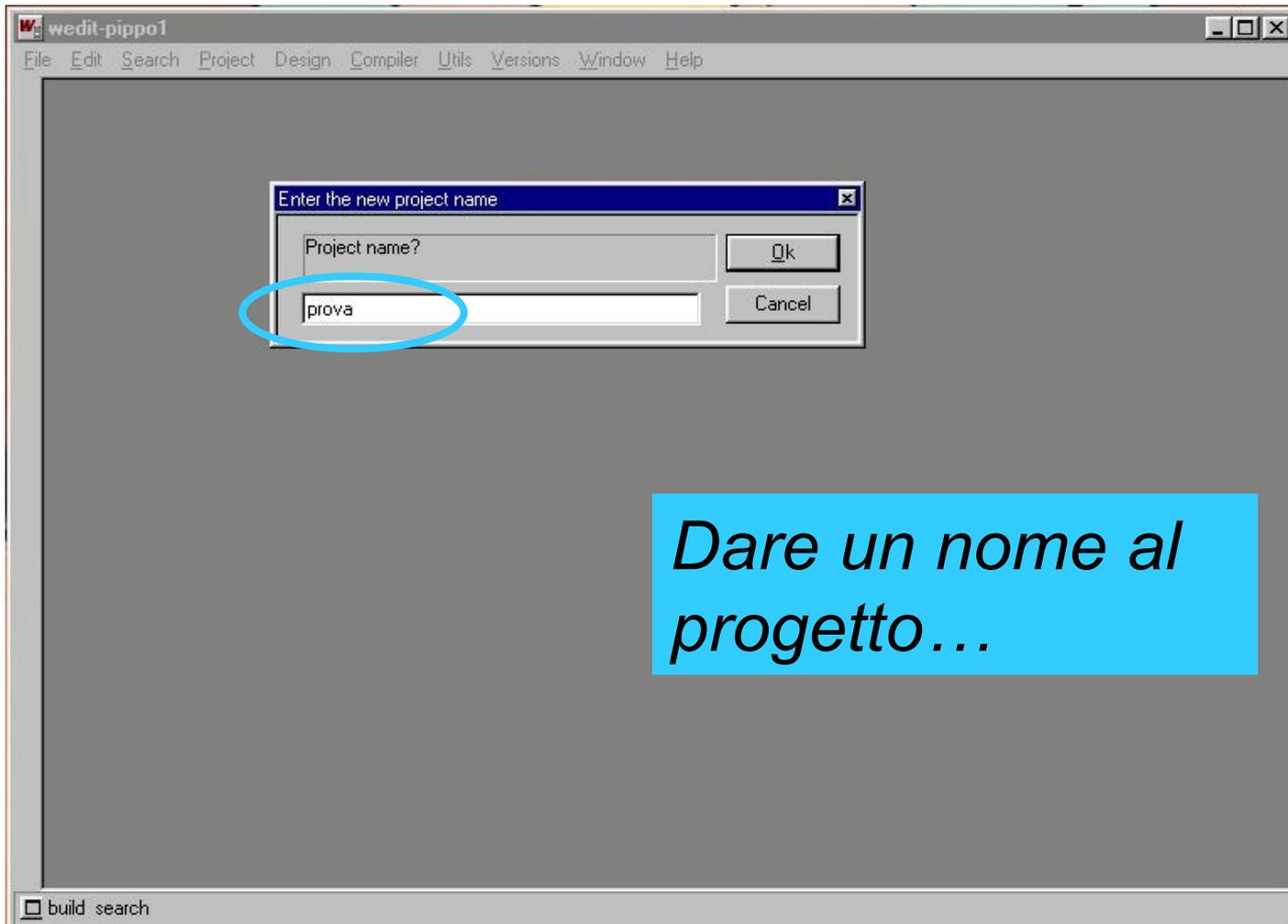
- **un contenitore concettuale (e fisico)**
- **che elenca i file sorgente in cui l'applicazione è strutturata**
- **ed eventualmente altre informazioni utili.**

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

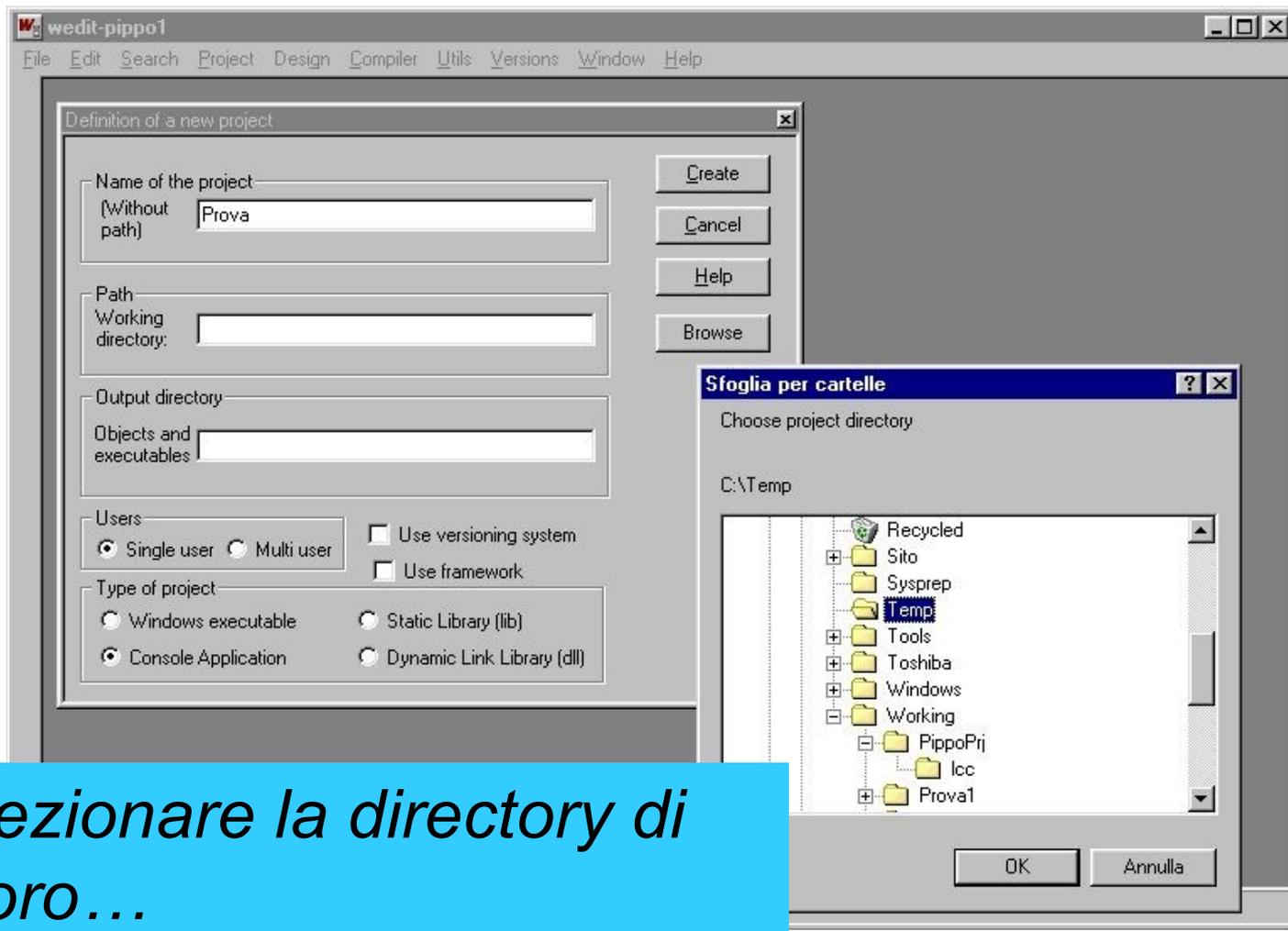
Progetti in LCC



Progetti in LCC

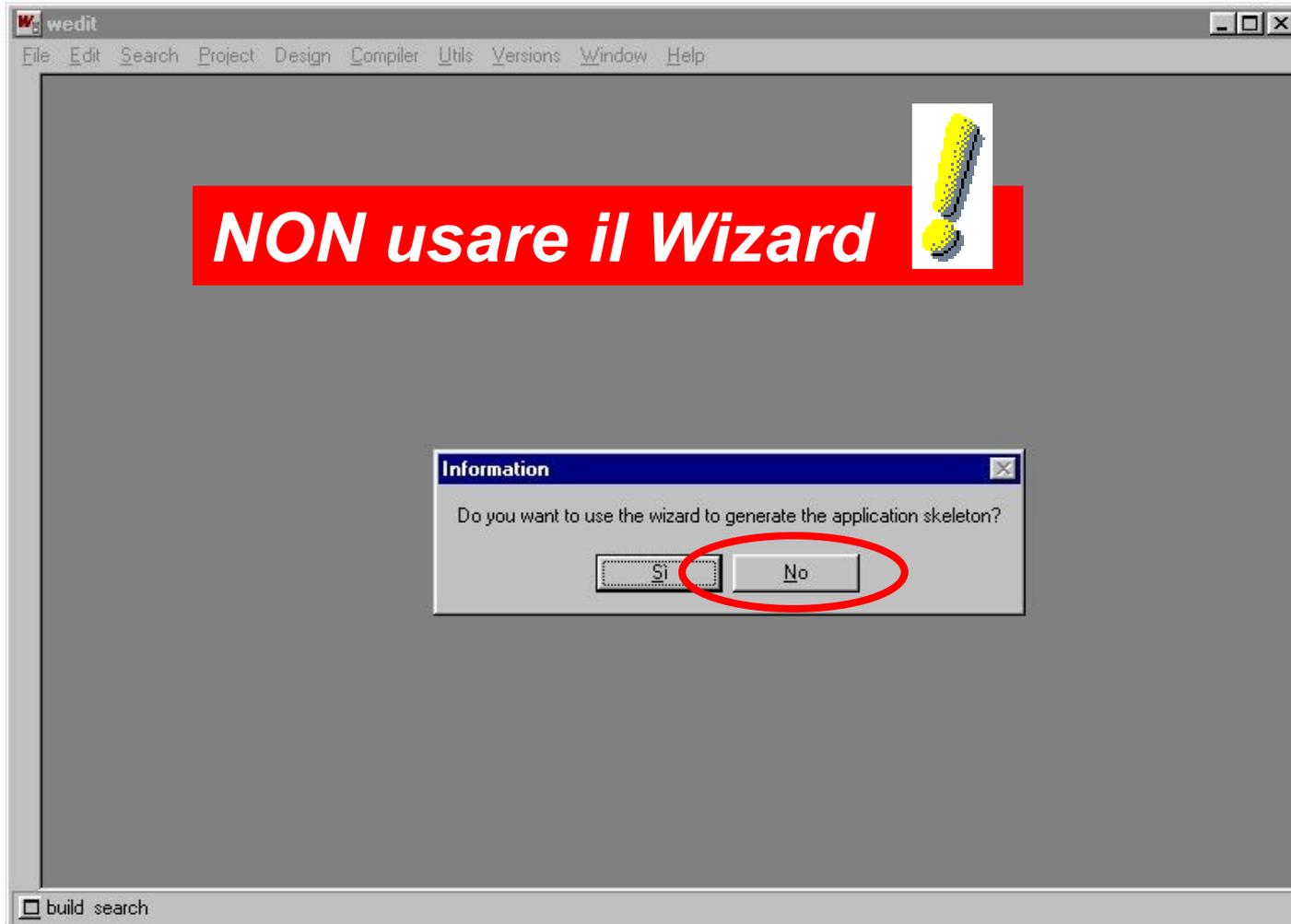


Progetti in LCC



Selezionare la directory di lavoro...

Progetti in LCC



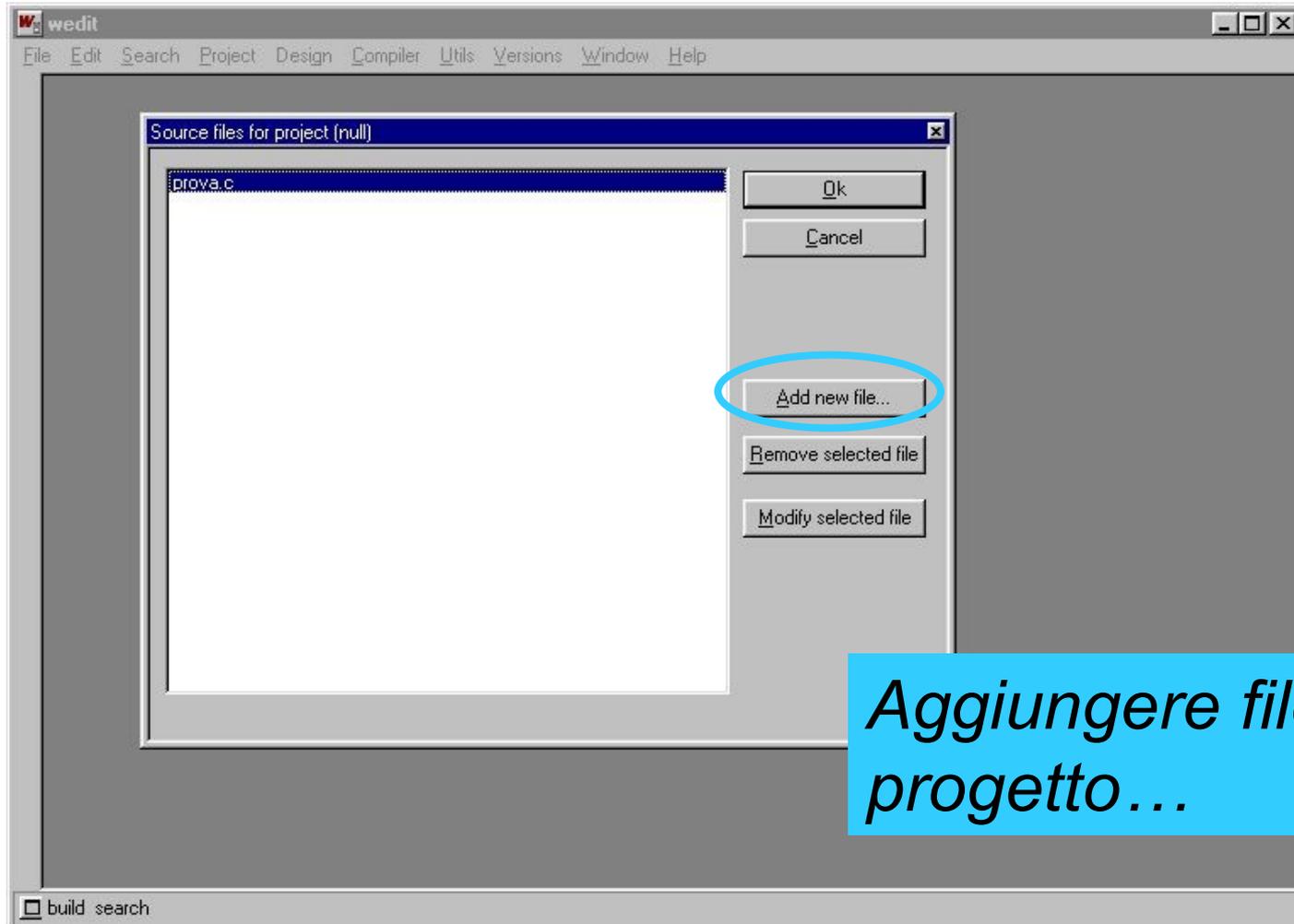
Progetti in LCC



Dare un nome al file sorgente...

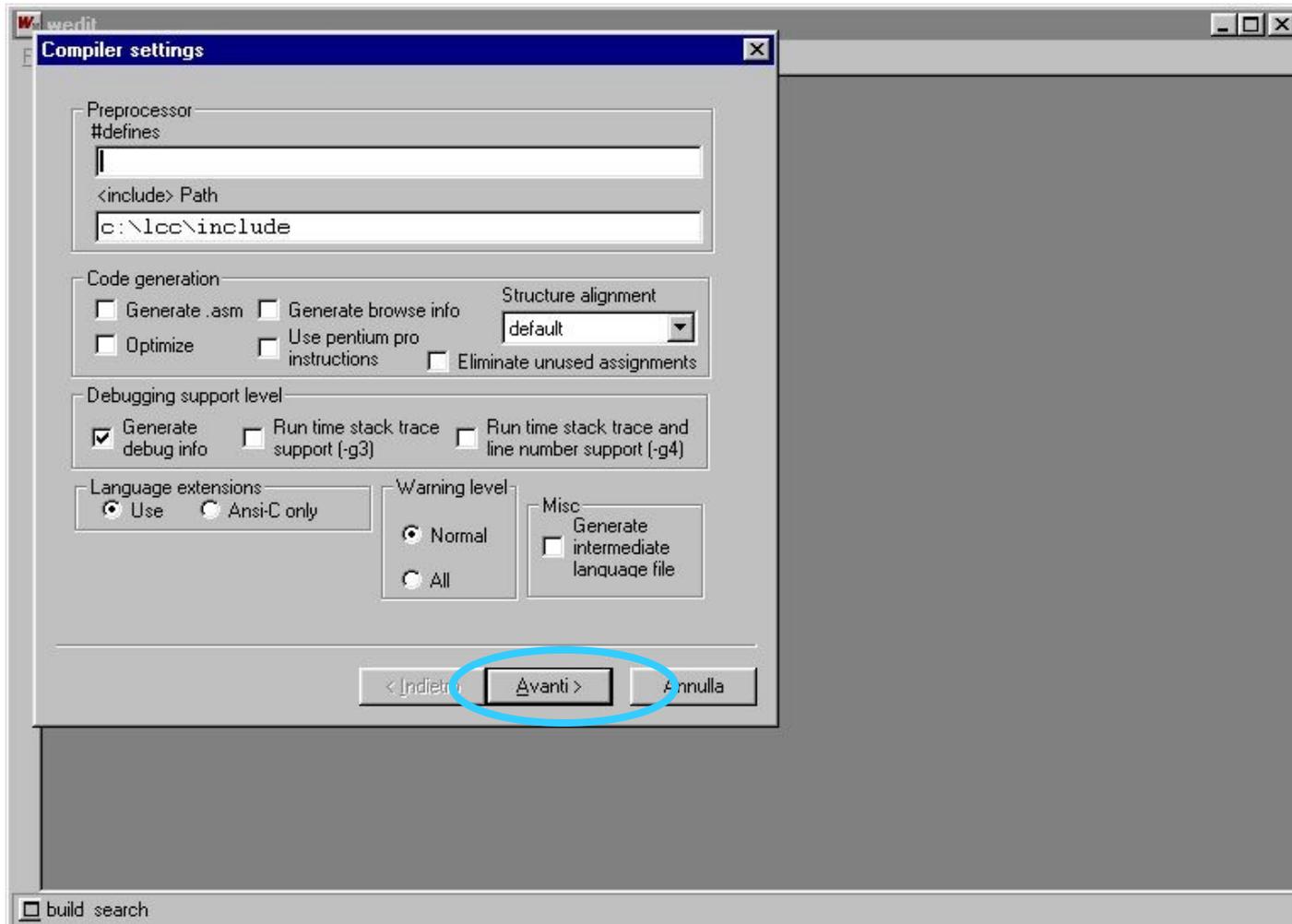
*Deve essere **nomefile.c***

Progetti in LCC

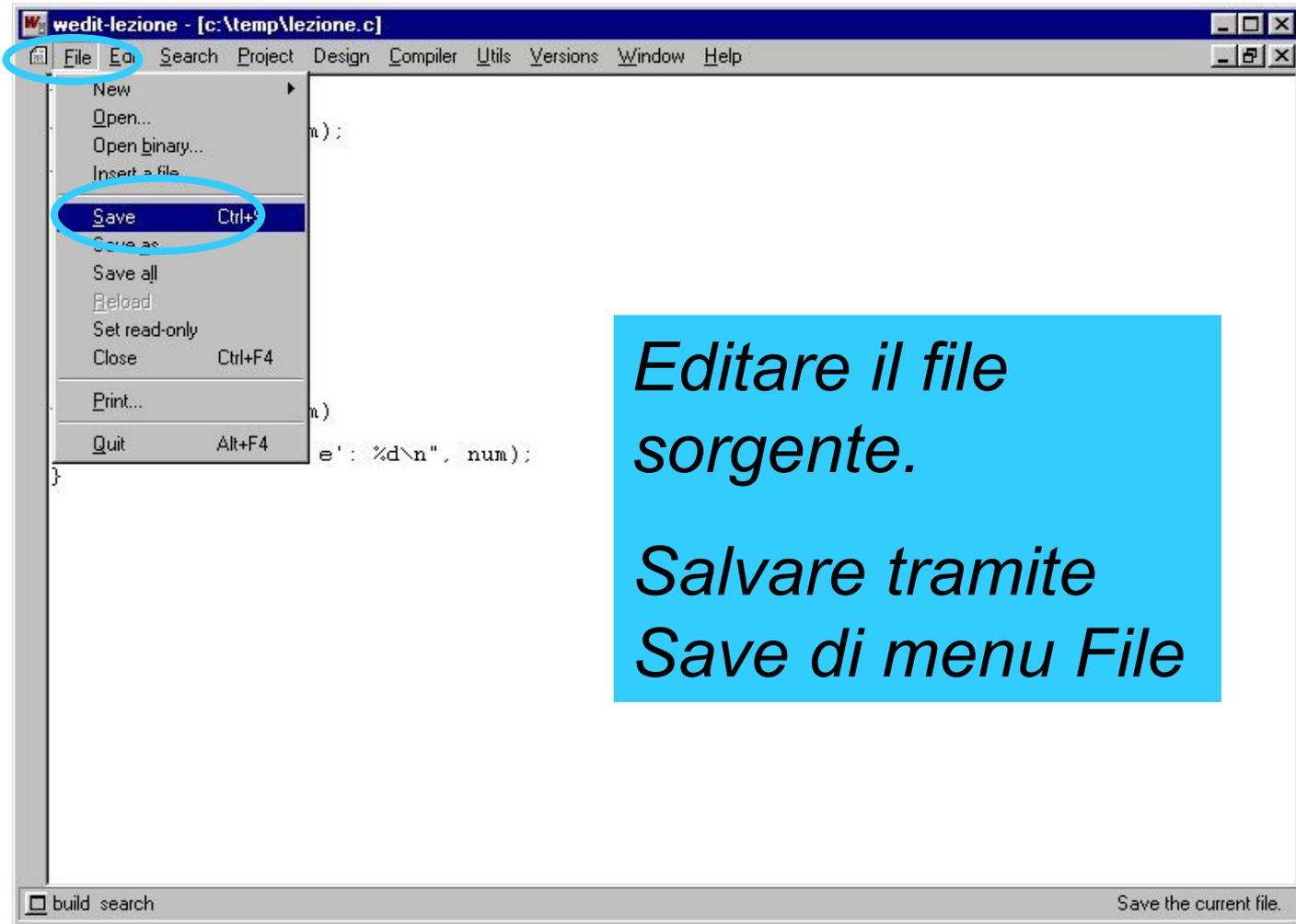


Aggiungere file al progetto...

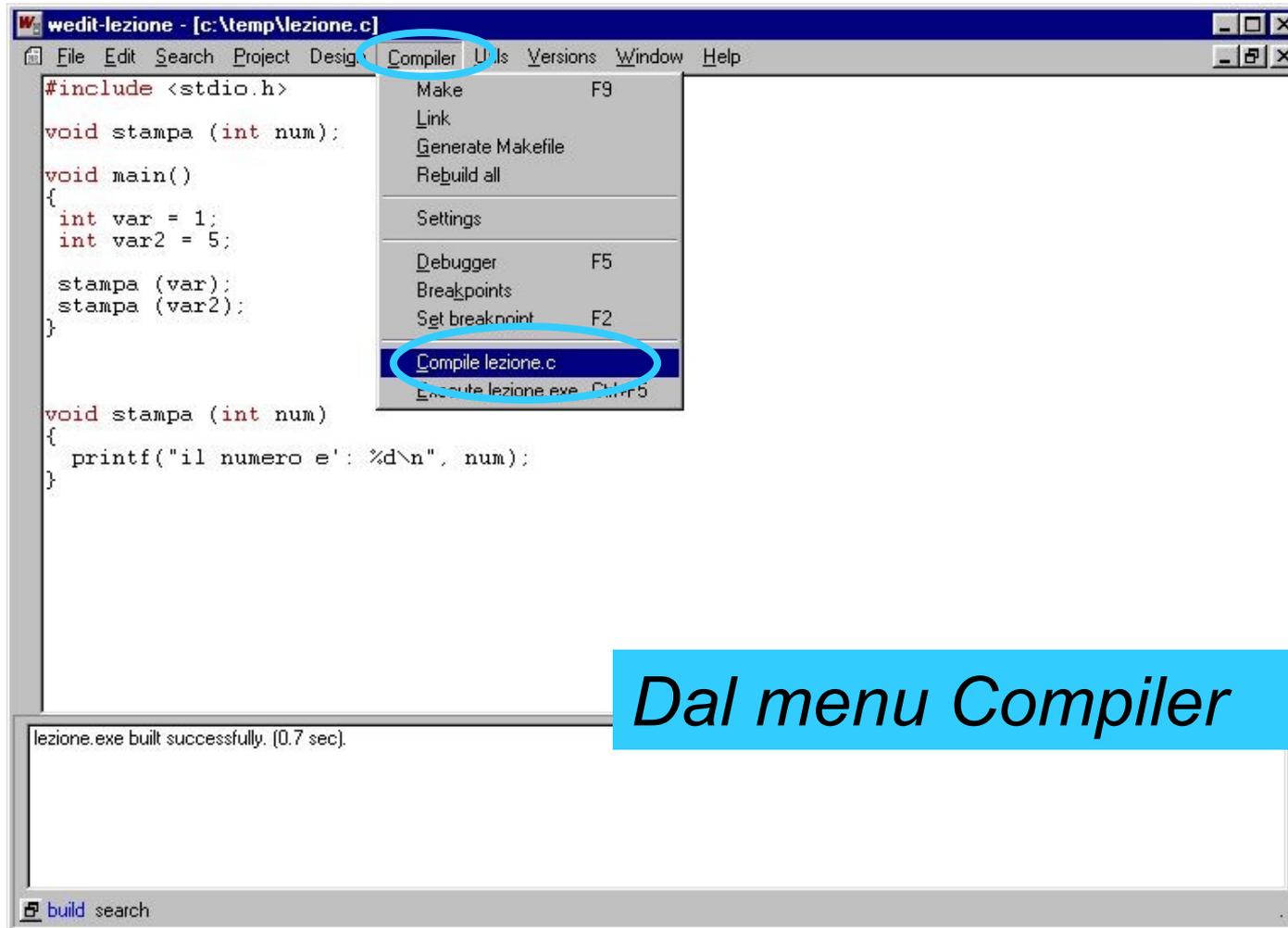
Progetti in LCC



Editare e Salvare

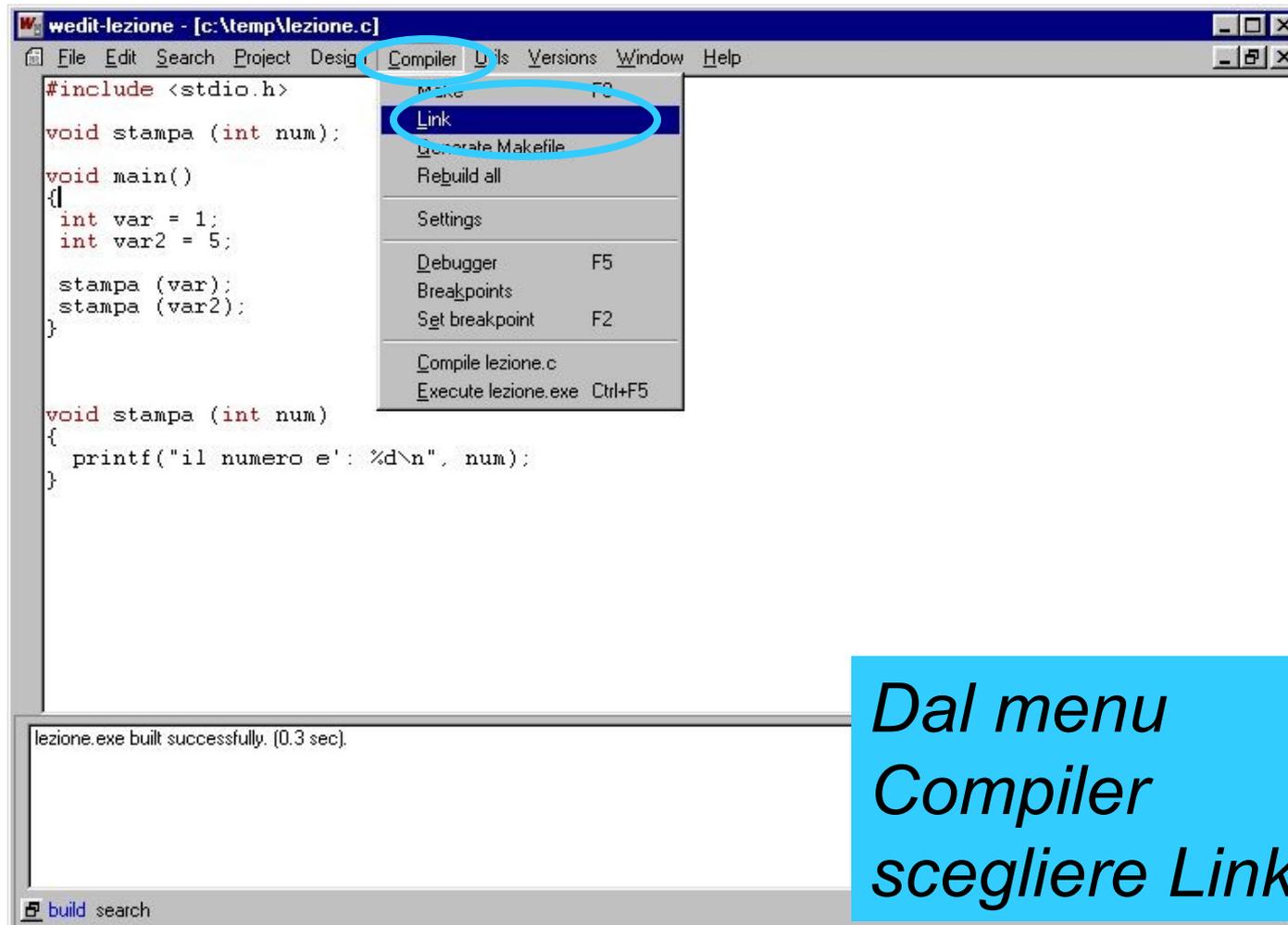


Compilare



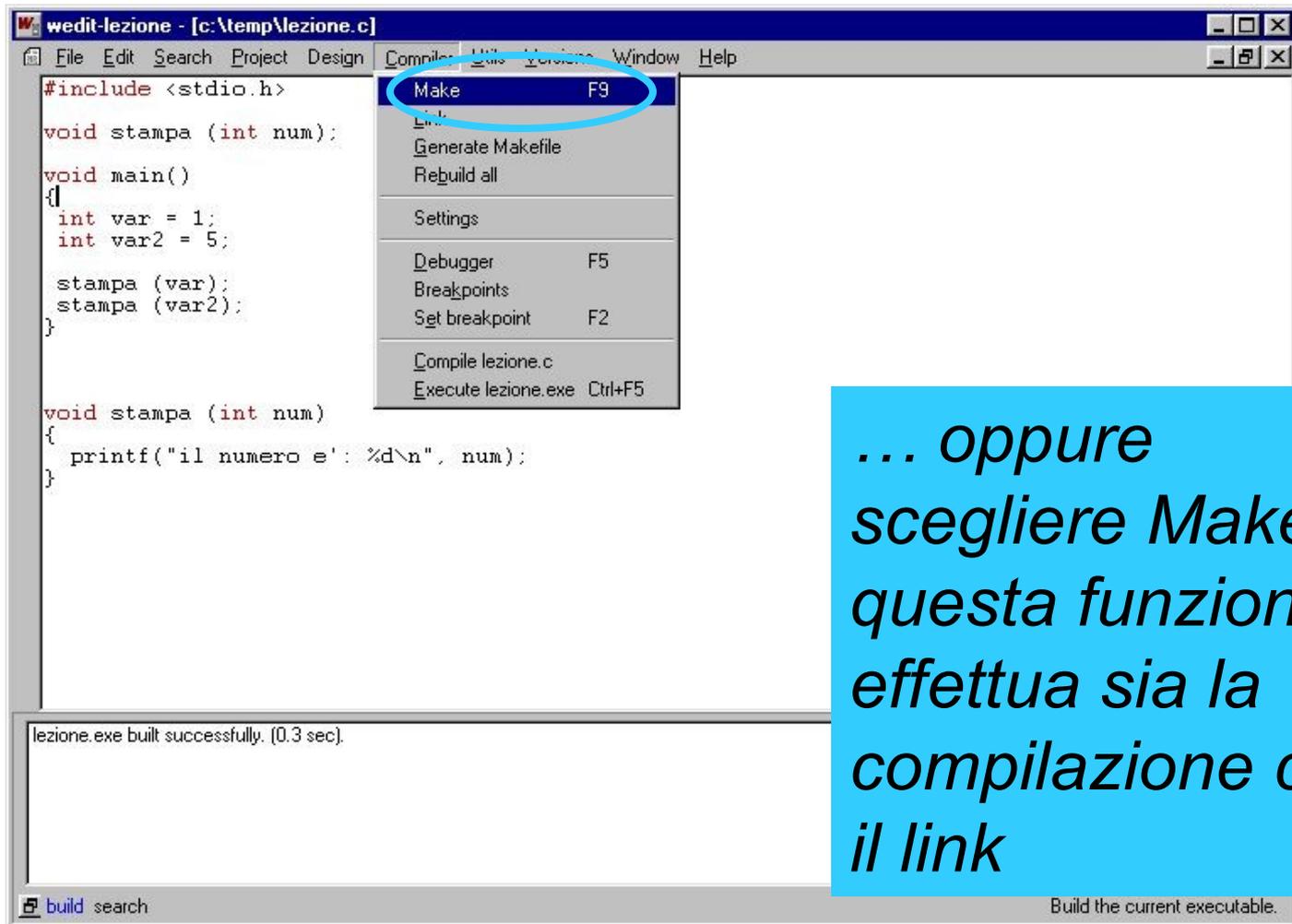
Dal menu Compiler

Link



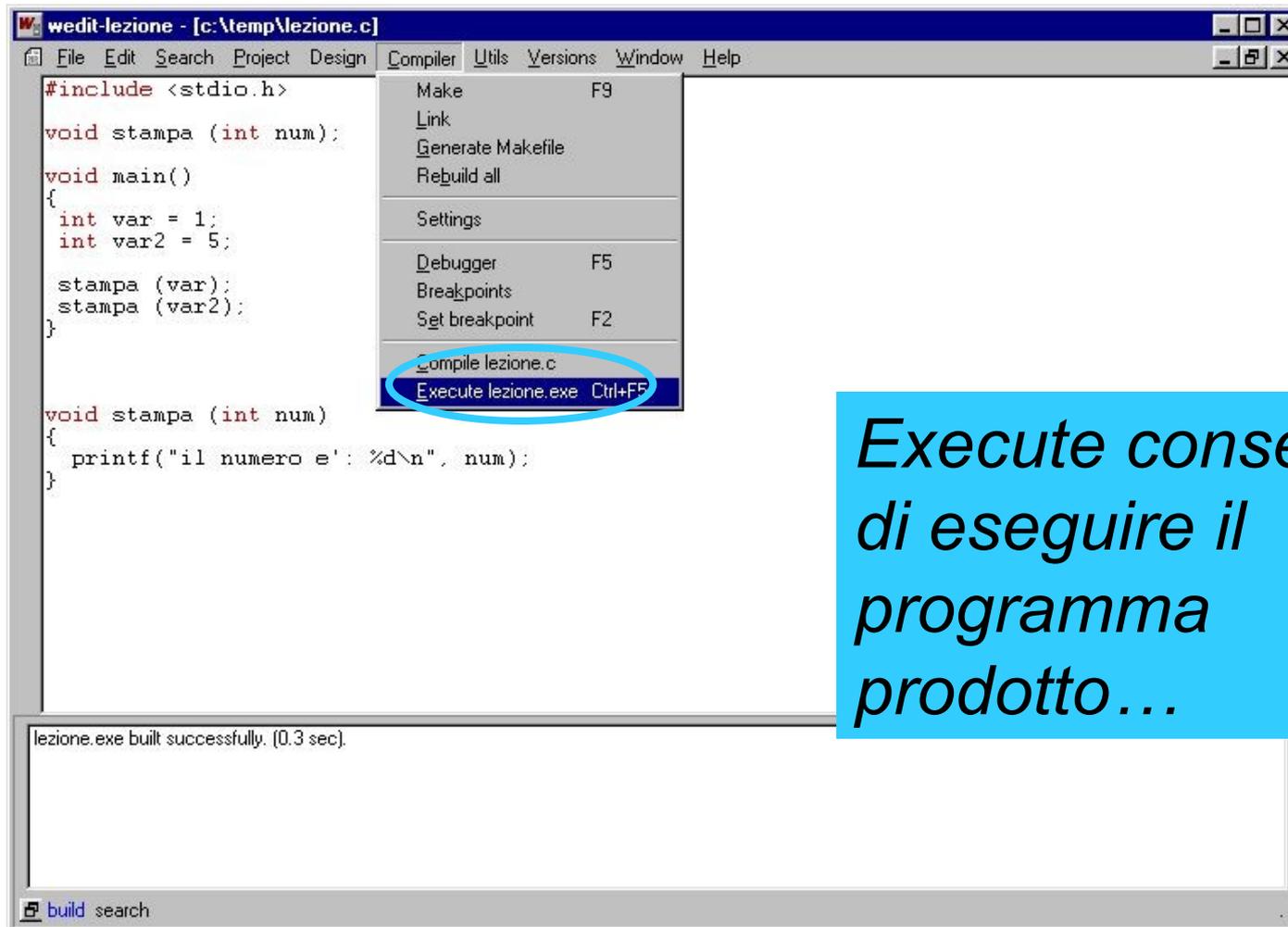
*Dal menu
Compiler
scegliere Link*

Make



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

Execute



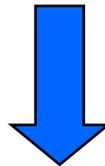
Execute consente di eseguire il programma prodotto...

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.**



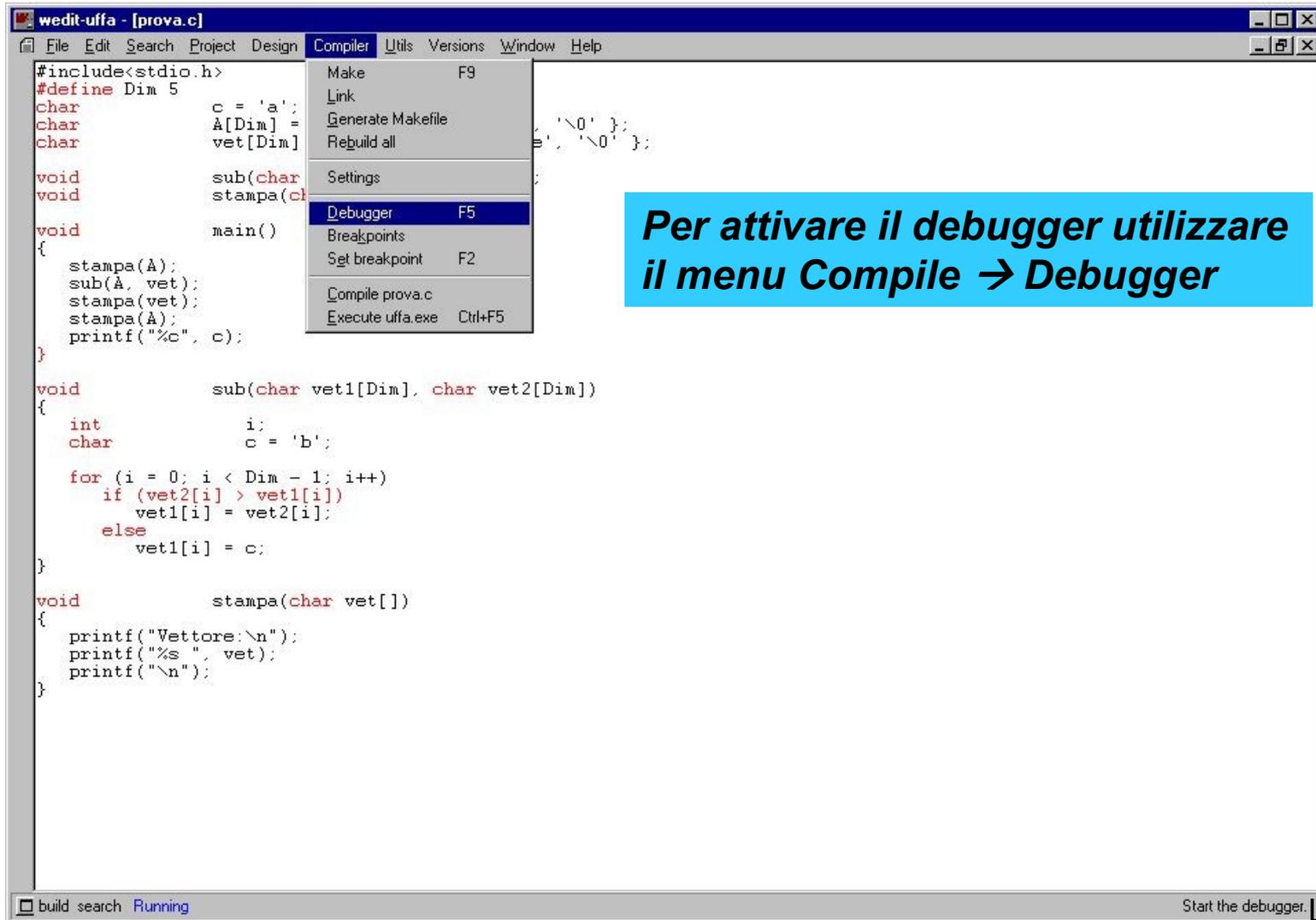
Debugger

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



The screenshot shows a Windows IDE window titled "wedit-uffa - [prova.c]". The menu bar includes "File", "Edit", "Search", "Project", "Design", "Compiler", "Utils", "Versions", "Window", and "Help". The "Compiler" menu is open, showing options: "Make" (F9), "Link", "Generate Makefile", "Rebuild all", "Settings", "Debugger" (F5), "Breakpoints", "Set breakpoint" (F2), "Compile prova.c", and "Execute uffa.exe" (Ctrl+F5). The "Debugger" option is highlighted. The main editor area contains C code for a program named "prova.c". The code includes a header file, defines a constant "Dim" as 5, and declares variables "c", "A", and "vet". It defines functions "sub", "stampa", and "main". The "main" function calls "stampa", "sub", and "stampa" again, and prints the character "c". The "sub" function compares two arrays "vet1" and "vet2" and updates "vet1" based on the comparison. The "stampa" function prints the contents of an array "vet".

```
#include<stdio.h>
#define Dim 5
char c = 'a';
char A[Dim] =
char vet[Dim]

void sub(char
void stampa(ch

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");
}
```

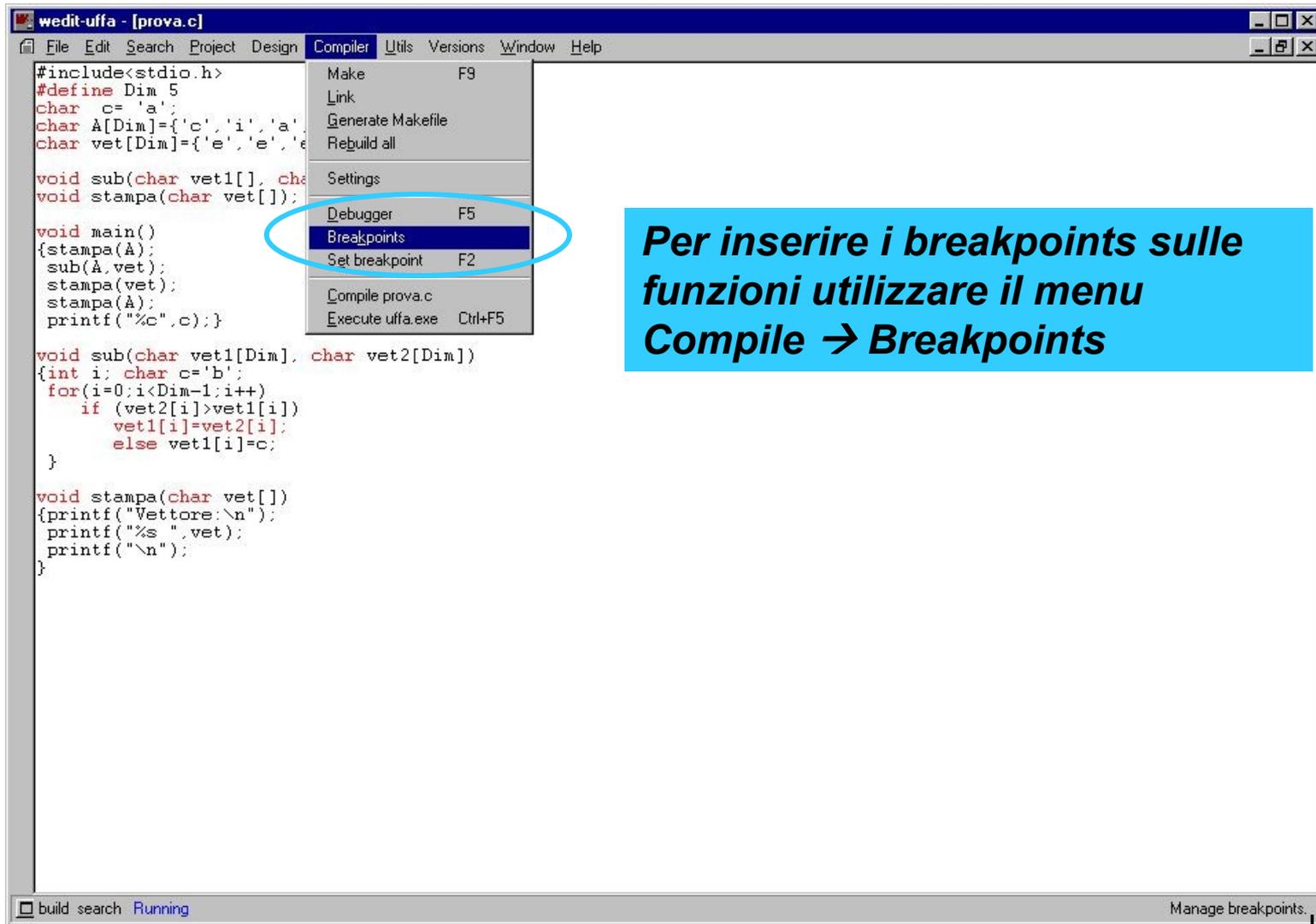
Per attivare il debugger utilizzare il menu Compile → Debugger

build search Running Start the debugger. Lab01 23

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



The screenshot shows a window titled "wedit-uffa - [prova.c]" with a menu bar including File, Edit, Search, Project, Design, Compiler, Utils, Versions, Window, and Help. The "Compiler" menu is open, showing options: Make (F9), Link, Generate Makefile, Rebuild all, Settings, Debugger (F5), Breakpoints (highlighted with a blue oval), Set breakpoint (F2), Compile prova.c, and Execute uffa.exe (Ctrl+F5). The main editor area contains C code for a program that prints a vector and compares it with another.

```
#include<stdio.h>
#define Dim 5
char c= 'a';
char A[Dim]={ 'c', 'i', 'a' };
char vet[Dim]={ 'e', 'e', 'e' };

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");
 }
```

build search Running Manage breakpoints.

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

Debugger

The image shows a debugger window titled "wedit-uffa - [prova.c]". The main window contains a C code editor with the following code:

```
#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]>vet
 vet1[i]=vet
 else vet1[i
 }

void stampa(char v
{printf("Vettore:\n
 printf("%s ",vet
 printf("\n");
}
```

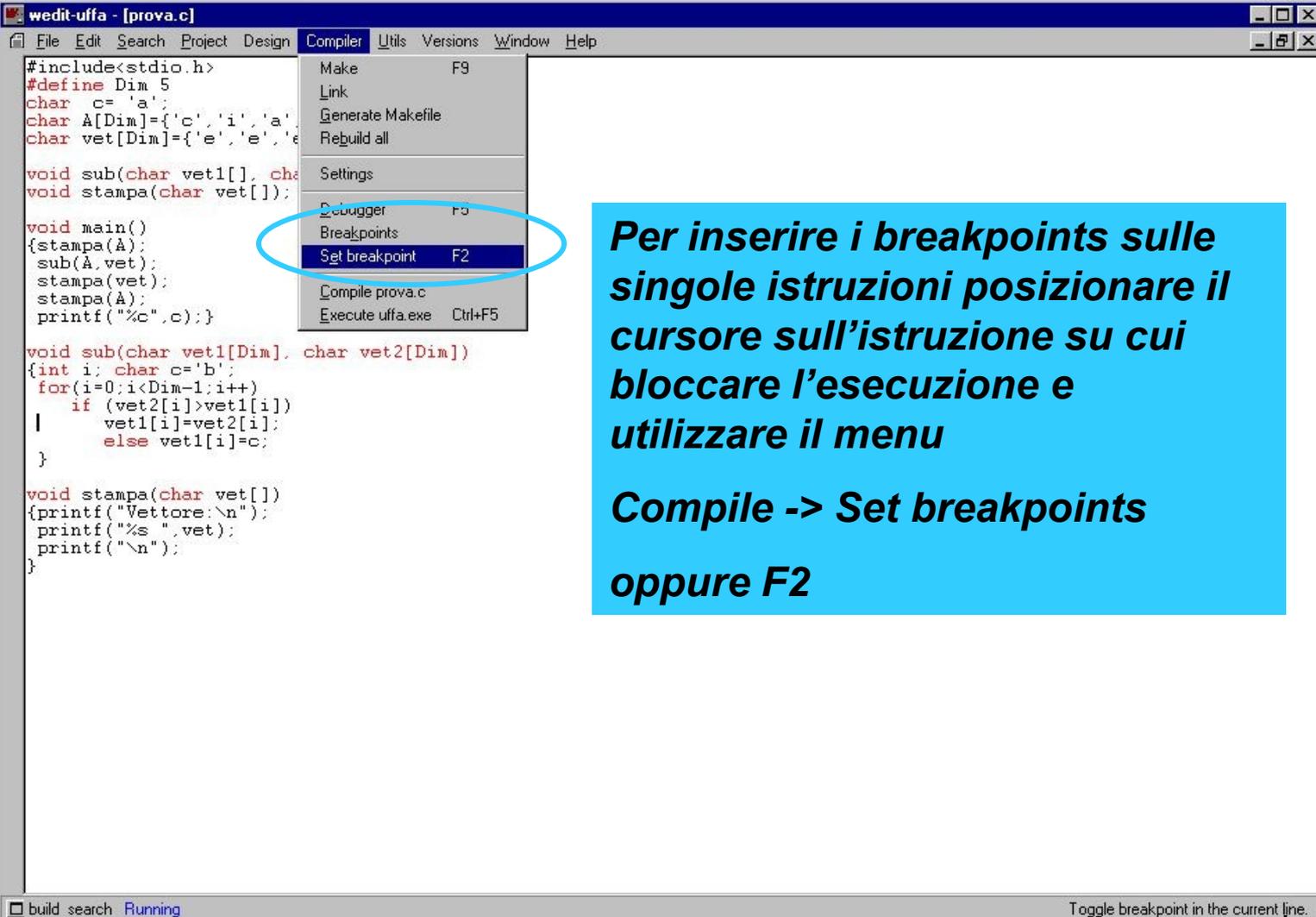
Overlaid on the code editor is a dialog box titled "Edit breakpoints". The dialog has a "Modules (exe and dll)" dropdown menu with "prova.c" selected. Below it is a list of functions: "main", "stampa", and "sub", with "sub" selected. The "New breakpoint: source, line number or function name" field contains "C:\TEMP\PROVA.C 17". The "Established breakpoints" list is empty. The "Add" button is circled in blue. Other buttons include "Ok", "Cancel", "Help", "Erase", and "Clear all".

Selezionare la funzione e cliccare Add

build search Running

Lab01 26

Debugger



The screenshot shows a Windows IDE window titled "wedit-uffa - [prova.c]". The menu bar includes "File", "Edit", "Search", "Project", "Design", "Compiler", "Utils", "Versions", "Window", and "Help". The "Compiler" menu is open, showing options: "Make F9", "Link", "Generate Makefile", "Rebuild all", "Settings", "Debugger F8", "Breakpoints", "Set breakpoint F2", "Compile prova.c", and "Execute uffa.exe Ctrl+F5". The "Set breakpoint F2" option is highlighted with a blue oval. The code editor displays a C program with the following content:

```
#include<stdio.h>
#define Dim 5
char c= 'a';
char A[Dim]={'c','i','a','a','a'};
char vet[Dim]={'e','e','e','e','e'};

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");
 }
```

At the bottom of the IDE, there is a status bar with "build search Running" on the left and "Toggle breakpoint in the current line." on the right.

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

The screenshot shows a debugger window titled "wedit-uffa - [prova.c]". The main pane displays C code with a breakpoint set on the line `wet1[i]=wet2[i];`. A blue callout box on the right states: "L'esecuzione del programma si ferma sull'istruzione o funzione precedentemente associata al breakpoint". The bottom pane shows the current state of variables: `i = 0`, `wet2[i] = 101 'e'`, `wet1[i] = 99 'c'`, `c = 98 'b'`, and `Dim = 5`. A blue callout box on the right of this pane states: "Vengono visualizzati i valori delle variabili". The status bar at the bottom indicates "Stopped" and "sub 21:2".

```
#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])
   wet1[i]=vet2[i];
  else vet1[i]=c;
 }

void stampa(char vet[])
{printf("Vettore:\n");
 printf("%s ",vet);
 printf("\n");
 }
```

L'esecuzione del programma si ferma sull'istruzione o funzione precedentemente associata al breakpoint

Vengono visualizzati i valori delle variabili

`i = 0`
`wet2[i] = 101 'e'`
`wet1[i] = 99 'c'`
`c = 98 'b'`
`Dim = 5`

auto locals stack events search Stopped sub 21:2

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

```
#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");
}
```

Watch che permette di monitorare variabili di particolare interesse

Stack: lo vedremo piu' avanti

A[5]A = [0..5] = "ciao"
vet[5]vet = [0..5] = "eeee"

Name	Value
A	[0..5] = "ciao"
c	97 'a'

auto locals stack events search Stopped sub 19:52