

COSTRUZIONE DI UN'APPLICAZIONE

Per costruire un'applicazione occorre:

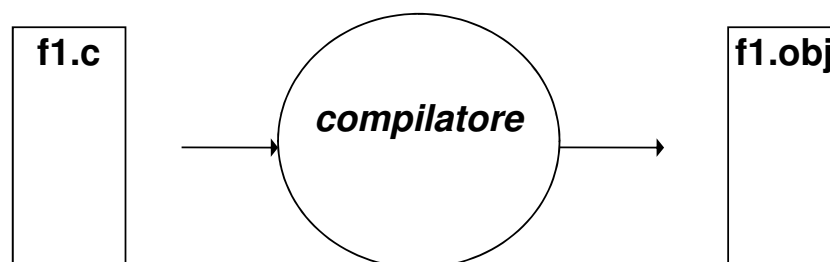
- **compilare il file (o / file se più d'uno) che contengono il testo del programma (file *sorgente*)**
Il risultato sono uno o più file *oggetto*.
- **collegare i file oggetto l'uno con l'altro e con le librerie di sistema.**

1

COMPILAZIONE DI UN'APPLICAZIONE

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

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



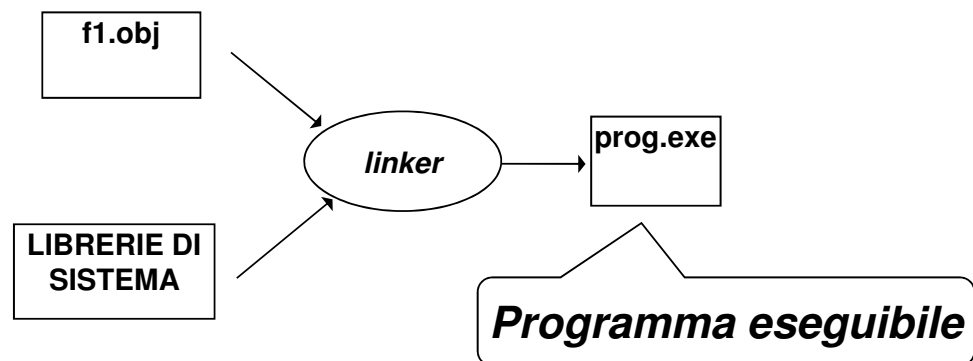
f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

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

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.*

5

PROGETTI

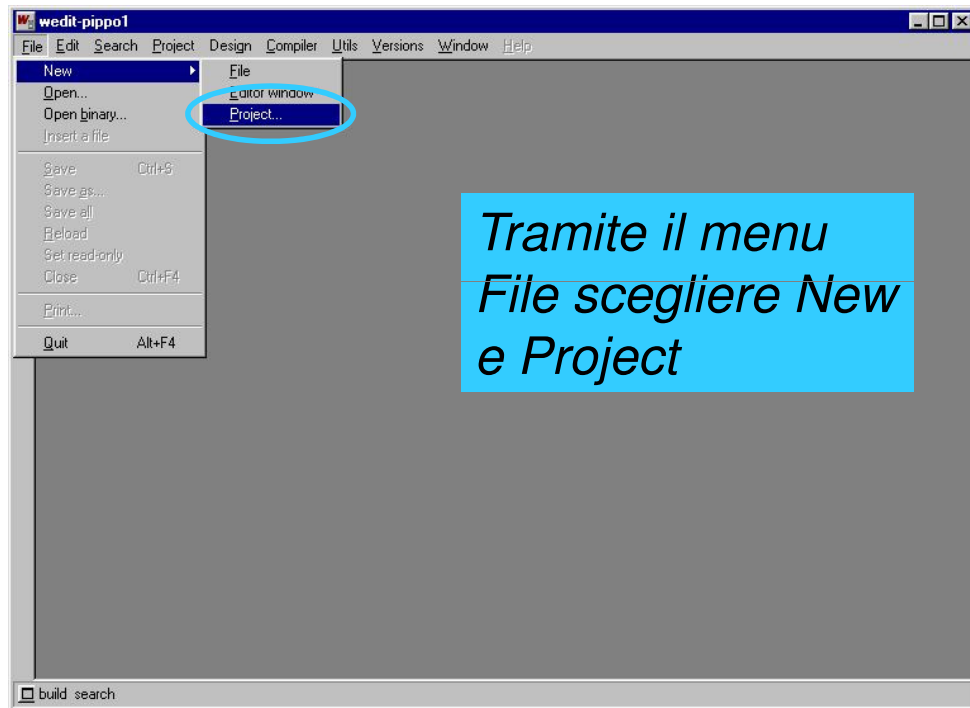
È da queste esigenze che nasce il concetto di *PROGETTO*

- *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.

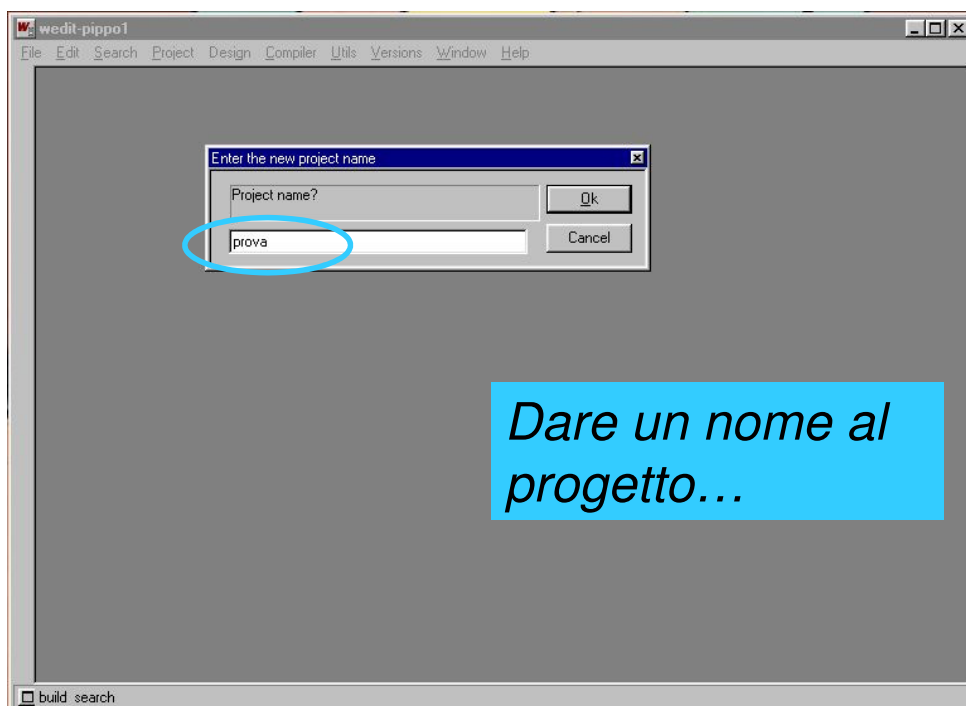
6

PROGETTI IN LCC



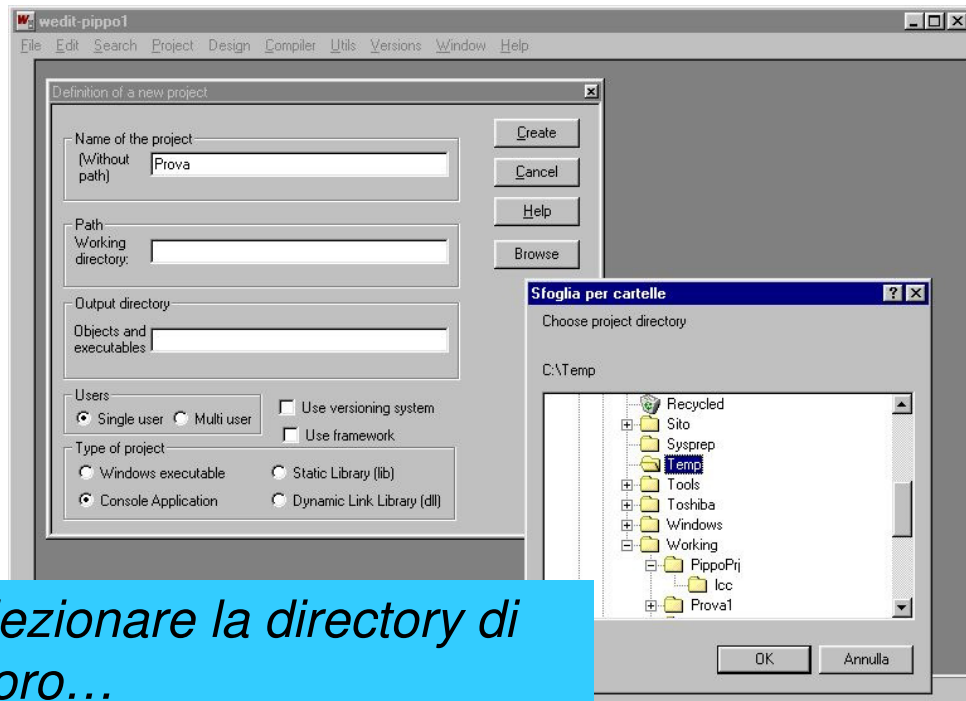
7

PROGETTI IN LCC



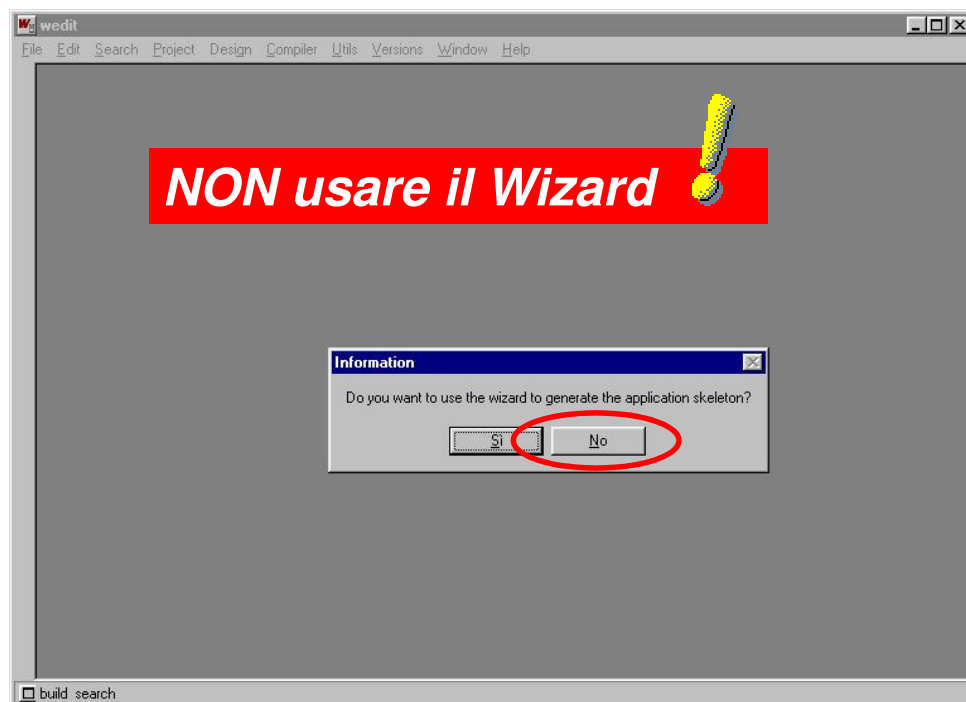
8

PROGETTI IN LCC



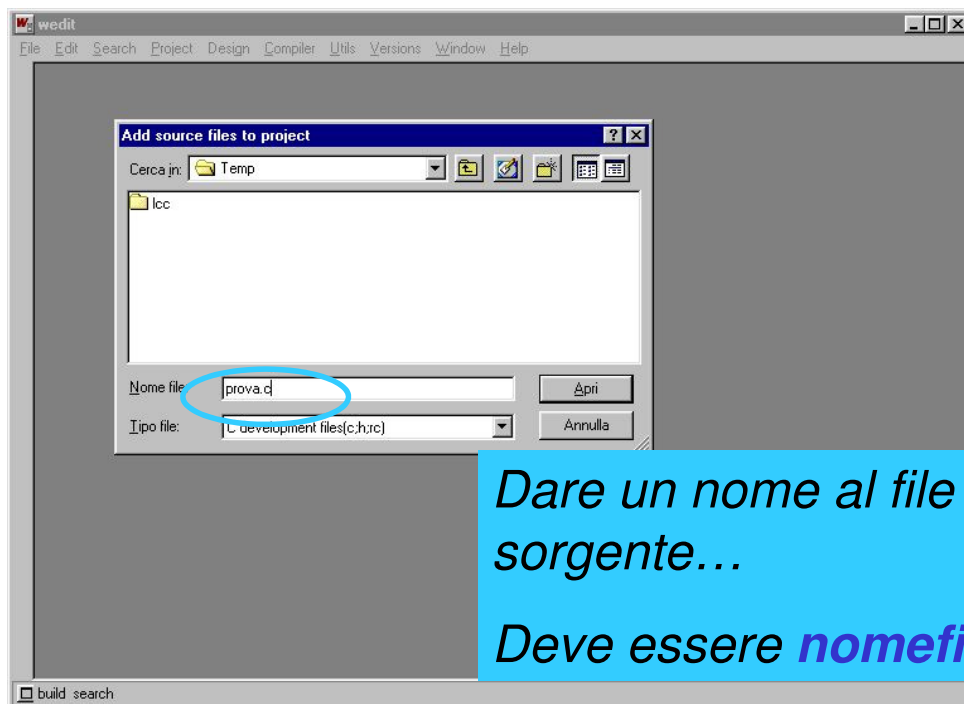
9

PROGETTI IN LCC



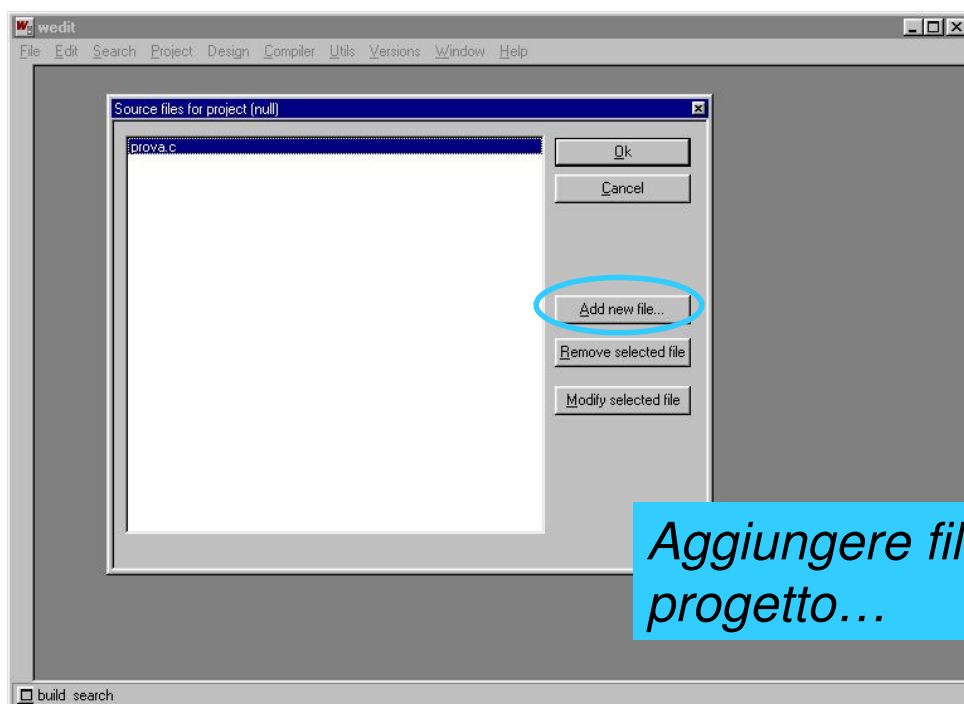
10

PROGETTI IN LCC



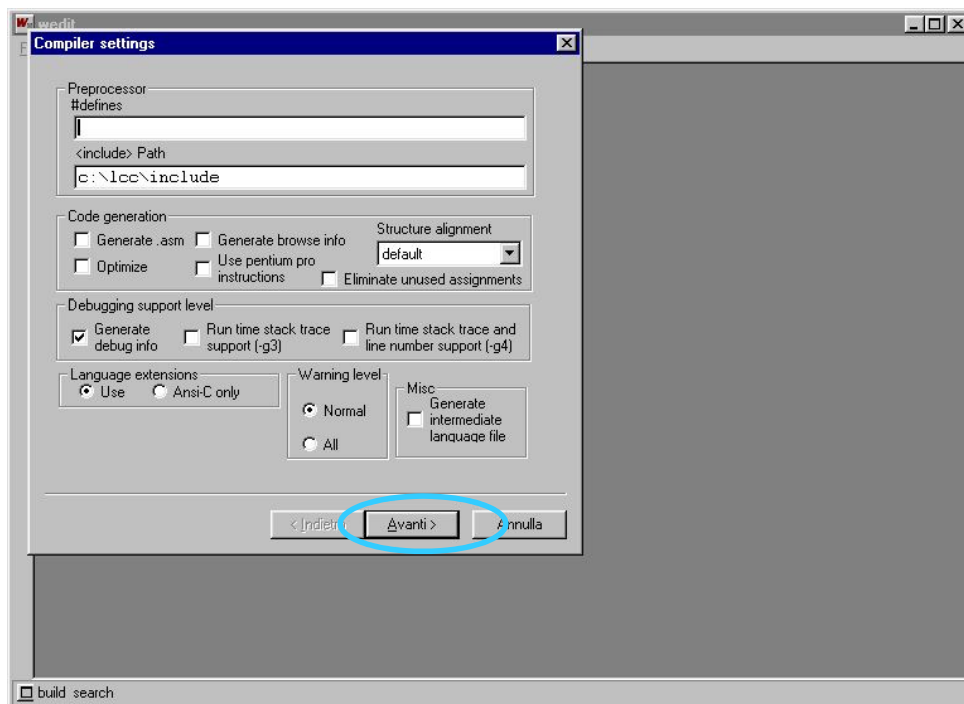
11

PROGETTI IN LCC



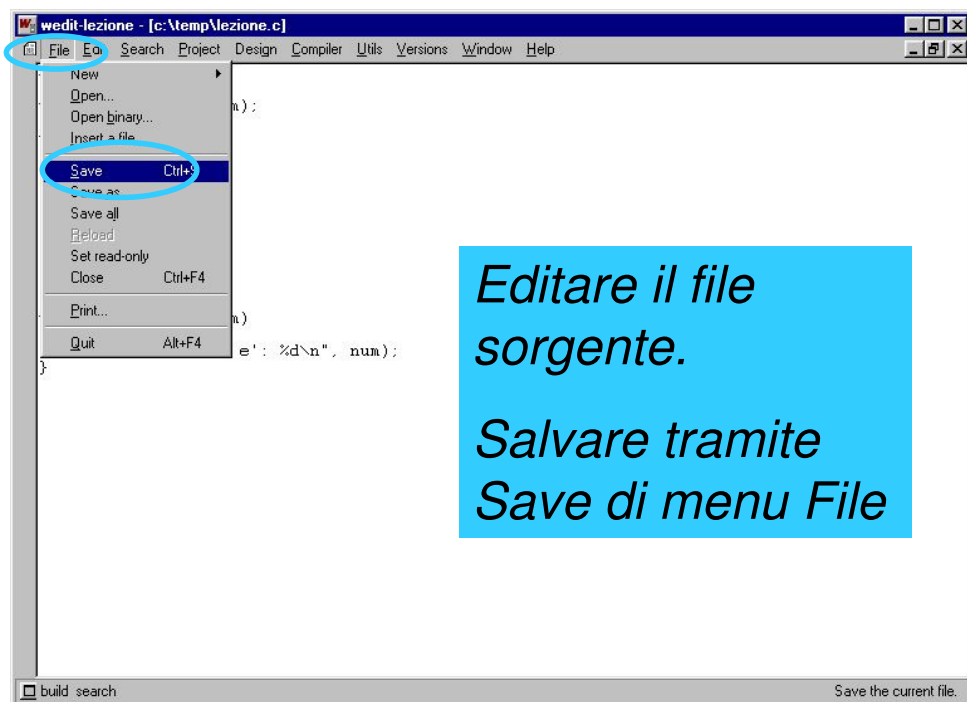
12

PROGETTI IN LCC



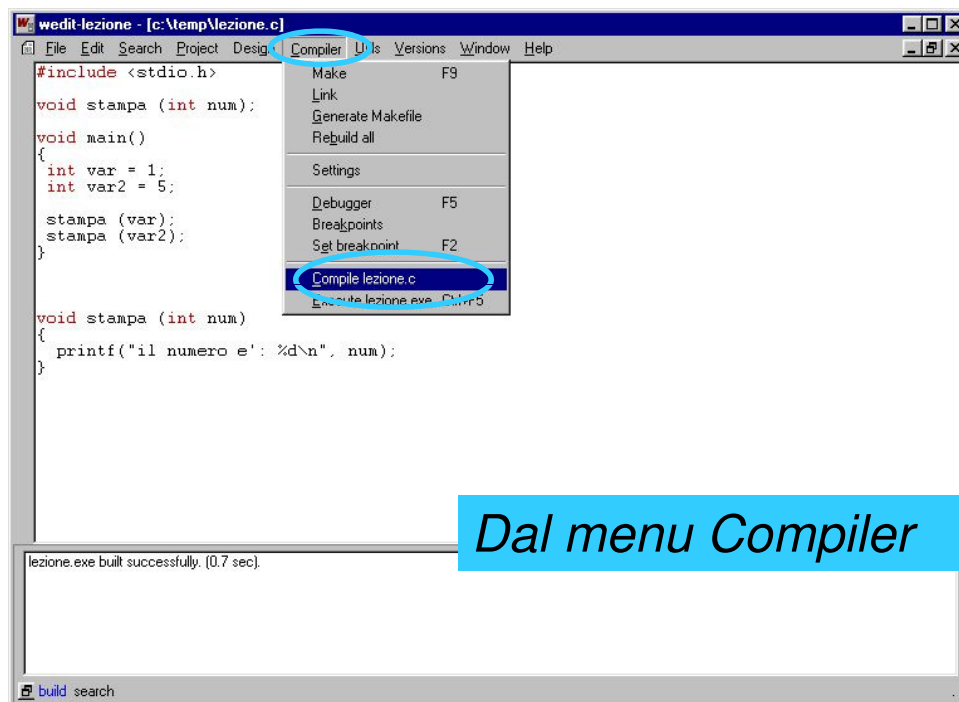
13

EDITARE E SALVARE



14

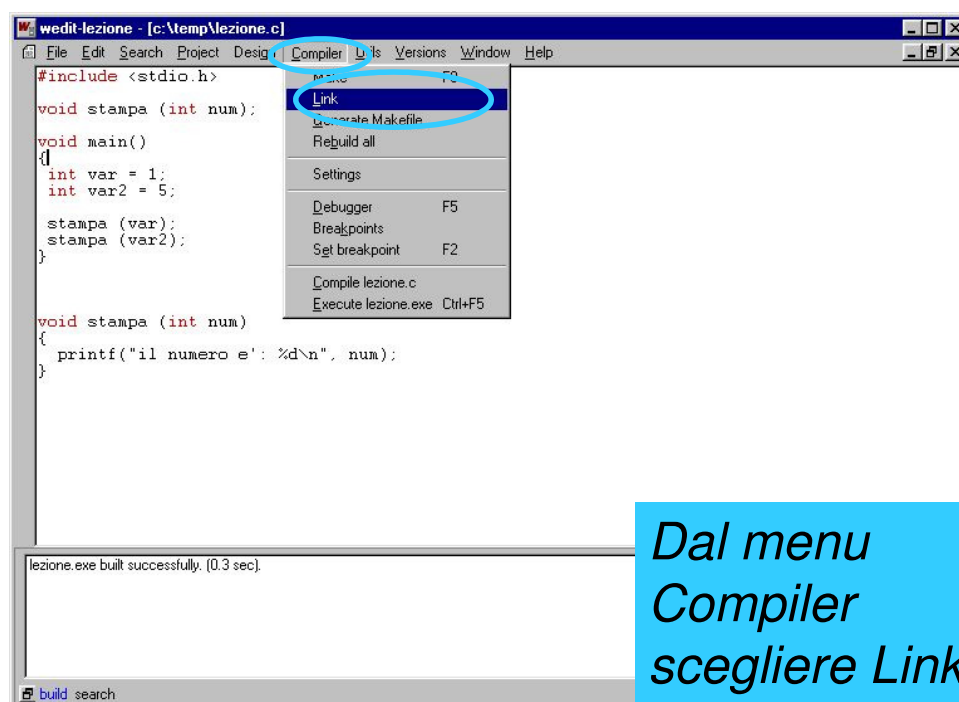
COMPILARE



Dal menu Compiler

15

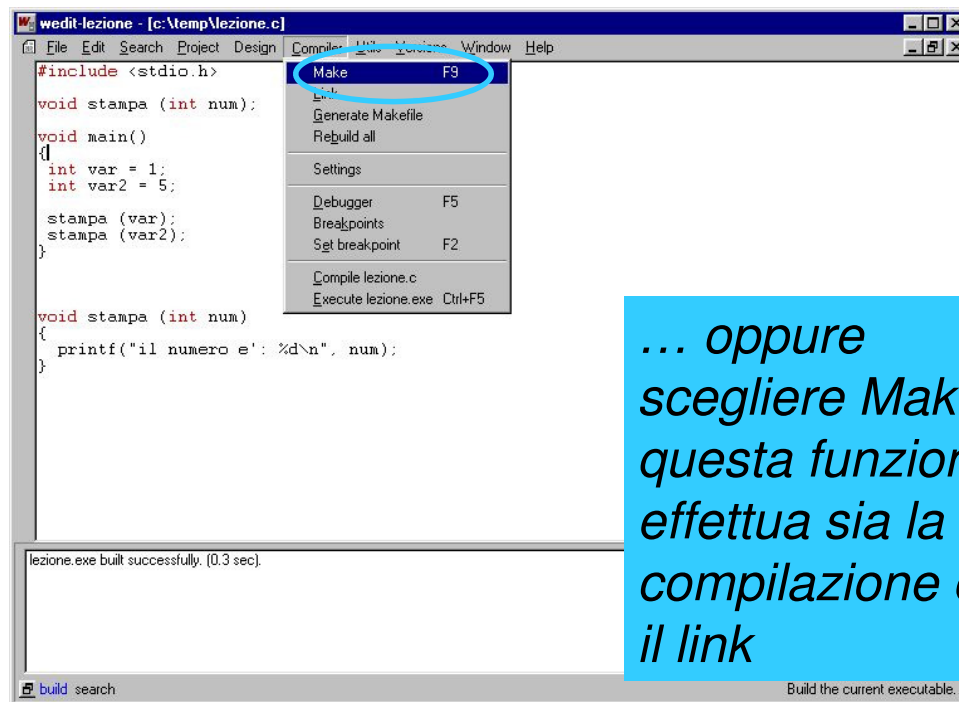
LINK



*Dal menu
Compiler
scegliere Link*

16

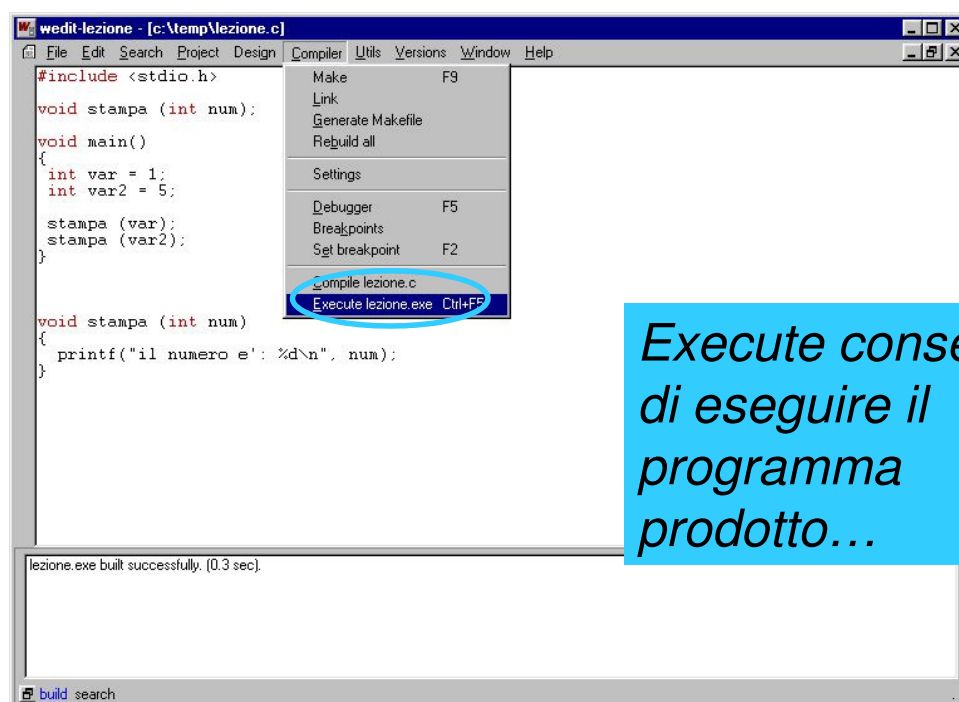
MAKE



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

17

EXECUTE

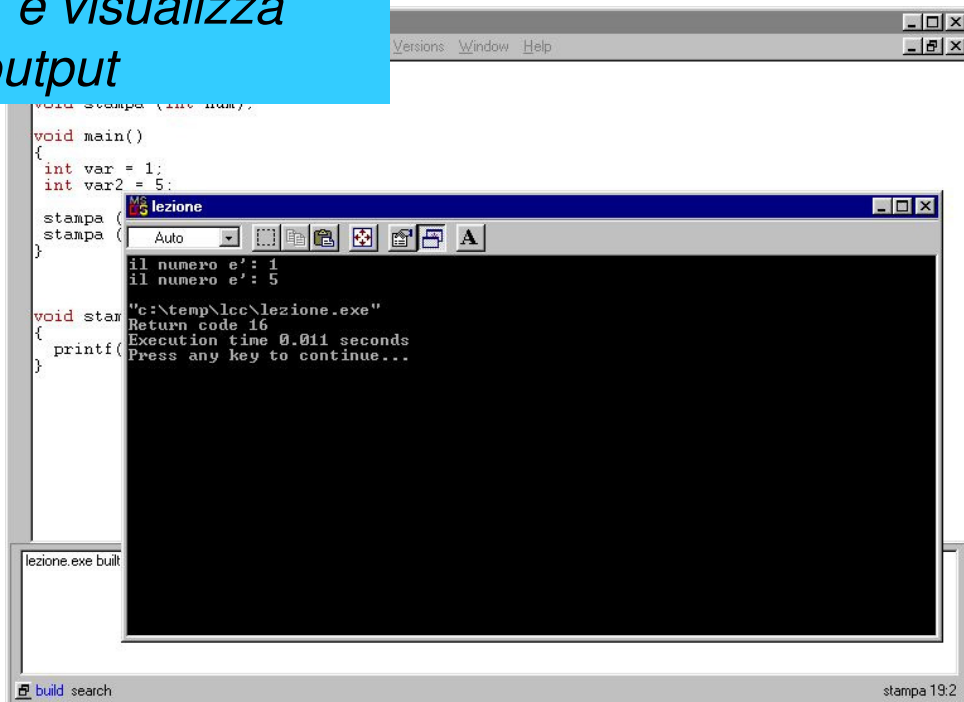


*Execute consente
di eseguire il
programma
prodotto...*

18

EXECUTE

... e visualizza
l'output



19

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

20

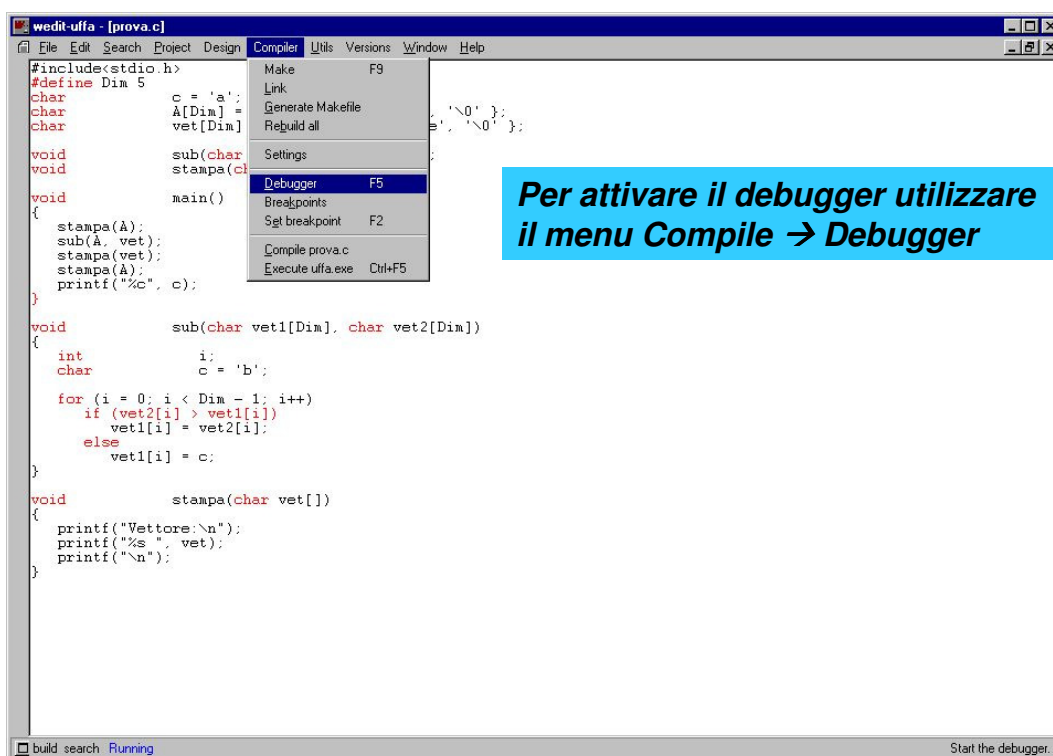
DEBUGGER

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**

21

DEBUGGER



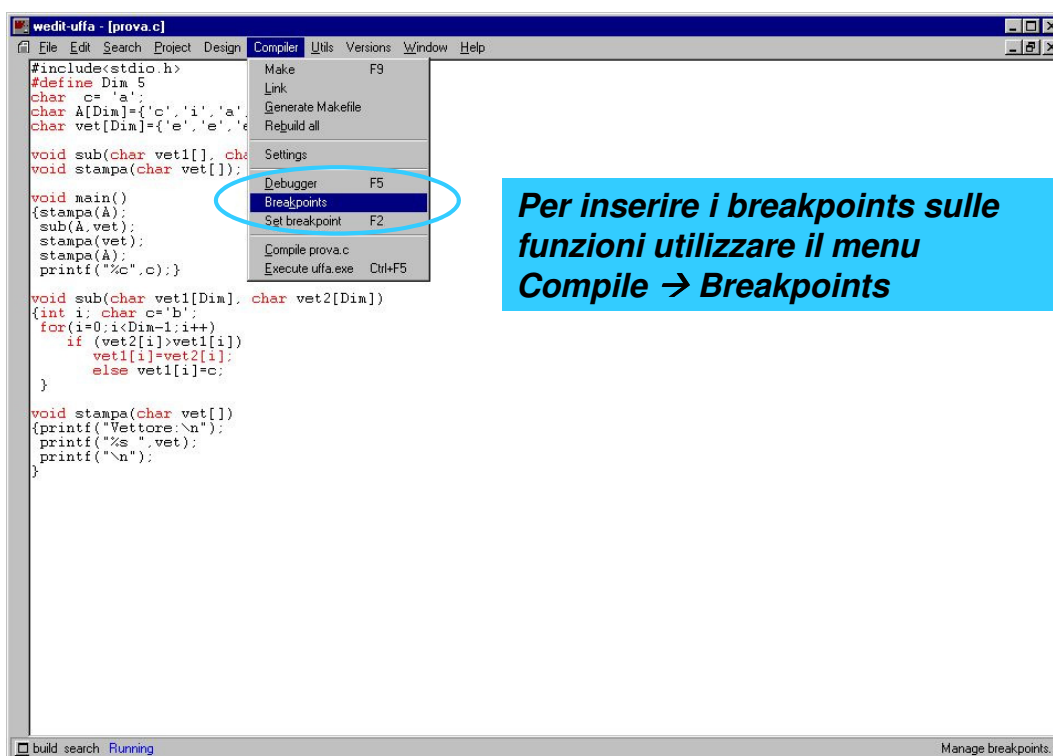
22

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*

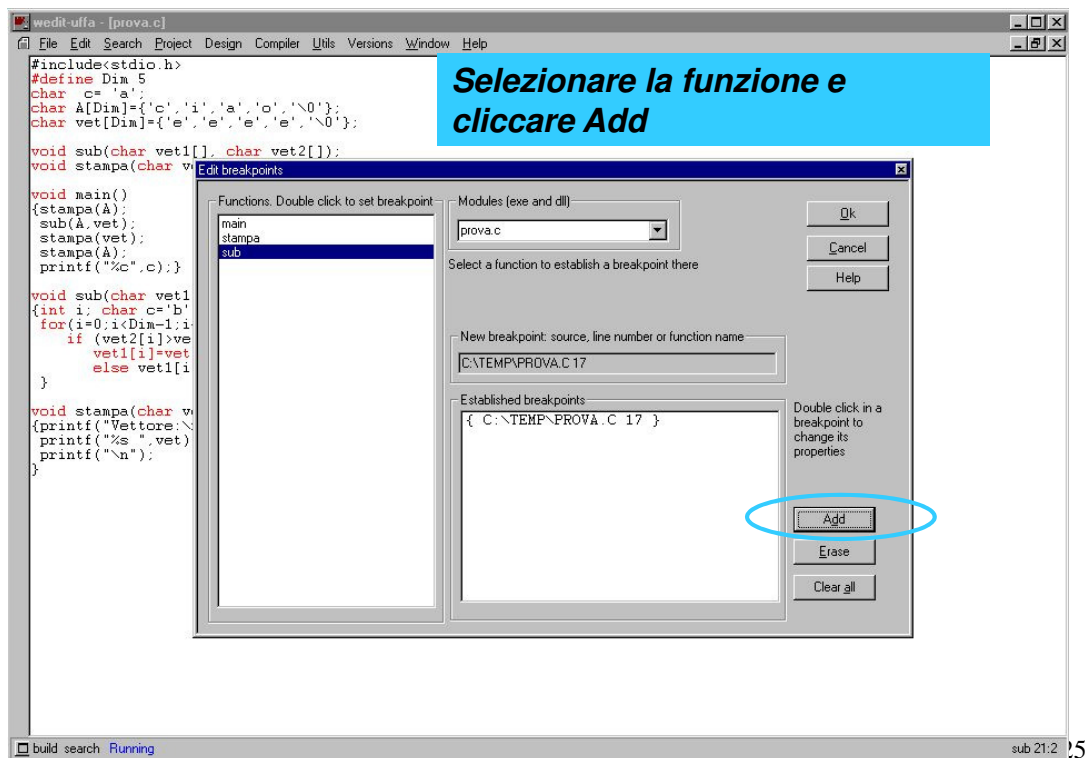
23

DEBUGGER

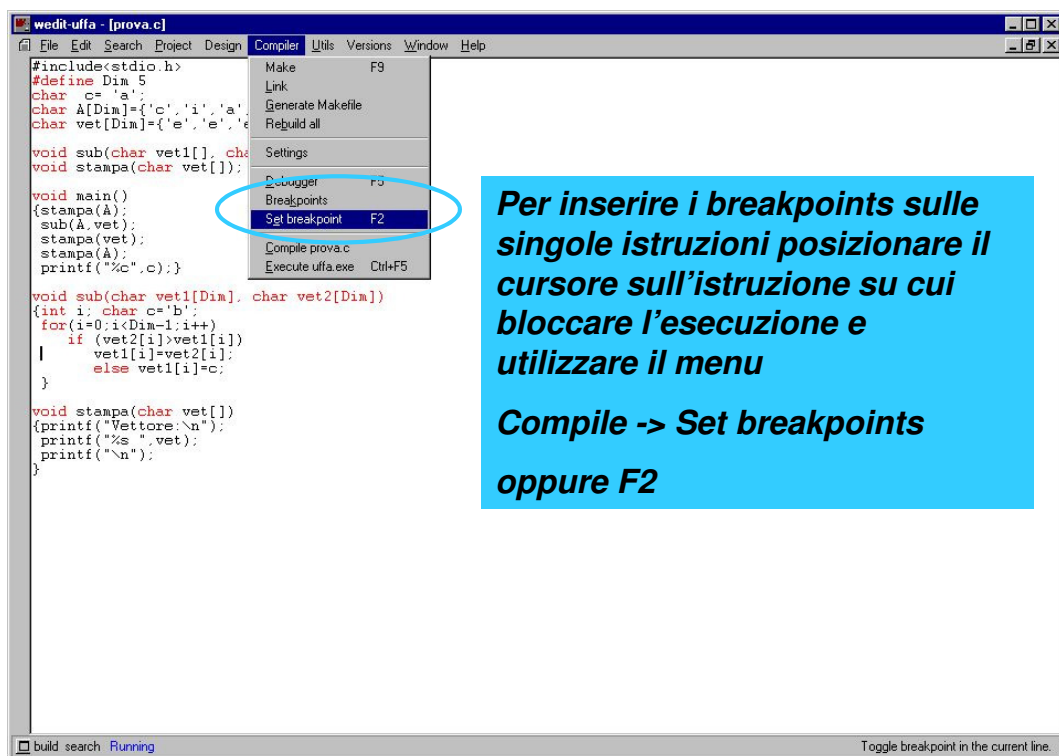


24

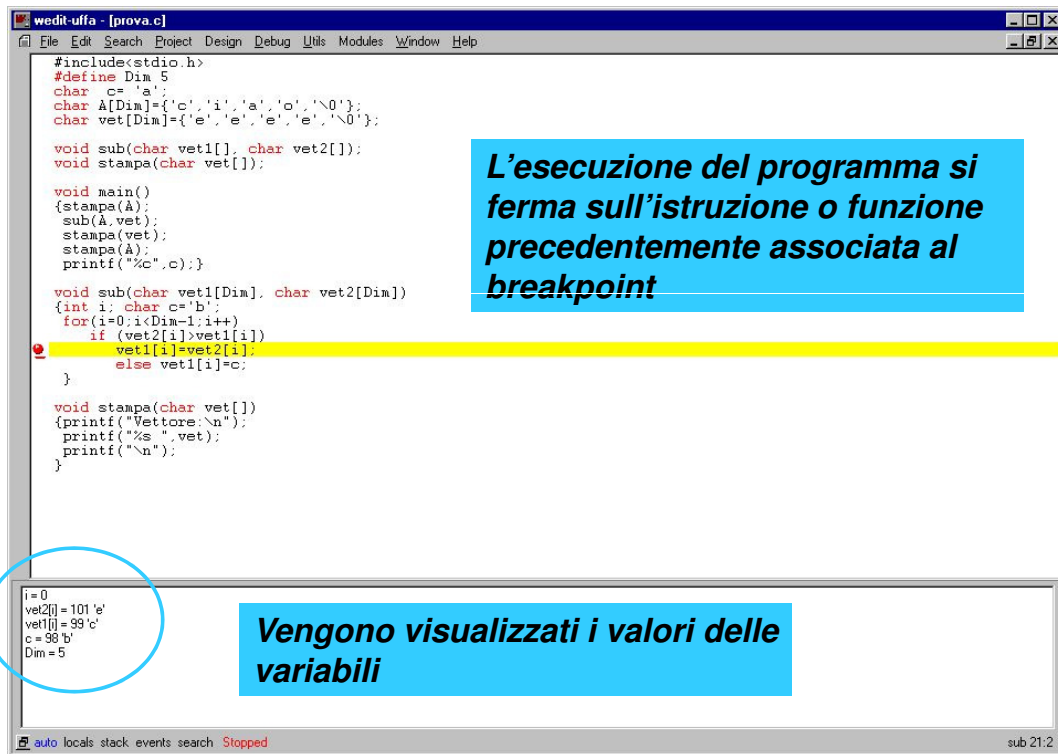
DEBUGGER



DEBUGGER



DEBUGGER



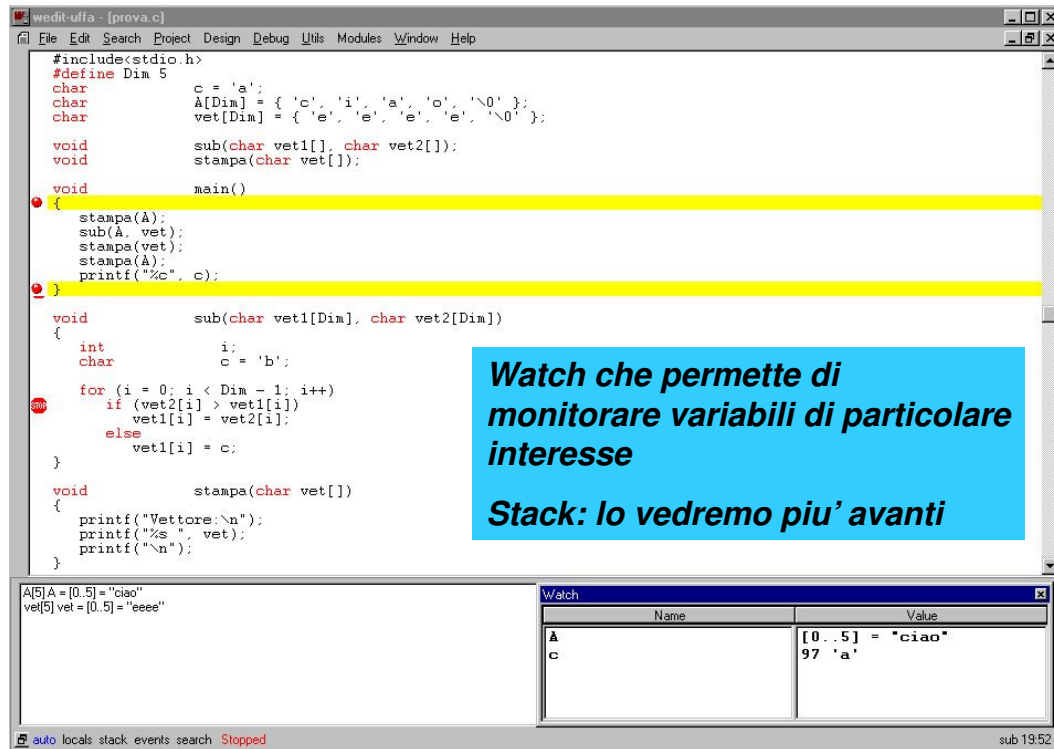
27

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.

28

DEBUGGER: COME PROCEDERE



The screenshot shows a debugger window with a C program. The code defines a character array `A` and a function `sub` that compares two arrays. The `main` function calls `sub` and `stampa`. The debugger is stopped at the `for` loop in `sub`. A blue box highlights the text: "Watch che permette di monitorare variabili di particolare interesse" and "Stack: lo vedremo piu' avanti". The Watch window shows the variable `A` with value `[0..5] = "ciao"` and the variable `c` with value `97 'a'`.

```
#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

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

sub 19:52