

COME MODIFICARE LA PASSWORD

Collegarsi al sito

<https://labx.ing.unibo.it/changepassword/index.php>

Inserire username e password, dopodichè scegliere la nuova password.

1

PER UTILIZZARE LCC IN LAB2

La prima volta che un utente utilizza lcc-win32 deve impostare una chiave nel registro utente.

Per fare questo occorre semplicemente:

- Aprire con il gestore risorse la cartella x:\lcc**
- Fare doppio clic sul file registry.reg**

E' sufficiente fare questa operazione la prima volta e le impostazioni verranno mantenute.

2

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*.

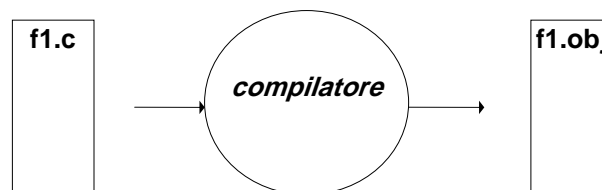
- **collegare i file oggetto l'uno con l'altro e con le librerie di sistema.**

3

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**



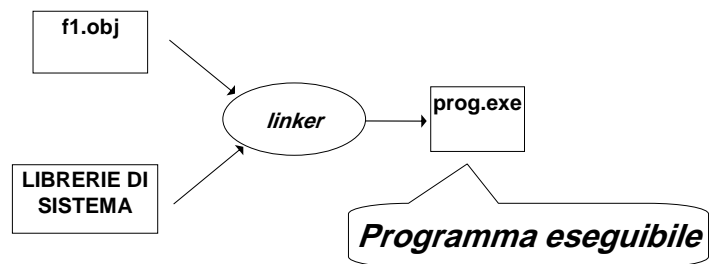
f1.obj: Una versione tradotta che però non è autonoma (e, quindi, non è direttamente eseguibile).

4

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



5

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

6

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.**

7

PROGETTI

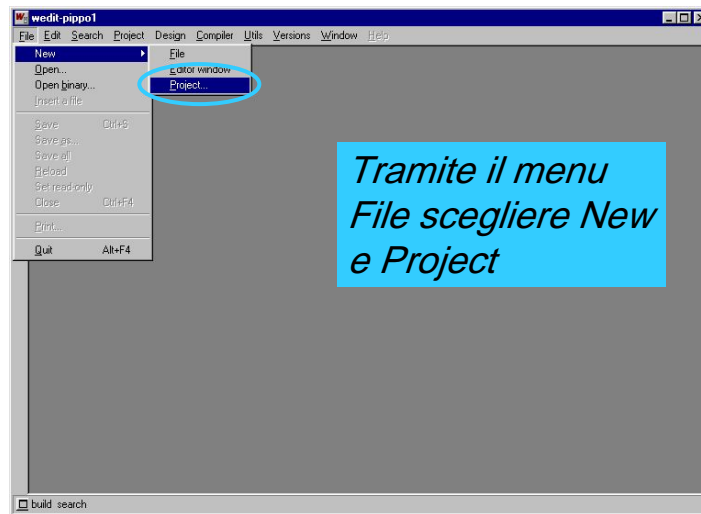
È da queste esigenze che nasce il concetto di **PROGETTO**

- **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.

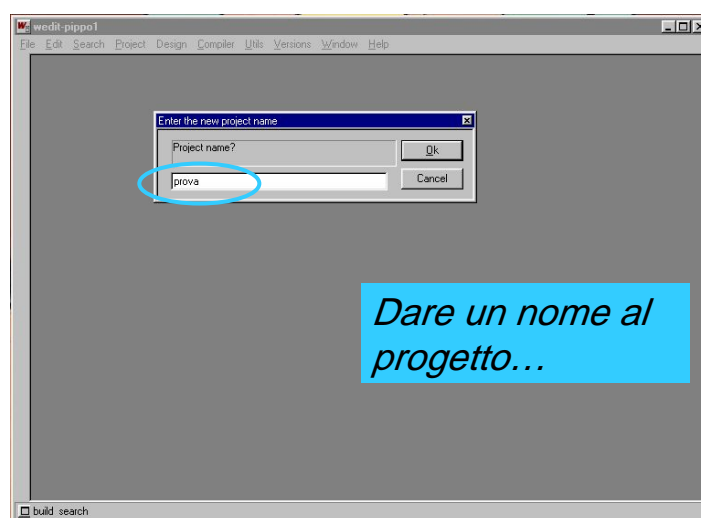
8

PROGETTI IN LCC



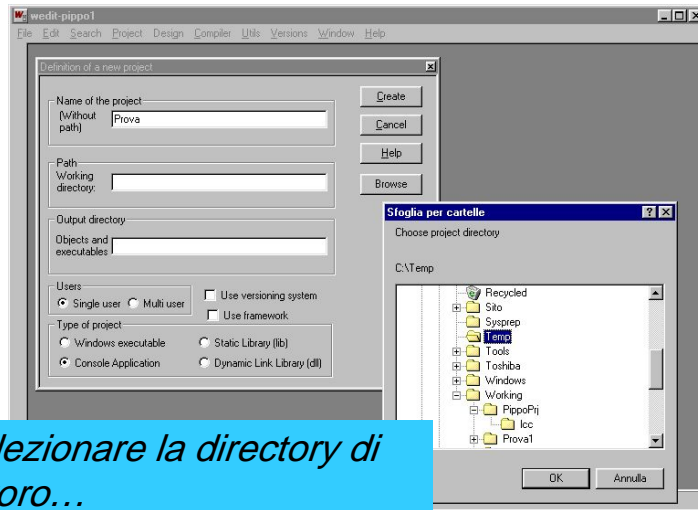
9

PROGETTI IN LCC



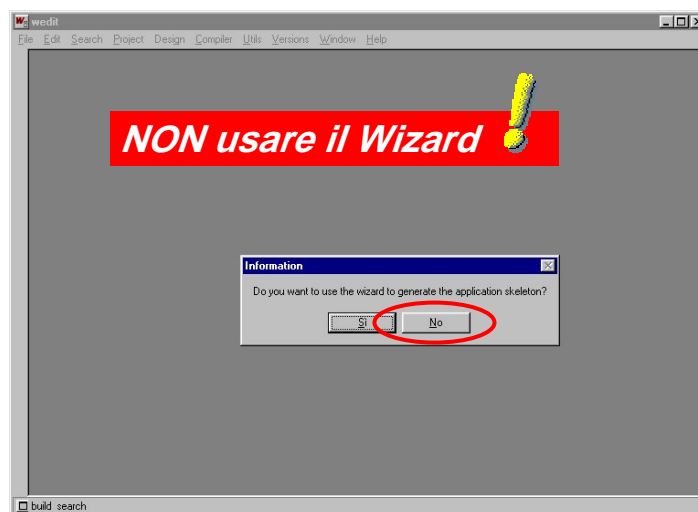
10

PROGETTI IN LCC



11

PROGETTI IN LCC



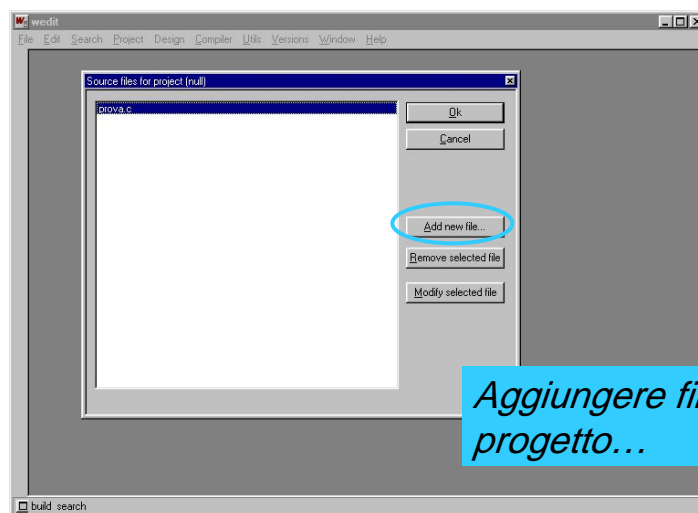
12

PROGETTI IN LCC



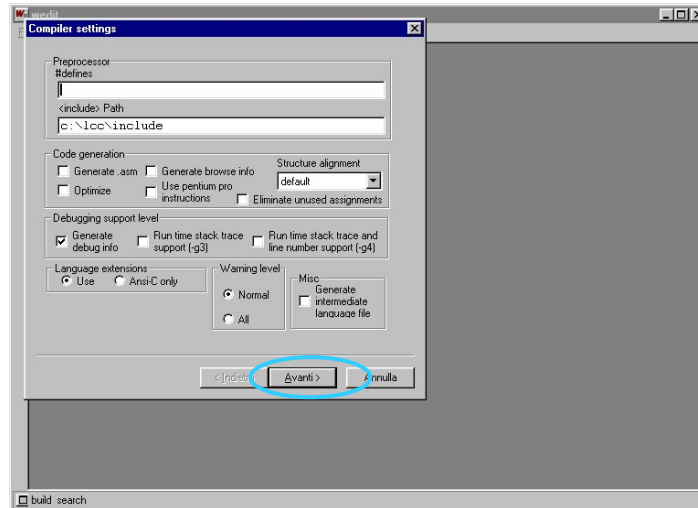
13

PROGETTI IN LCC



14

PROGETTI IN LCC



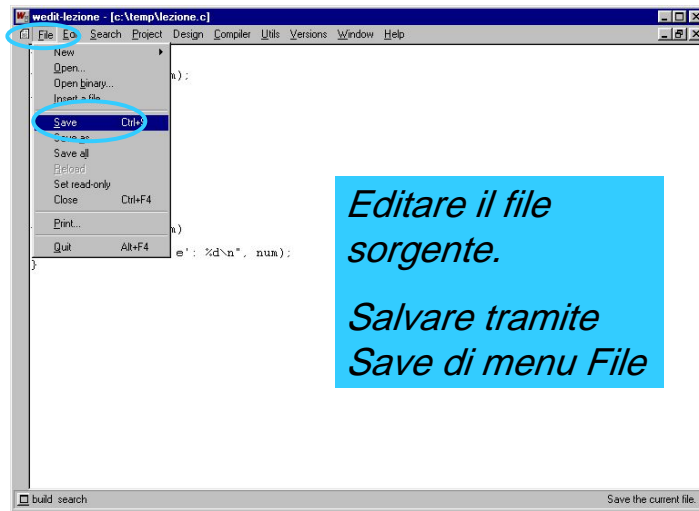
15

PRIMO PROGRAMMA IN LCC

```
#include <stdio.h>  
  
main()  
{int x,y;  
scanf("%d%d",&x,&y);  
printf("%d",x+y);  
}
```

16

EDITARE E SALVARE

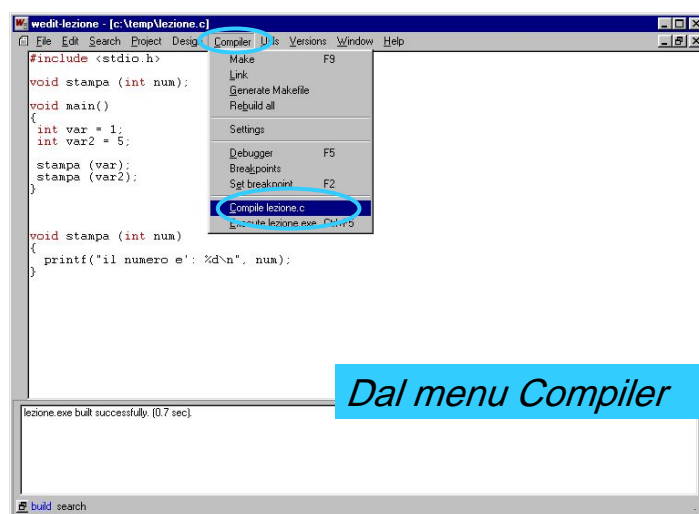


Editare il file sorgente.

Salvare tramite Save di menu File

17

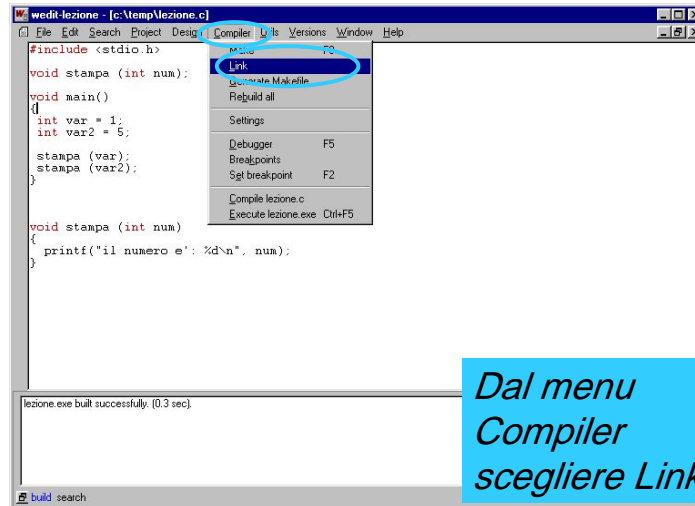
COMPILARE



Dal menu Compiler

18

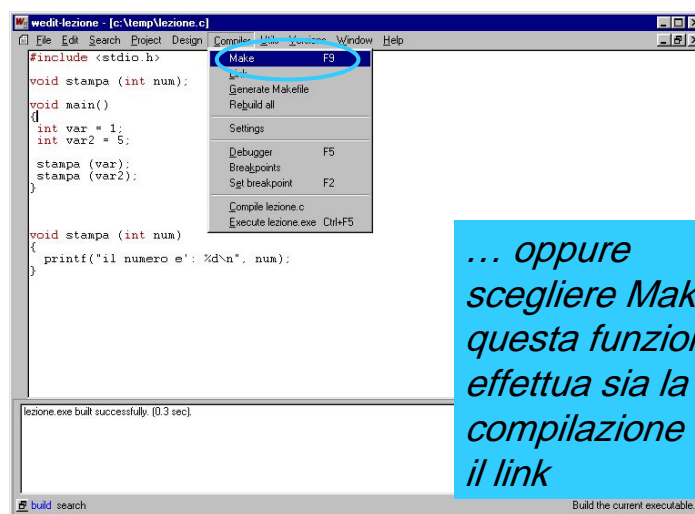
LINK



*Dal menu
Compiler
scegliere Link*

19

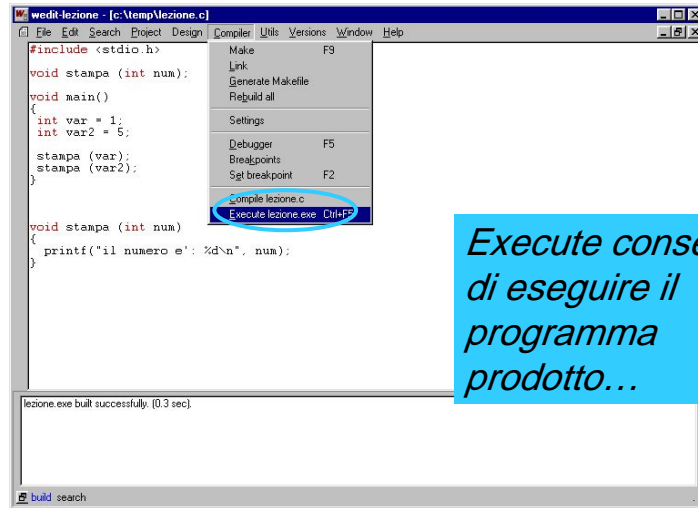
MAKE



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

20

EXECUTE

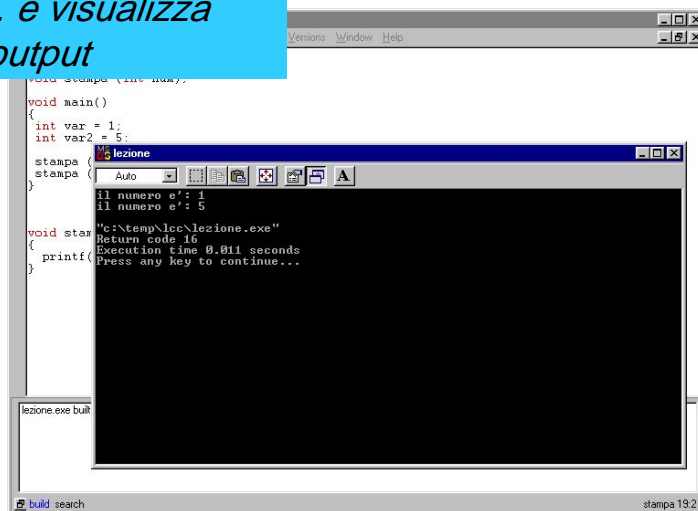


Execute consente di eseguire il programma prodotto...

21

EXECUTE

... e visualizza l'output



22

PRIMO PROGRAMMA CON COMMENTI

```
#include <stdio.h>

main()
{int x,y;
 printf("Inserire due numeri separati da virgola: ");
 scanf("%d,%d" , &x , &y);
 printf("%d + %d = %d" , x , y , x+y);
}
```

23

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

24

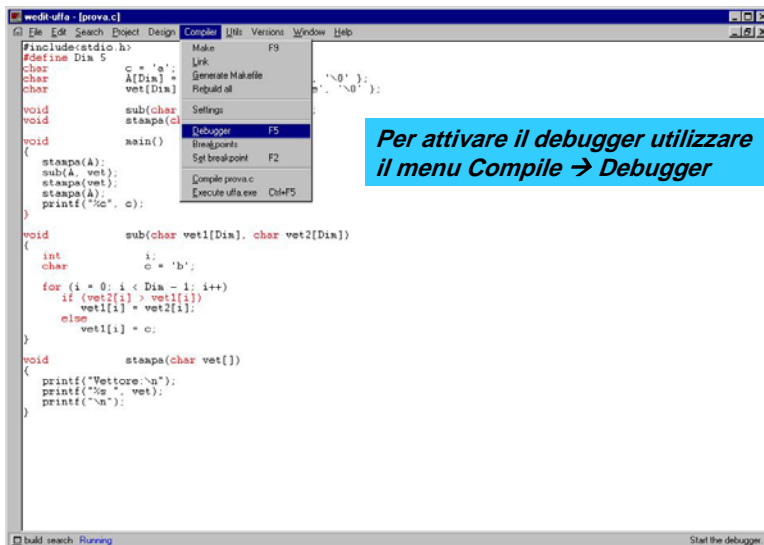
DEBUGGER

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**

25

DEBUGGER



```
File Edit Search Project Design Compile Units Versions Window Help
Make F9
Link
Generate Makefile
Rebuild all
Settings
Debugger F5
Breakpoints
Set breakpoint F2
Compile prova.c
Execute uffa.exe Ctrl+F5

#include<stdio.h>
#define Dim 5
char c = 'a';
char A[Dim] =
char vet[Dim]
void sub(char
void staapa(c)
void main()
{
    staapa(A);
    sub(A, vet);
    staapa(vet);
    staapa(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 staapa(char vet[])
{
    printf("Vettore:\n");
    printf("%s", vet);
    printf("\n");
}

build search Running Start the debugger.
```

Per attivare il debugger utilizzare il menu Compile -> Debugger

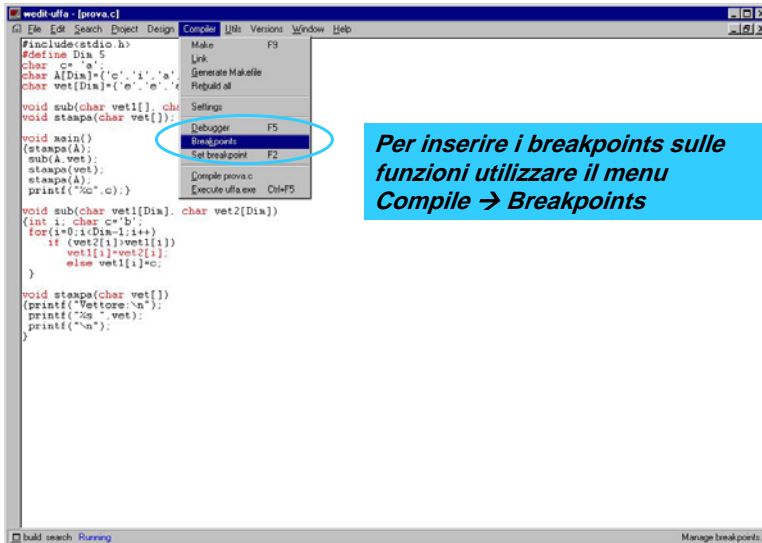
26

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*

27

DEBUGGER



```
void main()
{
    stampa(A);
    sub(A, vet);
    stampa(vet);
    stampa(A);
    printf("%c", c);
}

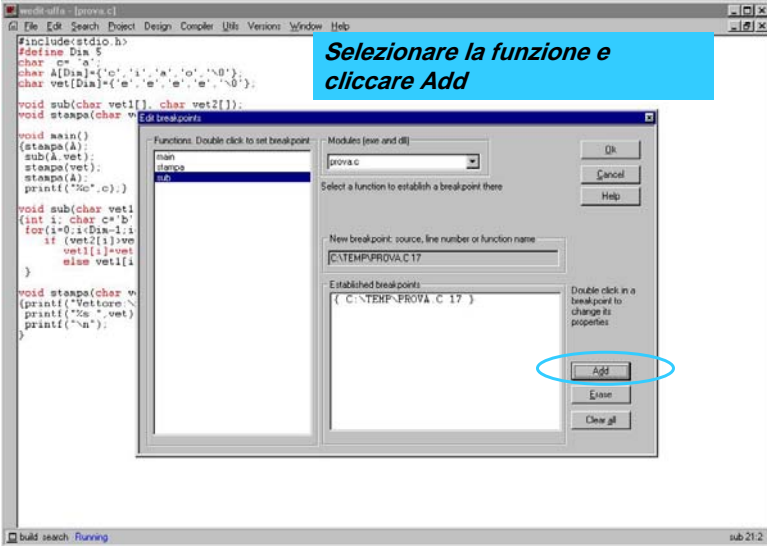
void sub(char vet1[Dim], char vet2[Dim])
{
    int i, char c='b';
    for(i=0; i<Dim-1; i++)
    {
        if (vet2[i]==vet1[i])
            vet1[i]=vet2[i];
        else vet1[i]=c;
    }
}

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

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

28

DEBUGGER



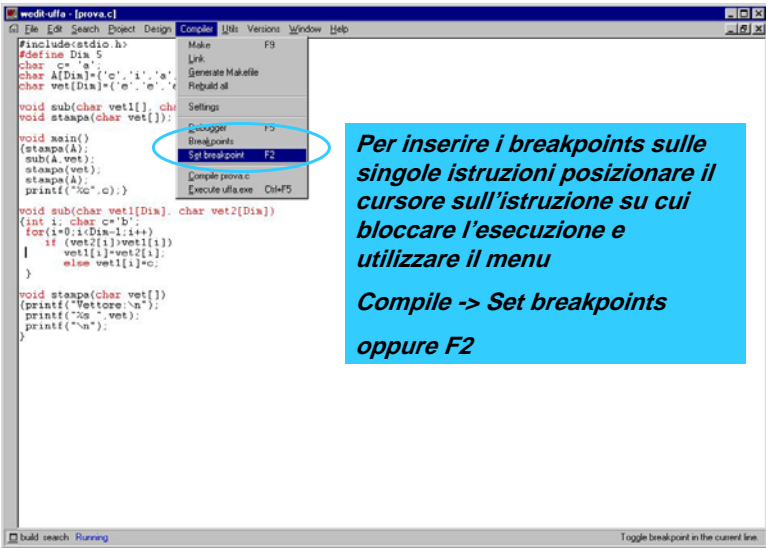
Selezionare la funzione e cliccare Add

```
void main()
{
    stampa(A);
    sub(A, vet);
    stampa(vet);
    stampa(A);
    printf("%c", c);
}

void sub(char vet1[], char vet2[]);

void stampa(char v[]);
```

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

```
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");
}
```

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 `vet1[i]=vet2[i];`. A blue callout box on the right states: *L'esecuzione del programma si ferma sull'istruzione o funzione precedentemente associata al breakpoint*. A yellow highlight is under the breakpoint line. A bottom pane shows variable values: `i=0`, `vet2[0]=101 '\'`, `vet1[0]=99 'c'`, `c=99 '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'.

```
#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;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[0]=101 '\'
vet1[0]=99 'c'
c=99 'b'
Dim=5

31

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.

32

DEBUGGER: COME PROCEDERE

The screenshot shows a debugger window 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 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");
}
```

Two lines in the code are highlighted in yellow: the first line of the `main` function and the `printf` statement. A blue box contains the text: *Watch che permette di monitorare variabili di particolare interesse* and *Stack: lo vedremo piu' avanti*.

The Watch window shows the following variables and their values:

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

At the bottom of the debugger window, the status bar shows: `sub locals stack events search Skipped sub 19/52`.

33