

ERRORI A RUN TIME

- Gli errori in un programma C possono comparire:
 - in fase di editing (sintassi)
 - in fase di compilazione o linking (prima dell'esecuzione)
 - in fase di esecuzione (errori a run time)
- Gli errori a run time possono essere complessi da correggere
 - Occorre quindi uno strumento per “monitorare” l'andamento dell'applicazione durante l'esecuzione
 - Il *debugger*

IL DEBUGGER

- Il debugger viene invocato dopo la compilazione ed il linking
- Consente di eseguire il programma step-by-step
 - In questo modo è possibile capire, ad esempio, in quale punto del programma viene causato l'errore
- Consente di monitorare:
 - Il valore delle variabili
 - La chiamata a funzioni
 - E altro ancora... (il contenuto dei registri, lo stack)

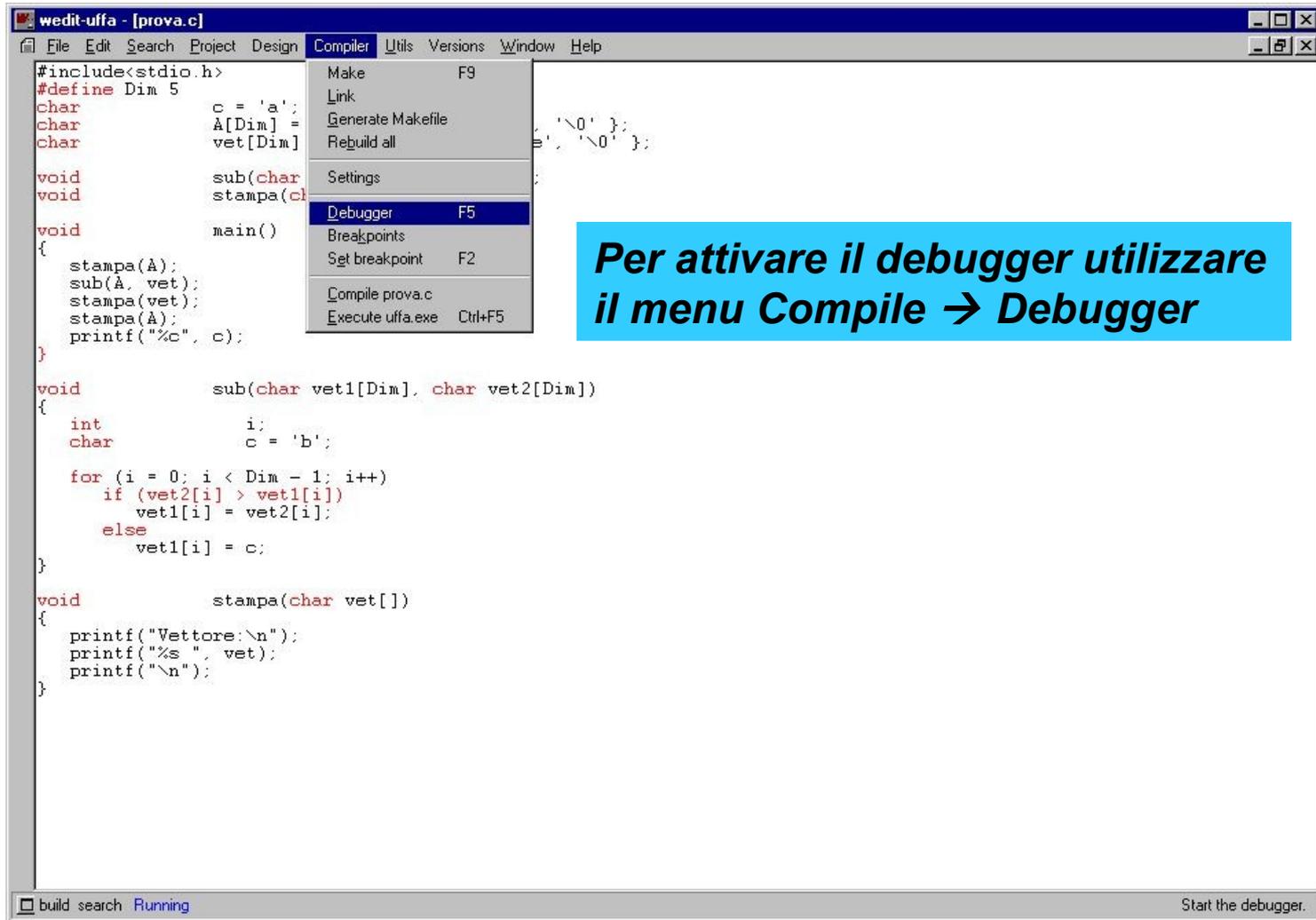
IL DEBUGGER IN LCC

- LCC, come altri IDE, fornisce un debugger
- Il debugger può essere usato in due modi:
 1. Fermando l'esecuzione in punti specifici del codice (*breakpoints*)
 2. Eseguendo il codice riga per riga (che equivale ad un breakpoint per riga)

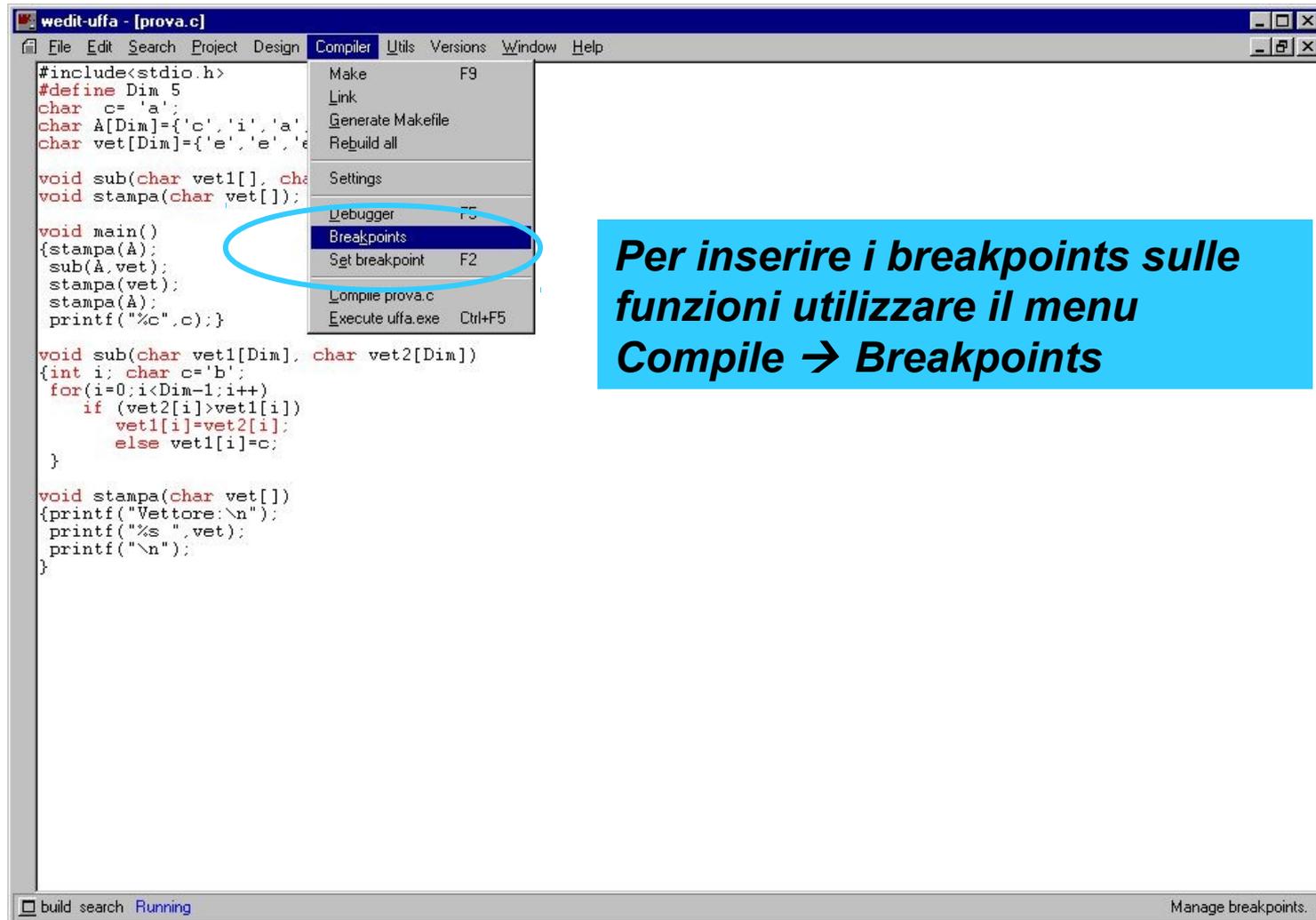
BREAKPOINTS

- I breakpoint sono punti in cui l'esecuzione del programma si ferma e fornisce una “fotografia” dello stato di esecuzione
 - in particolare, del valore corrente delle variabili
- I breakpoint possono essere inseriti:
 - su singole *istruzioni* (l'esecuzione si blocca alla riga contenente l'istruzione specifica)
 - su *funzioni* (l'esecuzione si blocca ad ogni chiamata della funzione specifica)

LCC – DEBUGGER (1)



LCC – DEBUGGER (2)



The screenshot shows the wedit-uffa IDE window with the Compiler menu open. The menu items are: Make (F9), Link, Generate Makefile, Rebuild all, Settings, Debugger (F5), Breakpoints (highlighted), Set breakpoint (F2), Compile prova.c, and Execute uffa.exe (Ctrl+F5). The code in the background includes a C program with functions sub, stampa, and main. A blue callout box on the right contains the text: **Per inserire i breakpoints sulle funzioni utilizzare il menu Compile → Breakpoints**. The status bar at the bottom indicates 'build search Running' and 'Manage breakpoints.'

LCC – DEBUGGER (3)

Selezionare la funzione e cliccare Add

```
#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:\n
 printf("%s ",vet)
 printf("\n");
}
```

Functions. Double click to set breakpoint

main
stampa
sub

Modules (exe and dll)

prova.c

Select a function to establish a breakpoint there

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

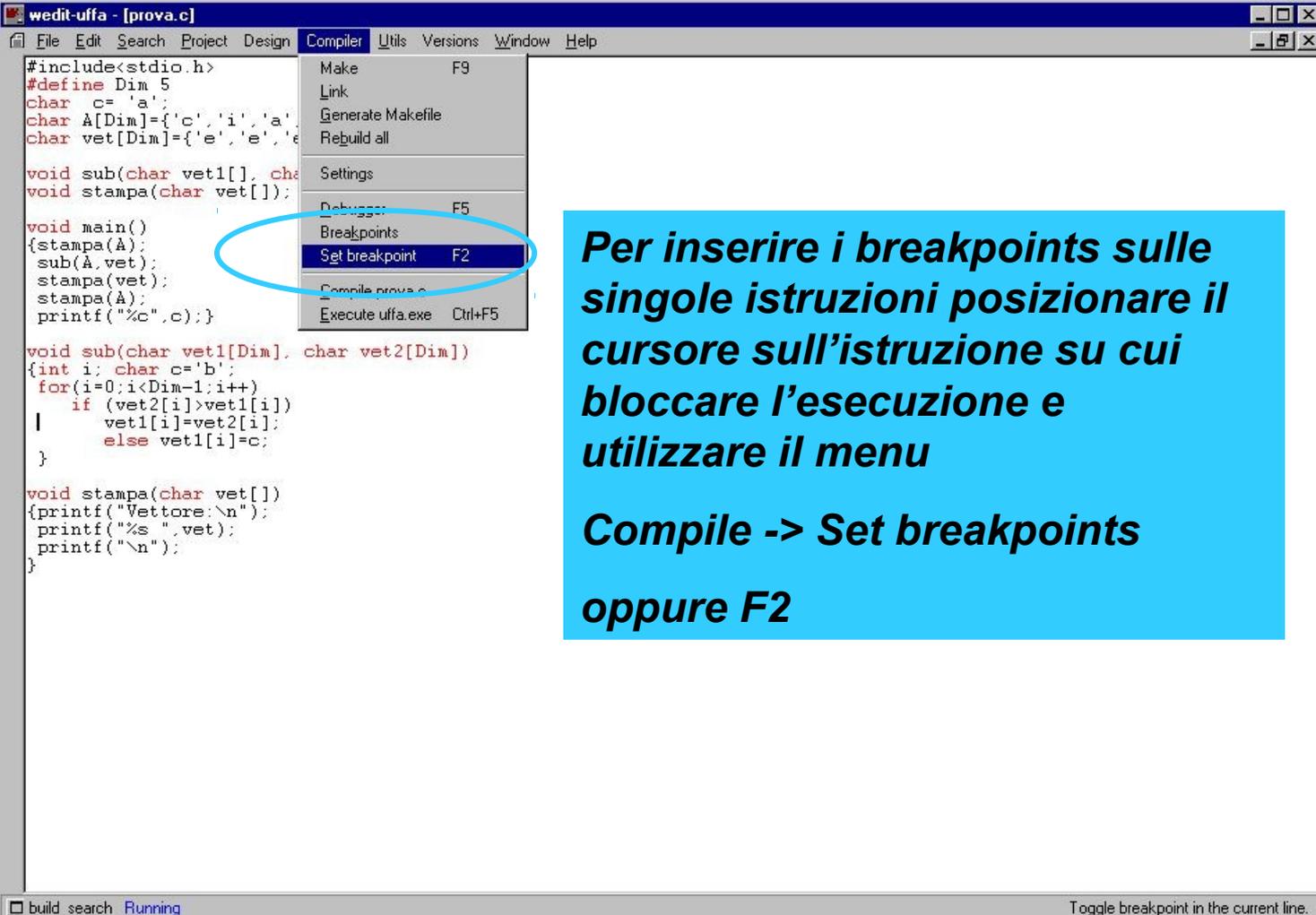
Erase

Clear all

build search Running

sub 21:2

LCC – DEBUGGER (4)



The screenshot shows the LCC debugger window with the 'Compiler' menu open. The 'Set breakpoint' option is highlighted with a blue oval. The code in the background includes a main function that calls 'sub' and 'stampa' functions, and a 'sub' function that iterates over an array and compares elements.

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

LCC – DEBUGGER (5)

```
#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");
}
```

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

Vengono visualizzati i valori delle variabili

```
i=0
vet2[0]=101 'e'
vet1[0]=99 'c'
c=98 'b'
Dim=5
```

sub 21:2

LCC – MENU DEBUG

- Il menu *Debug* compare quando il debugger e' attivo
- Alcuni item di menu:
 - **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.

LCC – MENU DEBUG

```
#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