UNIVERSITÀ DEGLI STUDI DI ROMA
LA SAPIENZA


FACOLTÀ DI INGEGNERIA

# Guida all'uso dell'ambiente di sviluppo DEV-C++ 4 per la programmazione in C

Simone Fratini
[fratini@dis.uniroma1.it]

Il presente documento è una versione preliminare (Gennaio 2003) distribuito agli studenti del corso di Sistemi di Elaborazione. Qualsiasi segnalazione o suggerimento utile al miglioramento dello stesso sarà ben gradito.

ANNO ACCADEMICO 2002/2003

# Guida all'uso dell'ambiente di sviluppo "Dev-C++ 4" per la programmazione in C

Questo documento ha lo scopo di illustrare le caratteristiche generali di un ambiente di sviluppo per programmi scritti in liunguaggio C. Ci si concentra su un ambiente in particolare, il "Dev-C++", ma i concetti esposti sono molto generali e facilmente adattabili anche ad altri ambienti di sviluppo disponibili in commercio, sia per lo stesso linguaggio di programmazione (Turbo C, Visual Studio, Code Warrior, ecc..) sia per altri linguaggi (Turbo Pascal per il Pascal o Kawa per Java ad esempio). Questa guida è uno strumento iniziale di supporto che non può sostituire l'esperienza che si acquisisce da un uso in prima persona dell'ambiente stesso.

## 1 Programmare in C

Un ambiente di sviluppo è un software che serve per scrivere ed eseguire programmi. Generalmente integra almeno tre funzionalità:

1. **Editor**: è presente un *editor*, ovvero un programma che coadiuva l'utente nella scrittura del codice. Gli editor degli ambienti di sviluppo sono progettati per indentare automaticamente il codice e per evidenziare con colori diversi parti del codice sintatticamente distinte (costanti, identificatori, ecc..);

2. **Compiler**: un *compilatore* è un programma che traduce un codice scritto in un linguaggio di alto livello (C nel nostro caso) in linguaggio macchina, ovvero in un nuovo codice direttamente comprensibile dall'elaboratore. Ogni ambiente di sviluppo è basato su un compilatore del linguaggio per il quale è pensato, compilatore che può essere attivato automaticamente dall'interno dell'ambiente;

3. **Debugger**: spesso sono messe a disposizione del programmatore delle funzionalità per monitorare l'esecuzione del programma al fine di individuare eventuali errori.

Un ambiente di sviluppo utilizza un **progetto** per gestire un insieme di *files sorgente*, normali files di testo che contengono il programma scritto in linguaggio C. Un programma è costituito da un insieme di *funzioni* che, idealmente, prendono dati in ingresso e restituiscono un risultato in uscita (vedi fig. 1). Le funzioni che compongono un programma possono essere scritte in file separati per facilitare operazioni di debug e per accelerare i tempi di compilazione (in caso di errore non si ricompila tutto il programma ma solo il file modificato). Il progetto serve appunto per facilitare la gestione di programmi spezzati in diversi file sorgente.

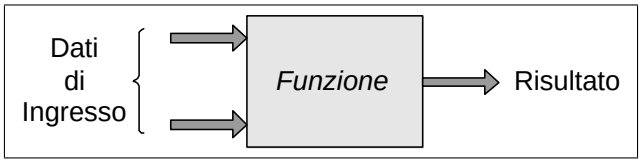


Figura 1: Concetto di funzione

Una funzione in linguaggio C ha un nome, un elenco di parametri di ingresso, un corpo (dove i parametri di ingresso vengono utilizzati per produrre un risultato di uscita attraverso opportune istruzioni) ed un risultato di uscita. Ad esempio una funzione *Quadrato* che, preso in ingresso un numero intero $x$ restituisce in uscita $x^2$ è riportata in figura 2.

Deve esistere nel programma una funzione particolare, chiamata *main*, che indica il punto di inizio del programma (in effetti la prima istruzione della funzione *main* è la prima eseguita nel programma) e i cui paramnetri

di ingresso (se esistono) vengono “dall’esterno”, ovvero vengono specificati dall’utente quando fa partire il nostro programma. Un programma C è costituito da una funzione *main* nel corpo della quale vengono richiamate altre funzioni, che a loro volta possono usarne altre ancora, e così via.

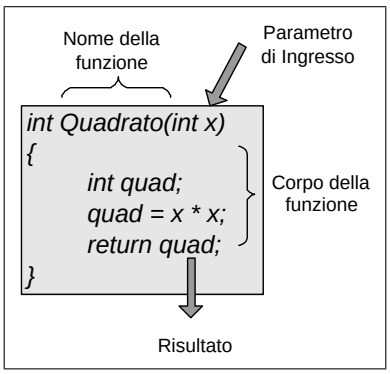


Figura 2: Funzione Quadrato

Le funzioni che si possono usare sono:

1. Quelle definite direttamente da noi nel programma;
2. Quelle di uso generale messe a disposizione dal linguaggio C e che possono essere usate come se le avessimo scritte noi (quando è disponibile una *funzione di libreria* per svolgere un determinato compito conviene in genere usarla, sia per risparmiare tempo sia perchè sono state scritte con particolare cura e possono quindi essere considerate esenti da errore ed efficienti).

Ogni file sorgente contiene una o più funzioni utili al programma. Questi files vengono *compilati* ad uno ad uno per produrre *files oggetto*: questi ultimi sono la traduzione in linguaggio macchina delle funzioni scritte in linguaggio C nel file. In questa fase si traducono le funzioni in linguaggio macchina e si rilevano *errori sintattici* eventualmente presenti nel file sorgente. Generalmente le funzioni codificate nel file sorgente utilizzano altre funzioni non definite nello stesso file (perchè le abbiamo scritte in un altro files sorgente o perchè fanno parte delle funzioni standard del C). I files oggetto prodotti quindi *non* sono eseguibili. Da notare che nel files sorgente possono essere introdotte delle *direttive al preprocessore*, ovvero dei comandi che non fanno parte del programma C ma che sono istruzioni per il compilatore (sono quei comandi che iniziano con #). Le più frequenti sono le direttive *#include*: quando il compilatore ne trova una sostituisce la direttiva con il files relativo e procede nella compilazione.

Più files oggetto vengono *linkati*[1] tra di loro per produrre il *files eseguibile*, ovvero il programma vero e proprio. Compito del linker è (grosso modo) quello di mettere insieme il codice oggetto di *tutte* le funzioni (scritte da noi o standard del C) per produrre un programma. In questa fase vengono rilevati degli errori relativi alle chiamate di funzione: il linker si accorge se viene chiamata una funazione che non esiste o se viene invocata una funzione con parametri diversi da quelli che ci si attende.

Il file eseguibile generato dal linker può essere eseguito in un ambiente di debug, per individuare eventuali errori *run time*, ovvero quegli errori che si verificano solo quando il programma sta funzionando (il programma è sintatticamente corretto e le chiamate di funzione sono a posto, ma non fa quello che dovrebbe fare). Nella fase di debug si eseguono le istruzioni una ad una (*tracing del programma*) e si monitorizzano i valori delle variabili impiegate, per cercare valori anomali.

L’intero processo (schematizzato in figura 3) può essere inquadrato all’interno di un ambiente di sviluppo.

# 2 L’ambiente Dev-Cpp

L’ambiente di sviluppo Dev-Cpp è un ambiente pensato per i linguaggi C e C++ (in appendice in fondo al documento ci sono informazioni relative all’installazione del software).

[1] Il nome viene dal programma, il linker o “collegatore” in italiano, che si occupa di questa operazione.

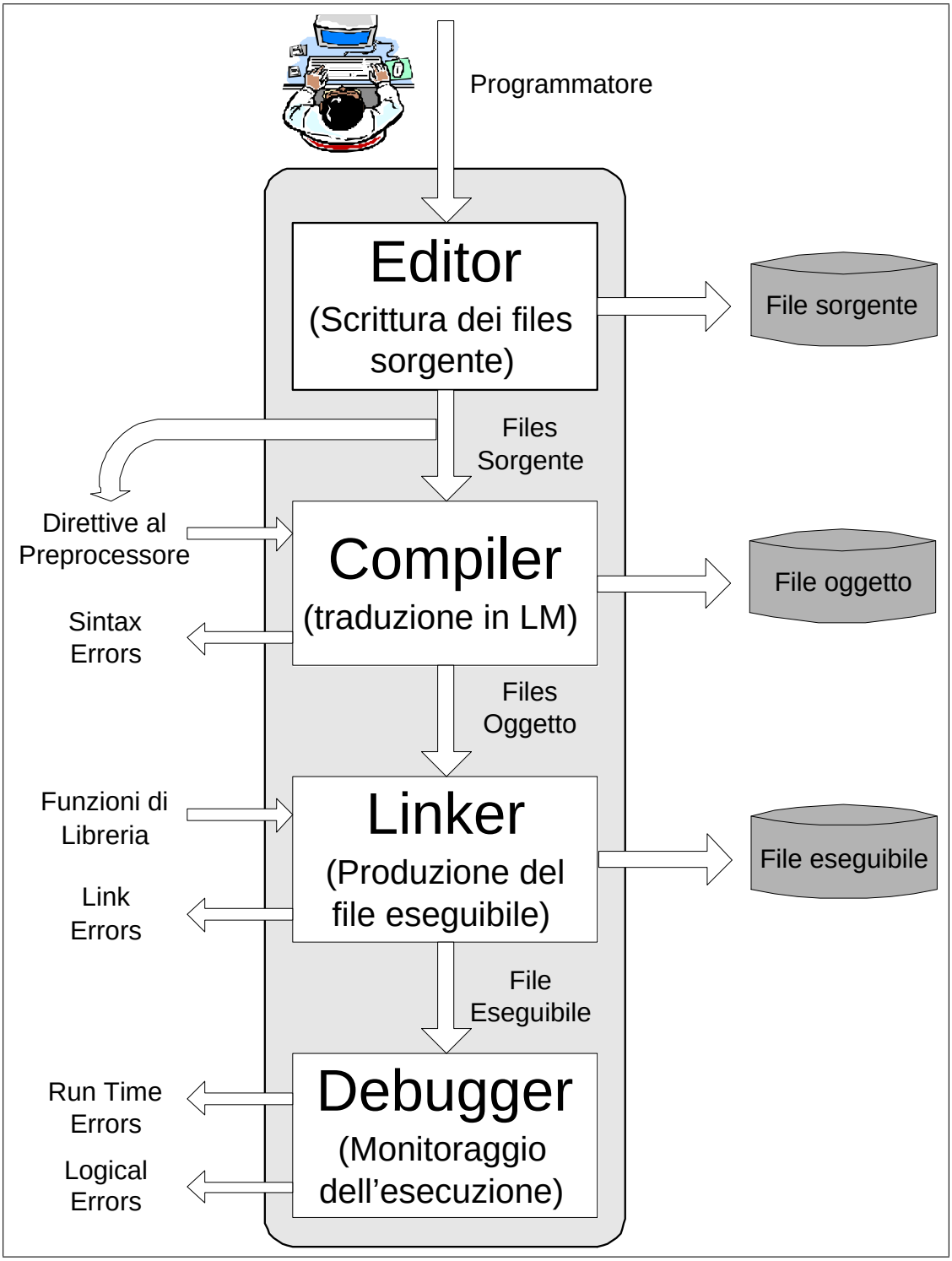


Figura 3: Processo

La finestra dell'applicazione (fig. 4) è normalmente divisa in tre zone, in particolare:

- Una zona dove vengono evidenziati i files sorgente che compongo il progetto;
- Una zona dove vengono editati i files sorgente;
- Una zona dove si possono leggere informazioni restituite dal compilatore e dal linker.

Nella figura 4 sono poi indicati anche:

- La barra dei menu;
- La barra degli strumenti (visualizzata su due righe);
- I pulsanti di gestione delle finestre che permettono di ridurre a icona (il primo da sinistra), ridimensionare (il secondo) o chiudere (il terzo) l'ambiente di sviluppo.

I pulsanti presenti nella barra degli strumenti attivano le funzioni più comuni e di più frequente utilizzo (funzioni attivabili anche tramite i menu, dove risultano però più "nascoste"). I pulsanti di maggior interesse sono:

- **Nuovo Progetto** e **Apri Progetto** per creare un nuovo progetto o aprirne uno precedentemenete creato e salvato su disco (Un file di progetto è un file con estensione .dev). Queste funzioni sono attivabili anche dal menu File;
- **Nuovo File**, **Salva File** e **Chiudi File** per creare, salvare e chiudere files sorgente. I files sorgente appartengono ad un progetto indipendentemente dal fatto che siano "aperti" o meno. Quando sono "aperti" sono visualizzati nella zona delle finestre di editing e possono essere modificati (chiameremo *attivo* il file che si sta editando). Quando si chiude un file questo continua a far parte del progetto e viene compilato e linkato come i files aperti. Quando si crea un files questo viene automaticamente aggiunto al progetto ma *non* viene salvato. Queste funzioni sono attivabili anche dal menu File;
- **Stampa file** per stampare un file sorgente. Questa funzione è attivabile anche dal menu File;
- **Annulla Operazione**, **Trova** e **Vai alla linea** per le operazioni di editing: il primo consente di annullare le operazioni effettuate (dalla più recente all'indietro); il secondo consente di cercare delle stringhe nel file sorgente e il terzo posiziona il cursore del file sorgente aperto alla riga indicata nell'apposita finestra di dialogo. La prima funzione è attivabile anche dal menu Edit, la seconda e la terza dal menu Search;
- **Compila**, **Esegui**, **Compila ed Esegui**, **Rebuild** e **Debug** per le operazioni di compilazione, esecuzione e debug del codice. Queste funzioni sono attivabili anche dal menu Execute;
- **Aggiungi file** ed **Elimina file** per aggiungere un file sorgente (già creato e salvato sul disco) ad un progetto. L'eliminare un file dal progetto implica che questo file sorgente non verrà più coinvolto nelle operazioni di compilazione e linkaggio del progetto, ma *non* viene fisicamente cancellato dal disco. Queste funzioni si trovano anche nel menu Project;
- **Risultato della Compilazione** apre la finestra con i risultati dell'ultima compilazione effettuata. La funzione è attivabile anche dal menu Tools;
- **Help** apre il menu di Aiuto. Queste funzioni sono contenute nel menu Help;
- **Nuovo...** apre un menu dal quale è possibile creare un nuovo progetto e un nuovo file sorgente.
- **Inserisci...** apre un menu dal quale è possibile inserire nella posizione corrente del cursore (nel file sorgente attivo) un commento di intestazione, la data e l'ora, una funzione main al codice (viene generata con una sola istruzione: return 0), ecc...
- **Bookmarks** apre un menu per creare (Toggle) o per posizionarsi (Goto) su un *bookmark*, ovvero un segnalibro. I bookmark sono molto comodi per segnare una riga di codice alla quale si vuol tornare in un secondo tempo senza doverla ricercare. Si possono posizionare fino a 10 bookmark (o con il menu attivabile col bottone Toggle Bookmark o premendo CTRL+n, dove n va da 0 a 9). Ci si può posizionare sulle righe dove si era posto il bookmark o con il menu Goto Bookmark o premendo ALT+n. Un bokmark viene visualizzato nella finestra di editing del file con un numeretto posto a sinistra della riga (vedi fig. 6).

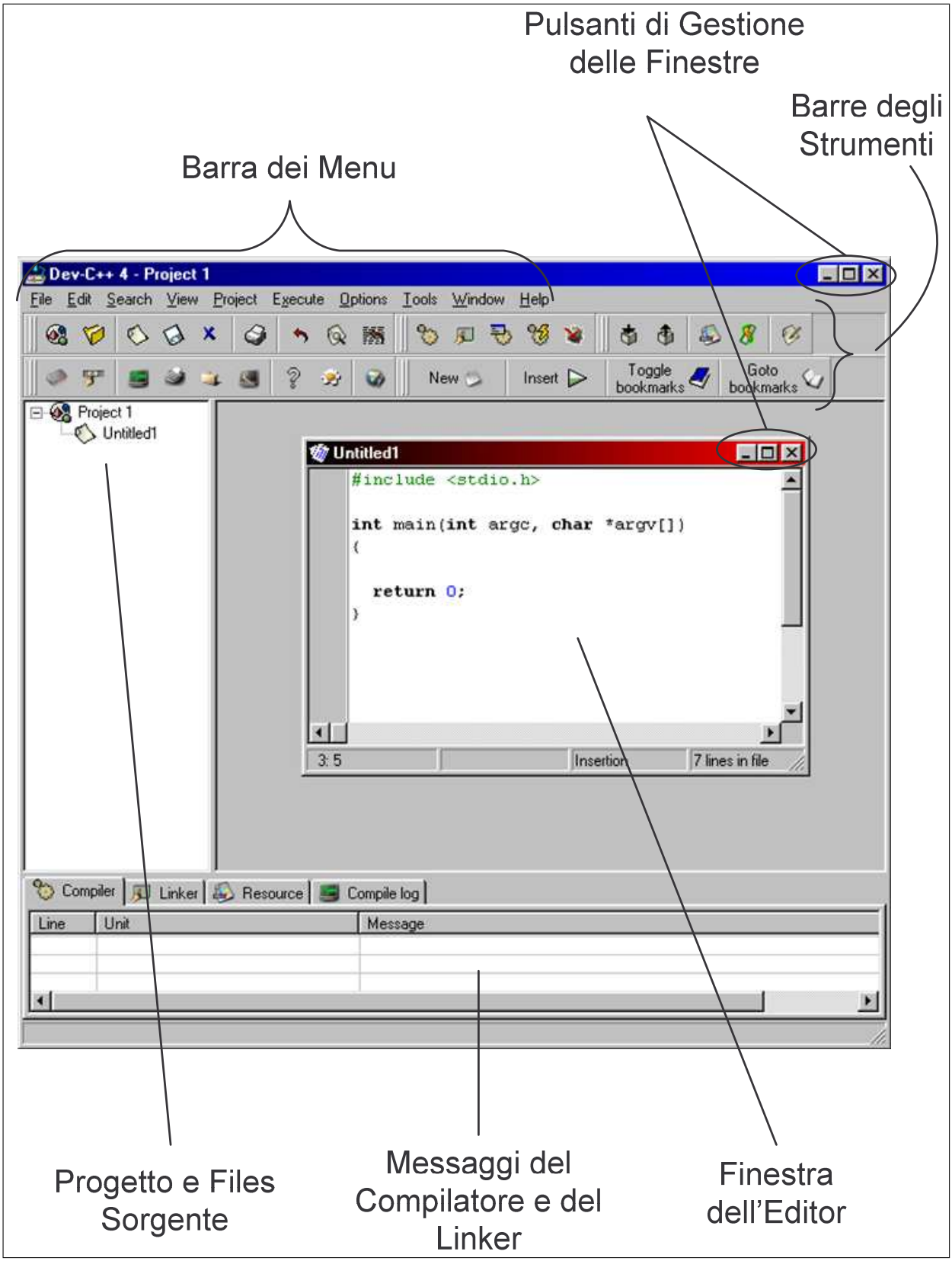


Figura 4: L'ambiente Dev-Cpp

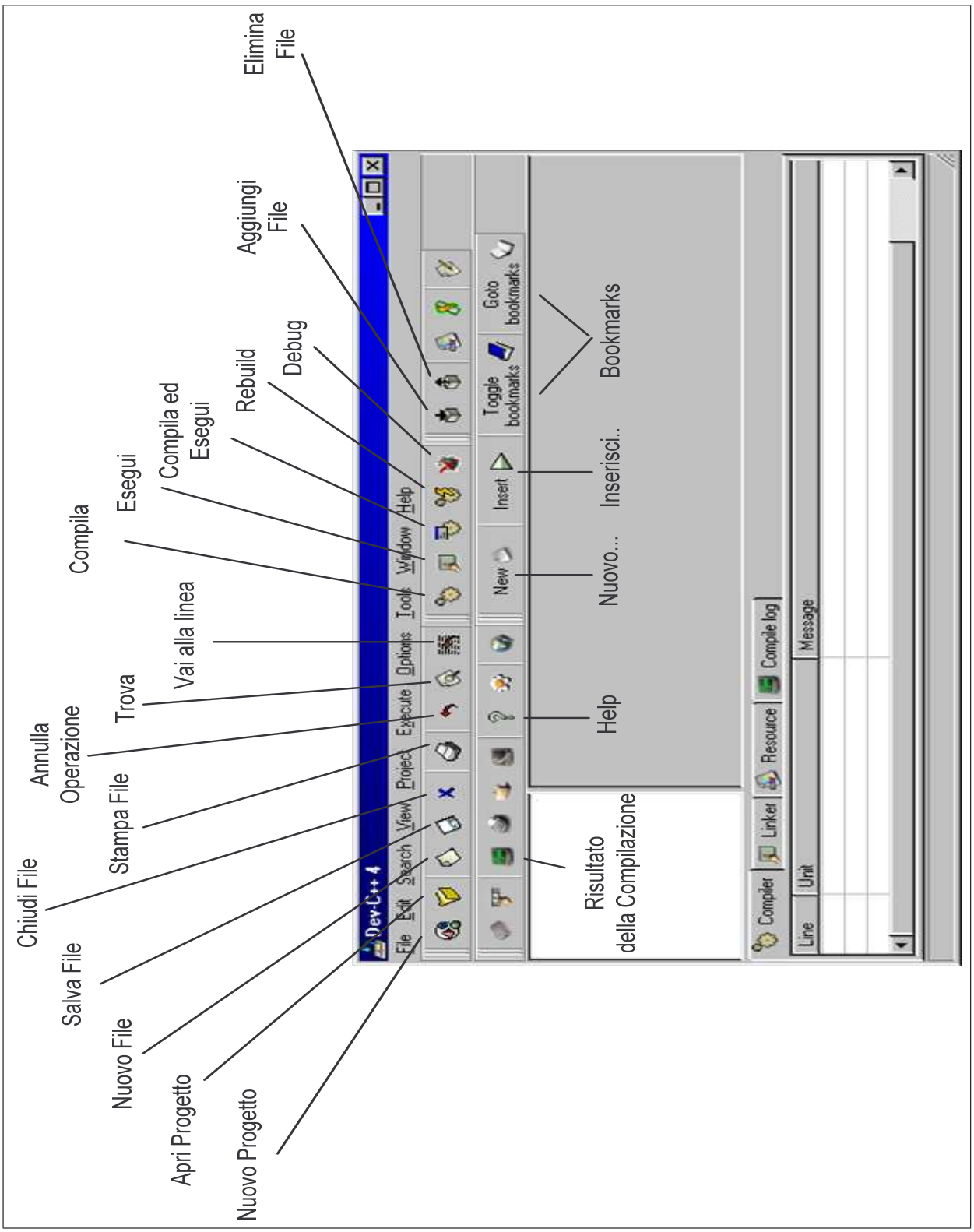


Figura 5: La barra degli strumenti

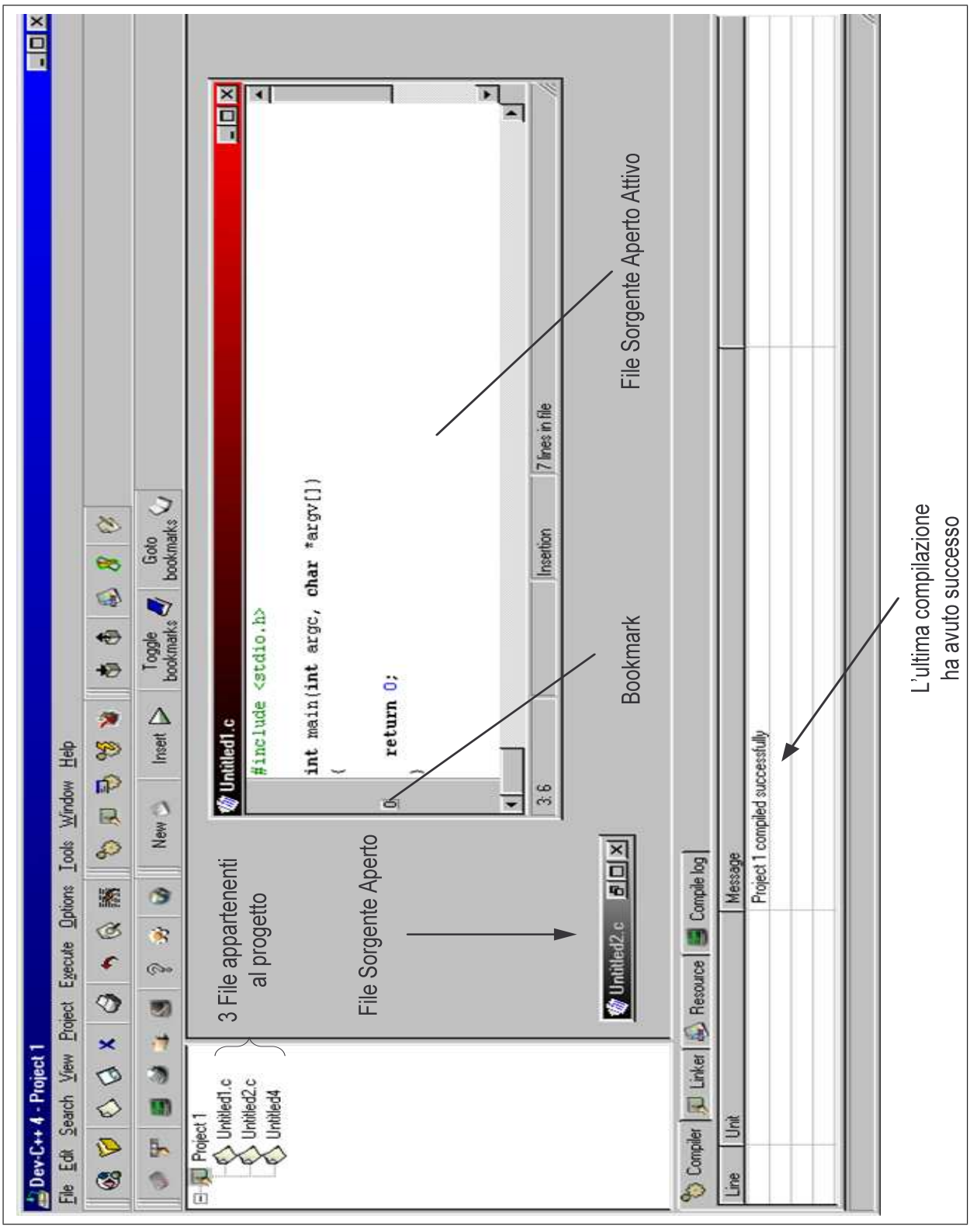


Figura 6: Files e Bookmark

In figura 6 possiamo vedere nella finestra del progetto come “Project1” sia costituito da tre files:“Untitled1.c”, “Untitled2.c” e “Untitled4”. Nella zona di input vediamo “Untitled1.c” aperto e attivo, dove si è posizionato un bookmark sulla riga dell’istruzione “return 0”[2].

# 3 Realizzazione di un programma

Nell’ambiente di sviluppo si può realizzare un programma costituito da diversi file sorgente (più frequente nei casi pratici) o da un solo file che contiene tutto il programma. Ove necessario distingueremo i due casi.

## 3.1 Creazione

Una volta aperto l’ambiente di sviluppo si può procedere alla creazione di un nuovo programma da scrivere o all’editing di un programma già precedentemente impostato. In questo corso ci concentreremo su programmi scritti in linguaggio C (ricordiamo che l’ambiente permette anche la programmazione in lunguaggio C++) per creare applicazioni a *console*, ovvero concepite per una interazione testuale con l’utente (senza quindi l’utilizzo di finestre o di interfacce tipo Windows). Possiamo procedere come visto sia gestendo un progetto (caso più generale) sia scrivendo tutto il codice in un unico file. Analizziamo i due casi.

### 3.1.1 Creazione di un Progetto

Per creare un nuovo progetto:

1. Si seleziona **Nuovo Progetto**;
2. Nella finestra di dialogo (vedi fig. 7) per scegliere che tipo di progetto si vuole creare si deve selezionare sempre “Console Application” e “C project” come in figura e si preme il bottone “OK”;
3. Si inserisce un nome per il progetto nella successiva finestra di dialogo e si preme il bottone “OK” (viene automaticamente aggiunto il suffisso .dev al file);
4. Si sceglie una cartella sul disco dove posizionare i files del progetto e si preme il bottone “Salva”.

Questa procedura genera un nuovo progetto con un solo files dal nome “Untitled1”. Questo file contiene solamente il *template* di base per la crezione di una semplicissima applicazione (vedi fig. 8).

E’ opportuno rinominare il file e salvarlo subito una prima volta. A questo scopo si deve:

1. Cliccare con il pulsante destro del mouse sul nome del file (“Untitled1”);
2. Dal menu che si apre selezionare “Rename file”;
3. Nella finestra di dialogo che si apre immettere un nome per il file e premere “OK”(viene automaticamente aggiunto il suffisso .c al nome scelto);
4. Cliccare (con il sinistro) sul pulsante “Salva File” nella barra degli strumenti;
5. Scegliere una cartella dove posizionare il file (noi lo metteremo sempre nella stessa cartella dove abbiamo creato il progetto) e premere “Salva”.

### 3.1.2 Creazione di un programma su un singolo file

E’ possibile creare, compilare e gestire un singolo file sorgente al di fuori di un progetto. Si deve cliccare su **New source file** quando non c’e’ alcun progetto aperto (altrimenti il nuovo file viene automaticamente aggiunto al progetto aperto). Viene creato un semplice framework contenente la sola funzione *main*. [3]

[2] In un contesto di pratico utilizzo è buona norma evitare nomi generici come *Untitled* per i file rinominando quindi i progetti e i file sorgente generati in automatico dall’ambiente con questi nomi.

[3] Il file creato contiene l’istruzione *system(“PAUSE”)* per permettere la visualizzazione dell’output del programma in modo testo.

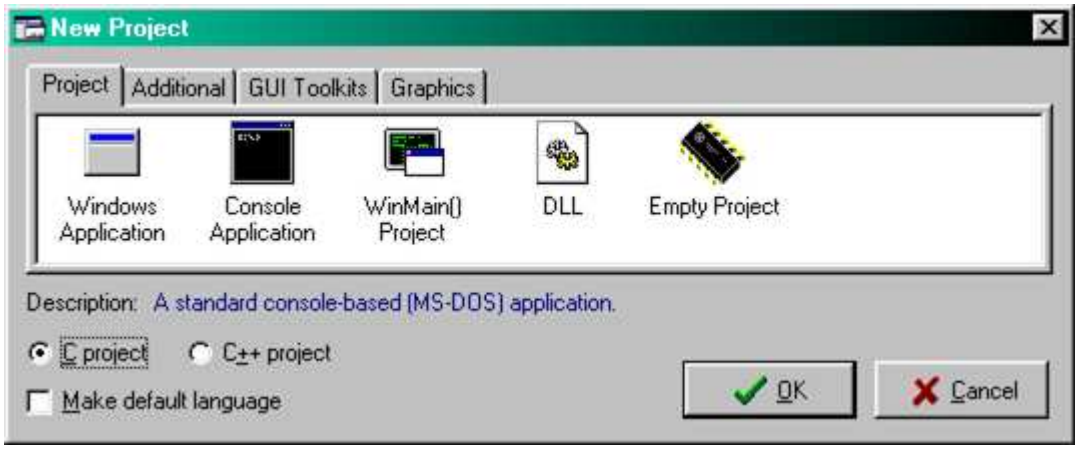


Figura 7: Nuovo Progetto

```
#include <stdio.h>
int main(int argc, char *argv[])
{       return 0;
}
```

Figura 8: Template di base

## 3.2 Editing dei file

Non è compito di questa dispensa entrare nel dettaglio del funzionamento dell'editor dell'ambiente. Si tenga presente che il suo funzionamento assomiglia moltissimo a quello di un qualsiasi semplice editor di testi per Windows. Richiamiamo alla memoria solamente alcuni semplici punti:

1. le frecce di spostamento consentono di posizionare il cursore nel punto sul quale si intende lavorare;
2. si possono selezionare parti di testo posizionandosi all'inizio del brano e, tenendo premuto il tasto *shift* (quello con il simbolo ⇑), spostarsi fino alla fine del testo che si intende selezionare. Il testo selezionato appare in negativo. L'operazione può essere anche effettuata con il mouse tenendo premuto il tasto sinistro;
3. per *copiare* in un altro posto un testo selezionato si procede con il *Copy* (premendo Ctrl+C o dal menu Edit) e, dopo essersi posizionati nel punto in cui si vuole copiare il testo, si procede con il *Paste*(premendo Ctrl+V o dal menu Edit);
4. per *spostare* in un altro posto un testo selezionato si procede con il *Cut* (premendo Ctrl+X o dal menu Edit) e, dopo essersi posizionati nel punto in cui si vuole copiare il testo, si procede con il *Paste*(premendo Ctrl+V o dal menu Edit);
5. per *cercare* una stringa nel testo si usa la funzione CTRL+F (o la voce Find.. nel menu Search). Una volta inserita la stringa da cercare nell'apposito spazio, si avvia la ricerca premendo Invio o cliccando sul pulsante Successivo. La prima occorrenza a partire dalla posizione del cursore (verso il basso o verso l'alto a seconda della direzione scelta) verrà evidenziata. Premendo ripetutamente il pulsante Successivo (o premendo F3) verranno via via evidenziate tutte le altre occorrenze. Ci sono due opzioni:
   - Parola Intera: se selezionata *non* cerca le sottostringhe;
   - Maiuscole/Minuscole: se selezionata rende la ricerca case sensitive (in questo caso ad esempio "Roma" e "roma" sono due parole diverse).

   Per chiudere la finestra di ricerca cliccare sul pulsante di chiusura della finestra (la X in alto a destra) o sul pulsante Annulla.

6. per *sostituire* una stringa con un altra nel testo si usa la funzione CTRL+R (o la voce Find and Replace.. nel menu Search). Si procede come per la funzione Find, ma bisogna ovviamente indicare la stringa da sostituire.

## 3.3 Salvataggio e recupero dei file

E' opportuno durante il lavoro *salvare di tanto in tanto* (cliccando su **Save Current File**) il proprio lavoro (ovvero le modifiche apportate al codice) onde prevenire malfunzionamenti, guasti e problemi vari che portebbero compromettere il lavoro effettuato.

I progetti o i files precedentemente salvati possono essere recuperati cliccando su **Open project**. Una volta aperto un progetto i singoli files che lo compongono possono essere aperti cliccando sul loro nome nell'elenco a sinistra.

## 3.4 Compilazione ed esecuzione

Per compilare, linkare ed eseguire un progetto deve essere stato preventivamente aperto (o creato ex-novo).

1. Cliccare su **Compile Project**;
2. Se la compilazione va a buon fine (in fig. 9 vediamo il risultato della compilazione del template di figura 8) il programma può essere eseguito (cliccando su **Execute**) [4] o si può semplicemente tornare all'ambiente di sviluppo cliccando su **Continue**.
3. Cliccando su **Parameters** compare un campo di testo dove è possibile inserire eventuali parametri a riga di comando da passare al programma.

Cliccando invece su **Run Project** si esegue un programma precedentemente compilato. **Compile and Run Project** combina le due funzioni (compilazione ed esecuzione).

Quanto detto per la compilazione e l'esecuzione di un progetto vale anche nel caso si stia lavorando con un singolo files sorgente.

Se invece la compilazione non va a buon fine vuol dire che il programma contiene qualche errore *sintattico*: in questo caso il file oggetto non viene prodotto e compare un[5] messaggio di errore nella finestra di output in basso dedicata al compilatore. L'ambiente di sviluppo ci facilita la vita: con un doppio click sulla riga che nella finestra di output spiega l'errore ci porta subito a quella riga del file sorgente, che viene evidenziata per maggior comodità. Ovviamente se il programma contiene errori sintattici e la compilazione non va a buon fine non sarà possibile eseguire il programma, e il pulsante *Execute* sarà disattivato (in grigio chiaro).

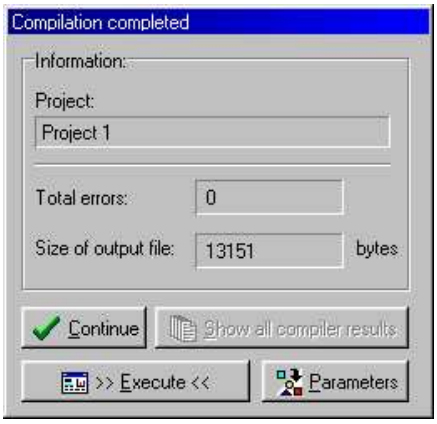


Figura 9: Compilazione

[4] Questa opzione non è disponibile se in nessun file del progetto c'e' una funzione *main*.

[5] Un errore può generarne a cascata molti altri. Una buona idea è quella di cercare di correggere il primo, ricompilare e vedere se gli altri errori spariscono (dimenticare di chiudere una parentesi può facilmente generare decine di errori ad esempio!)

## 3.5 Debug

Un programma oltre a contenere errori sintattici, può contenerne anche di *semantici*: viene cioè compilato e può essere eseguito, ma non fa quello che dovrebbe. Le funzionalità di debug ci aiutano nella ricerca di errori che si manifestano mentre il programma viene eseguito.

Concettualmente si usano dei tools che permettono di arrestare automaticamente l'esecuzione del programma dopo ogni istruzione. In pratica si esegue il programma *passo-passo*, monitorando i valori delle variabili e dei registri di sistema alla ricerca di valori strani o non previsti.

Procederemo all'illustrazione del tools per il debug del Dev-Cpp attraverso un esempio pratico. Consideriamo il programma riportato in figura 10: si inseriscono da tastiera dei numeri interi positivi dei quali viene visualizzata la somma quando si inserisce il valore -1 (valore sentinella).

```
#include <stdio.h>
int main()
{
    int numero, totale = 0;
    printf ("Inserisci il primo numero (-1 per terminare): ");
    scanf ("%d",&numero);

    while (numero != -1)
    {
        totale = totale + numero;
        printf("Altro numero: ");
        scanf ("%d",&numero);
    }

    printf ("\nTotale : %d\n",totale);

    system("PAUSE");
    return 0;
}
```

Figura 10: Programma di esempio

Dopo aver scritto e compilato il programma, si può lanciare il debug con l'apposito pulsante (vedi fig. 5). Compare un ambiente (fig. 11) all'interno del quale è possibile eseguire il programma e controllare i valori delle variabili. L'istruzione che sta per essere eseguita viene evidenziata in verde. Il programma può essere avviato premendo il pulsante **Run**. Questo pulsante diventa un segnale di STOP quando si esegua qualche istruzione che richieda un input dall'utente. In questo caso per proseguire si deve cliccare sulla finestra del programma e inserire i dati.

I pulsanti di avanzamento presenti nella barra degli strumenti del debugger servono per controllare l'esecuzione passo-passo del programma. In particolare[6]:

- Il pulsante *Step* (la funzione è attivabile premendo il tasto "S") esegue l'istruzione evidenziata. Se l'struzione richiede un input da parte dell'utente il debugger si blocca e il bottone "Run" si trasforma nel bottone "Stop". Per procedere è necessario l'input nella finestra dell'applicazione. Se l'instruzione prevede una chiamata a funzione il debug continua nel codice della funzione;
- Il pulsante *Next* (la funzione è attivabile premendo il tasto "N") attiva una funzionalità molto simile a quella del pulsante Step, con l'unica differenza che nel caso l'istruzione preveda una chiamata a funzione, questa viene eseguita interamente e il debug procede dalla successiva istruzione della funzione chiamante;
- Il pulsante *Finish* (la funzione è attivabile premendo il tasto "F") esegue tutte le successive istruzioni fino al punto di ritorno della funzione corrente;

[6] Posizionando il puntatore del mouse su un pulsante senza prememrlo, compare un *tooltip*, ovvero una piccola etichetta gialla che indica il nome del pulsante.

- Il pulsante *Continue* (la funzione è attivabile premendo il tasto “C”) permette di procedere per salti nel debug: nella finestra del debugger, cliccando con il pulsante sinistro del mouse su una riga di codice, si può fissare un *breakpoint*, ovvero un punto di arresto, in corrispondenza di quella riga. La pressione del pulsante *Continue* esegue insieme tutte le istruzioni dal punto in cui viene premuto fino al primo breakpoint incontrato (Se prima del preakpoint si incontra una istruzione di input sarà comunque necessario introdurre il dato per proseguire).

Ci sono poi sette funzionalità attivabili con i relativi pulsanti. Nell’ordine:

- Il pulsante *Register* visualizza un riepilogo dei valori contenuti nei registri del microprocessore;
- Il pulsante *Memory* mostra i byte contenuti nelle celle di memoria a partire dall’indirizzo indicato in alto a sinistra nella finestra;
- Il pulsante *Stack* mostra la situazione dello stack di attivazione delle funzioni;
- Il pulsante *Watch expressions* attiva una funzionalità che permette di impostare delle espressioni matematiche che coinvolgono le variabili del programma e di monitorarne il valore assunto;
- Il pulsante *Local Variables* permette di monitorare i valori assunti dalle variabili locali della funzione di cui si sta effettuando il debug;
- Il pulsante *Breakpoints* mostra un riepilogo del posizionamento dei breakpoints;
- Il pulsante *Console* consente di utilizzare il debugger in modalità testo.

La funzionalità più interessante è quella che ci permette di monitorare i valori assunti dalle variabili locali. Questo risultato può anche essere ottenuto posizionando il puntatore del mouse sul nome di una variabile direttamente nel codice visualizzato nella finestra del debugger: compare una etichetta gialla che mostra il valore correntemente assunto da quella variabile.

Notiamo infine che la finestra del debugger mostra in alto a destra l’indirizzo di memoria dell’istruzione evidenziata e il numero della riga di codice in cui compare.

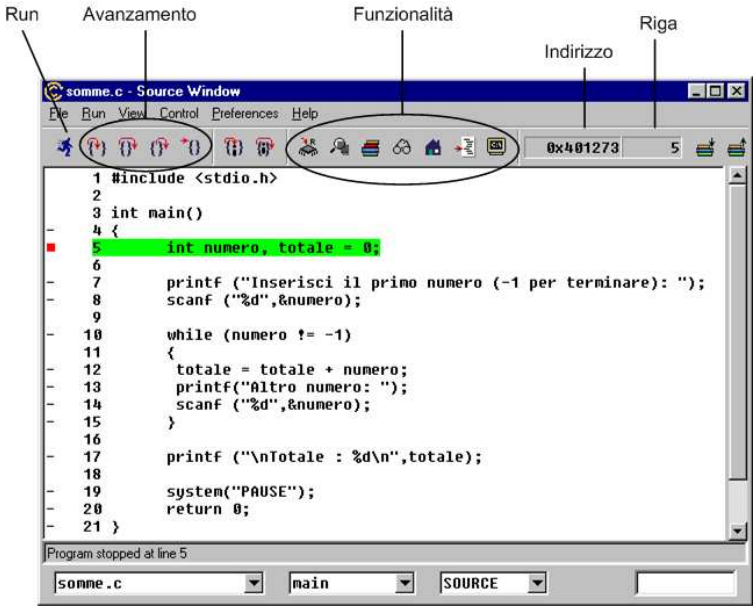


Figura 11: Debugger

# Installazione del software

Per installare il software si devono avere a disposizione i files **devcpp4.zip** (l'ambiente di sviluppo vero e proprio) e **insight5_win32.zip** (il plug-in per l'interfaccia grafica delle funzioni di debug). Per chi non li avesse sono reperibili sul sito del corso.

Si segua questa procedura di installazione:

1. Cliccando due volte (con il pulsante sinistro del mouse) sul primo file (devcpp4.zip) si apre il programma di decompressione dei files (è necessario che sul computer sia installato il WinZip) che mostra un elenco di files;
2. Cliccando sul programma **Setup.exe** parte un programma di supporto di installazione dell'ambiente di sviluppo. La procedura di installazione è semi-automatica: è sufficiente accettare tutte le opzioni proposte (cliccando sui pulsanti Ok, Yes o Next a seconda dei casi) per portarla a termine. La procedura copia il software nella cartella "C:\Dev-C++";
3. Al termine della procedura l'ambiente di sviluppo può essere avviato dal pulsante *Start* di Windows (con Start → Programmi → Dev-C++ → Dev-C++).

Fatto questo si deve procedere all'installazione del plug-in [7] per l'interfaccia grafica delle funzionalità di debug dell'ambiente (che nella versione base presenta un'interfaccia testuale per questo genere di operazioni!). A tale scopo si segua questa procedura:

1. Cliccando due volte (con il pulsante sinistro del mouse) sul secondo file (insight5_win32.zip) si apre il programma di decompressione che mostra un elenco di files;
2. Occorre decomprimere il contenuto del file nella cartella "C:\Dev-C++". Cliccando sul pulsante "Extract" del WinZip si selezioni la cartella "C:\Dev-C++" o immettendone direttamente il nome o selezionandola dalla finestrella di browse (vedi fig. 12);
3. Ad ogni domanda *Replace file...With file...* rispondere sempre "Yes".

Al termine di queste due procedure l'ambiente di sviluppo è pronto per essere usato. Occorre però predisporlo in modo che vengano incluse dal linker nel file eseguibile alcune informazioni utili per il debug dell'applicazione. A tale scopo, dal menu **Option** si scelga **Compiler Option**. Ora, cliccando sulla linguetta **Linker** (l'ultima a destra) si faccia comparire il segno di spunta sulla seconda opzione (quella con la scritta *Generate debugging information*. Ora l'ambiente introdurrà automaticamente le informazioni per il debug nelle applicazioni generate.

Alle esercitazioni e sul materiale distribuito si utilizzerà lo stile *Gnome* per le icone. Si può settare dal menu Option → Icon Style.

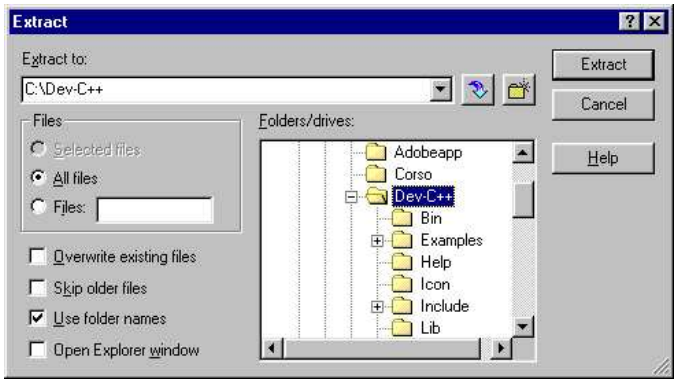


Figura 12: Estrazione files

[7] Un *plug-in* è un software aggiuntivo che migliora o estende le funzionalità di un altro software.