Corso di Laurea in Matematica

Seminario C/C++

Introduzione a Code::Blocks

Lorenzo Dusty Costa
Federico Paolo Kircheis

Università degli Studi di Milano
Dipartimento di Matematica

19 Ottobre 2011

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Cos'é un'IDE

IDE = Integrated Development Environment

Consiste in:

- Editor
- Compilatore
- Tool di Building Automatico
- Debugger

É necessario usare un IDE per programmare? No!

É utile usare un IDE per programmare? Sì!

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,. . .)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,...)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,...)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,...)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,...)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Perché?

Perché è utile usare un'IDE?

- Comoda gestione di programmi su più sorgenti
- Contiene tutti gli strumenti necessari per lo sviluppo di un programma (Editor, Compilatore, Debugger,...)
- Sviluppare buone tecniche per trovare e isolare gli errori
- Interfaccia Grafica
- Aumenta la velocità di scrittura (Autocompletamento, Consigli, Evidenziazione Errori, Scorciatoie da Tastiera)
- Ne esistono molti, per diversi linguaggi, su varie piattaforme, open source e a pagamento

Oggetti con cui abbiamo a che fare

Workspace : cartella in cui vengono memorizzati i progetti su cui vogliamo lavorare

Project : insieme di uno o più sorgenti o header files; permette di tenere traccia di tutti i file di cui abbiamo bisogno in maniera organizzata

Sorgenti : file (.c oppure .cpp) che contengono il codice del programma

Header file : file utilizzati per creare una libreria, cioè un insieme di funzioni (stdio.h., math.h, . . .)

Creare un Progetto

File $\rightarrow$ **New** $\rightarrow$ **Project...**

$\Downarrow$

New from template (Permette di scegliere il tipo di progetto)

$\Downarrow$

Console Application

$\Downarrow$

Scelta C/C++ $\rightarrow$ **Titolo del Progetto** $\rightarrow$ **Scelta Compilatore**

$\Downarrow$

Fine (Ritorna alla schermata iniziale)

Nota Bene:

Workspace $\rightarrow$ **Nome Progetto** $\rightarrow$ **Sources** $\rightarrow$ **main.c (.cpp)**

In automatico è stato creato un (elementare) main. In questo modo si evita di scrivere ogni volta l'intestazione del programma.

Creare un Progetto

File → **New** → **Project...**

⇓

New from template (Permette di scegliere il tipo di progetto)

⇓

Console Application

⇓

Scelta C/C++ → **Titolo del Progetto** → **Scelta Compilatore**

⇓

Fine (Ritorna alla schermata iniziale)

Nota Bene:

Workspace → **Nome Progetto** → **Sources** → **main.c (.cpp)**

In automatico è stato creato un (elementare) main. In questo modo si evita di scrivere ogni volta l'intestazione del programma.

Scrivere un Programma: Come aggiungere file al progetto?

- File già esistente:

  Project → Add files... → Cerca il file da aggiungere

- Nuovo File:

  File → New → Empty File

  Shortcut: Ctrl - Shift - N

Aggiungere al progetto? Sì \No (→ salvataggio)

Selezione Target:

Debug : permette di usare il debugger; crea anche un'eseguibile di grosse dimensioni

Release : non permette l'uso del debugger; crea un'eseguibile più leggero

Conviene lasciare selezionati entrambi in modo da poter usare il debugger per lo sviluppo del programma e avere comunque una versione "leggera" per uso personale o da rilasciare ad altri.

Scrivere un Programma: Come aggiungere file al progetto?

- File già esistente:

  Project $\rightarrow$ **Add files...** $\rightarrow$ **Cerca il file da aggiungere**

- Nuovo File:

  File $\rightarrow$ **New** $\rightarrow$ **Empty File**
  Shortcut: Ctrl - Shift - N

Aggiungere al progetto? Sì \No ($\rightarrow$ salvataggio)

Selezione Target:

Debug : permette di usare il debugger; crea anche un'eseguibile di grosse dimensioni

Release : non permette l'uso del debugger; crea un'eseguibile più leggero

Conviene lasciare selezionati entrambi in modo da poter usare il debugger per lo sviluppo del programma e avere comunque una versione "leggera" per uso personale o da rilasciare ad altri.

Scrivere un Programma: Come aggiungere file al progetto?

- File già esistente:

  Project $\rightarrow$ **Add files...** $\rightarrow$ **Cerca il file da aggiungere**

- Nuovo File:

  File $\rightarrow$ **New** $\rightarrow$ **Empty File**
  Shortcut: Ctrl - Shift - N

Aggiungere al progetto? Sì \No ($\rightarrow$ salvataggio)

Selezione Target:

Debug : permette di usare il debugger; crea anche un'eseguibile di grosse dimensioni

Release : non permette l'uso del debugger; crea un'eseguibile più leggero

Conviene lasciare selezionati entrambi in modo da poter usare il debugger per lo sviluppo del programma e avere comunque una versione "leggera" per uso personale o da rilasciare ad altri.

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cos'è un debugger?

debugging = de + bug + ing =
= azione ('ing') di eliminare ('de') gli errori ('bug')

Tecniche:

- Usando programmi specifici (**debugger**)
- Manualmente

Le fasi del debugging:

- Identificazione del bug
- Individuazione della componente in cui è presente il bug
- Individuazione della causa del bug
- Progettazione di una correzione per il bug
- Implementazione e testing della suddetta correzione

Cose da fare prima di usare il debugger

- Controllare che l'opzione **Produce debugging symbols [-g]** sia attiva

  Project → **Build Options**

- Nella scheda **Other Options** si possono aggiungere manualmente i comandi se non compaiono nella lista:

Esempio:

se si vuole dichiarare una variabile (di solito un contatore) all'interno di un ciclo è necessario usare lo standard C99 non attivo di default. Per far questo basta scrivere: **-std=c99**

- Controllare che sia selezionato

  Build → **Select Target** → **Debug**

- Aprire la finestra per il controllo delle varibili (**Watches**)

  Debug → **Debugging Windows** → **Watches**

Tecniche di debugging: Run to Cursor

1. Ci si posizione con il cursore sulla linea in cui si vuole che in programma si fermi
2. Si dice (**Shortcut: F4**) al debugger di eseguire il programma fino alla posizione del cursore (**breakpoint**)
3. Compare una shell che sarà vuota fino a che il programma non esegue un'istruzione che stampa a video qualcosa
4. Nella finestra **Watches** si possono osservare i valori delle variabili durante l'esecuzione del programma
5. Il triangolino giallo sulla sinistra indica che il programma si è fermato a quella linea

Tecniche di debugging: Breakpoints Multipli 1/2

1. Ci si posizione con il cursore sulla linea e si indica la linea in cui si vuole che in programma si fermi
   - **Debug** $\rightarrow$ **Toogle breakpoint** (**Shortcut: F5**)
   - Tasto sinistro del mouse a fianco al numero di linea
2. I breakpoint vengono evidenziati da un cerchio rosso sulla sinistra
3. Si possono inserire più breakpoint. Il programma si fermerà ogni volta che ne incontra uno.
4. Si dice al debugger di eseguire il programma fino alla posizione del breakpoint. (**Shortcut: F8**). Per passare al breakpoint successivo:

   Debug $\rightarrow$ **Continue** **(Shortcut: Ctrl - F7)**
5. Compare una shell che sarà vuota fino a che il programma non esegue un'istruzione che stampa a video qualcosa

Tecniche di debugging: Breakpoints Multipli 2/2

6. Nella finestra **Watches** si possono osservare i valori delle variabili durante l'esecuzione del programma
7. Il triangolino giallo dentro al cerchio rosso sulla sinistra indica che il programma si è fermato a quella linea
8. Per rimuovere un breakpoint, cliccare su di esso.

Si possono applicare le tecniche appena descritte per analizzare le variabili delle funzioni. Come?

1. Si applica una delle tecniche illustrate in precedenza
2. Nella finestra (**Watches**) ci possono vedere gli argomenti passati alla funzione
3. Per vedere i valore assunti da variabili create all'**interno** di una funzione:

Debug $\rightarrow$ **Step into** **(Shortcut: Shift - F7)**

Terminare il Debugging

Per terminare il debugging abbiamo due possibilità:

1. **Debug → Continue**
   Shortcut: Ctrl - F7
   ↓
   Il programma continua fino alla fine

2. **Debug → Stop debugger**
   ↓
   Si interrompe l'esecuzione del debugger

Cosa scegliere?

Solitamente è meglio usare il comando **Continue** fino a quando il programma non completa la sua esecuzione. Questo perché è meglio che il programma giunga alla sua naturale conclusione rispetto ad abortire l'esecuzione stessa. In ogni caso se il programma è entrato in un loop infinito oppure il programmatore è certo che l'uscita dal debugger è sicura, può utilizzare **Stop debugger**.