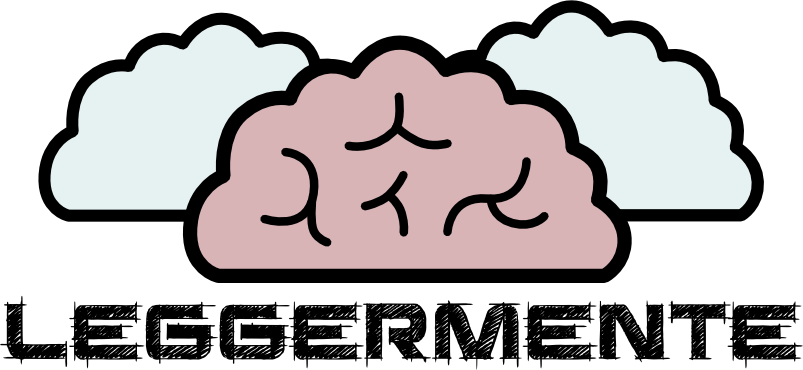


**ESAME DI STATO**

**a.s. 2013/2014**

**Area di progetto:**

****

**Realizzata da**: Canale Davide *(5B Informatica)*

Garabello Giacomo *(5B Informatica)*

**Docenti referenti**: Borra Giuseppe

Cambieri Oscar

*( Progetto sotto licenza GPL 2 )*

# Indice

[Indice 1](#_Toc389732004)

[Progetto “Leggermente” 2](#_Toc389732005)

[Introduzione 2](#_Toc389732006)

[ Cosa ci ha spinto? 2](#_Toc389732007)

[ Perché “Leggermente” 2](#_Toc389732008)

[Introduction *(English)* 3](#_Toc389732009)

[ What moved us to do it 3](#_Toc389732010)

[ Why “Leggermente” 3](#_Toc389732011)

[Progettazione 4](#_Toc389732012)

[ Studio dell’argomento 4](#_Toc389732013)

[ Scelte di programmazione 4](#_Toc389732014)

[Realizzazione “sistema di traduzione” 5](#_Toc389732015)

[ Pre-Elaborazione 5](#_Toc389732016)

[ Elaborazione 6](#_Toc389732017)

[ Compilazione 6](#_Toc389732018)

[ Problematiche 6](#_Toc389732019)

[Realizzazione “IDE di sviluppo” 8](#_Toc389732020)

[ Stampa di RTF 9](#_Toc389732021)

[ Numerazione delle linee di codice 9](#_Toc389732022)

[ Colorazione dinamica 11](#_Toc389732023)

[Manuale di programmazione 12](#_Toc389732024)

[Sintassi del linguaggio 12](#_Toc389732025)

[ I file e le loro estensioni 12](#_Toc389732026)

[ Commenti 12](#_Toc389732027)

[ Funzioni 13](#_Toc389732028)

[ Pacchetti 14](#_Toc389732029)

[ Variabili e Vettori 14](#_Toc389732030)

[ Comparatori logici e Simboli matematici 15](#_Toc389732031)

[Strutture di controllo 15](#_Toc389732032)

[ Strutture condizionali 15](#_Toc389732033)

[ Strutture d’iterazione 16](#_Toc389732034)

[ Indice nelle strutture ad iterazione 17](#_Toc389732035)

[Esempi 18](#_Toc389732036)

[ Bubble Sort 18](#_Toc389732037)

[Referenze Pacchetti 19](#_Toc389732038)

[ IO 19](#_Toc389732039)

[ VAR 19](#_Toc389732040)

[ MATE 19](#_Toc389732041)

[Risultati progetto 20](#_Toc389732042)

[Raccolta risultati 20](#_Toc389732043)

[Analisi risultati 20](#_Toc389732044)

[Ringraziamenti 20](#_Toc389732045)

# Progetto “Leggermente”

## Introduzione

### Cosa ci ha spinto?

Da alcuni anni ormai, nel nostro istituto, il biennio informatico utilizza un linguaggio di programmazione per facilitare l’apprendimento della logica necessaria all’istruzione di future figure tecniche.  
Il linguaggio di programmazione si chiama “Scratch” ed è stato ideato dal “Massachusetts Institute of Technology” con l’intento di facilitare l’apprendimento dei linguaggi di programmazione e della logica ai ragazzi.

Scratch è stato studiato per essere di semplice comprensione per i neofiti; i problemi legati a questo linguaggio di programmazione sono evidenti solamente nel passaggio a un linguaggio standard, poiché spesso per semplificare le procedure Scratch non rispetta alcuni paradigmi fondamentali dell’informatica; questo porta difficoltà ai ragazzi delle sezioni informatiche.

Noi studenti della classe quinta non abbiamo vissuto direttamente questo cambiamento, ma abbiamo potuto osservarlo nelle classi inferiori.

Per ovviare questo problema, abbiamo quindi tentato di estrapolare il maggior numero di semplificazioni da Scratch e di creare un linguaggio di programmazione che fosse, altresì facile da imparare, ma che rispettasse maggiormente i paradigmi comuni a tutti i linguaggi.

Con questa idea è quindi nato il progetto “Leggermente”.



### Perché “Leggermente”

Alcuni professori, inizialmente, si sono dimostrati dubbiosi e scettici riguardo all’utilità del progetto, infatti esistono moltissimi linguaggi di programmazione tradotti che semplificano la programmazione.  
Abbiamo subito chiarito che il nostro intento non è, infatti, quello di creare un linguaggio che sia semplice o veloce da scrivere, bensì quello di creare uno strumento di facile lettura e comprensione da parte degli studenti.

Abbiamo compreso che, per quanto semplice sia un linguaggio, perde la sua utilità didattica nel momento in cui risulta essere di difficile comprensione per i neofiti.

Il nome “Leggermente” cerca di comunicare lo scopo del progetto.

## Introduction *(English)*

### What moved us to do it

For several years the classes of the IT biennium of our institute have used a computer programming language to facilitate the understanding of the logic required for the future technical experts.  
This programming language is called "Scratch" and it was created by the "Massachusetts Institute of Technology" with the purpose of helping the students understand the programming languages and the computer logic.

Scratch was designed to be easily understandable by beginners; the problems related to this programming language are evident only by switching to the standard programming languages; to simplify the procedure Scratch does not often respect some fundamental paradigms of computer science; these simplifications leads the students of the IT sector to encounter some difficulties.

We, the students of the Fifth class, have not experienced this change directly, but we could observe it in the lower classes.

To overcome this problem, we attempted to extrapolate the largest number of simplifications from Scratch and create a programming language that is easy to learn, but that respects the most common paradigms off all programming languages.

With this idea in mind the project "Leggermente" was born.



### Why “Leggermente”

In the beginning some of our teachers were doubtful and skeptical about the utility of this project, because there are already so many translated programming languages that simplifie computer programming.

We immediately pointed out that our purpose was not to create a language that would be easier or quicker to write, but to create one that could be easier to be read and understood by the students.

We realized that, no matter how simple a school programming language is, it will be useless if not understandable by the students.

The name “Leggermente” tries to point out the purpose of the language.

## Progettazione

### Studio dell’argomento

Come sopra citato, il progetto “Leggermente” è nato poco alla volta con l’intento di semplificare la programmazione ai ragazzi  
Per questo motivo abbiamo compreso che il metodo più efficace per realizzare il linguaggio fosse in primo luogo comprende le problematiche di Scratch.

Di conseguenza abbiamo imparato ad utilizzare Scratch e, successivamente, abbiamo intervistato alcuni ragazzi che hanno vissuto sulla loro pelle il passaggio.

Tutti gli studenti da noi intervistati hanno risposto in modo unanime alle nostre domande formando un profilo dei problemi di Scratch abbastanza preciso:

* Impossibilità di operare al di fuori dell’ambiente di sviluppo
* Impossibilità di produrre file o eseguibili
* Difficile gestione e interpretazione dei codici lunghi
* Pessima gestione delle funzioni
* Limitazione sugli input
* Difficile gestione dei vettori
* Pesantezza del codice sui cicli

In più, alcuni dei ragazzi che sono già passati ai linguaggi di programmazione avanzati, si sono lamentati dell’inutilità della programmazione a blocchi di scratch che lo rende più un gioco che un vero mezzo per l’apprendimento.

### Scelte di programmazione

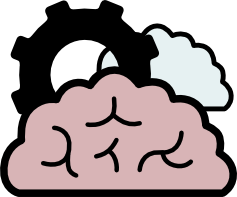
Per la realizzazione del progetto abbiamo dovuto, come prima cosa, andare a definire la sintassi stessa del linguaggio e prendere alcune che lo riguardano:

* Scelta .NET Framework: Abbiamo preferito trattare “Leggermente” come linguaggio tradotto, la sua traduzione e compilazione è affidata al linguaggio C#.  
  La scelta dell’ambiente è dovuta alla nostra conoscenza di esso ampliatasi durante il nostro periodo scolastico.
* Detipizzazione delle variabili: Come in numerosi linguaggi di Scripting quali Java script, PHP e altri specializzati nell’ambiente web, abbiamo preferito rimuovere i tipi.
* Localizzazione del linguaggio: La conoscenza della lingua inglese è essenziale in informatica, ma l’introduzione di un dizionario di parole chiave in italiano semplifica di molto la comprensione del linguaggio, permettendo di posticipare la standardizzazione a lingua inglese in una fase in cui i ragazzi abbiano piena consapevolezza del funzionamento del codice.
* Riduzione dei simbolismi: Per facilitare la comprensione del testo sono stati rimossi alcuni simboli, comunemente usati nei linguaggi di programmazione, a favore di parole chiave intuitive.
* Incapsulamento delle funzioni: Oltre a permette la scrittura delle funzioni, senza vincolarne l’ordine, abbiamo deciso di lasciare la possibilità, ai ragazzi, di creare dei veri e propri pacchetti di funzioni per permetterne l’esportazione e il riutilizzo in altri progetti.
* Continuità progettuale: Per evitare la dispersione del progetto, abbiamo contattato alcuni ragazzi frequentanti le classi quarte.  
  Alcuni di loro si sono dimostrati interessati a continuare il progetto in futuro.
* GitHub: Una scelta importante per il progetto è stata la piattaforma di condivisione, ci serviva infatti un mezzo rapido con cui condividere il codice e farne lo storico, in modo da lasciare diverso materiale ai ragazzi delle quarte.  
  La scelta è ricaduta quindi su GitHub, un servizio online di “repository”, che permette la memorizzazione e la condivisione di frammenti di codice.  
  Il progetto è fruibile all’indirizzo: https://GitHub.com/Fuzzuca/Leggermente.

A questo punto, decisa la sintassi, il nostro gruppo si è diviso in due, per la creazione dell’IDE di sviluppo e il sistema di traduzione.

Questa scelta è stata fatta dopo un rapido calcolo delle tempistiche, dove abbiamo visto, infatti, che la divisione del progetto in due parti differenti avrebbe reso possibile terminare, almeno in parte, il progetto in tempo per l’esame di stato.

## Realizzazione “sistema di traduzione”



La parte di progetto legata al sistema di traduzione è stata curata da Canale Davide.

Il sistema di traduzione è un insieme di classi che permetto l’analisi e la conversione di codice, dal “Leggermente” al C#.  
Le classi si comportano in parte come un compilatore per linguaggi standard e in parte come parser, poiché non tutto il codice viene tradotto, ma solamente convertito di forma per rispettare la sintassi del C#.

Il traduttore, per svolgere il suo compito, procede analizzando il codice in tre differenti fasi.

### Pre-Elaborazione

La Pre-Elaborazione ha l’obiettivo di modificare ed elaborare il codice in modo da unificarlo e renderlo facilmente fruibile alle fasi successive.  
Questo processo si estende in numerosi sottopassaggi che puntano a elaborare un qualsiasi codice generato da un utente.

Dopo aver unificato la tipologia di file come “file UNIX”, si procede alla creazione di costanti basate sul codice scritto dall’utente.

A questo punto, il sistema inizia una sequenza di pulizia e rimozione di frammenti di codice totalmente inutili dal punto di vista della traduzione, come commenti, spazi e righe vuote.

Infine, procede nell’analizzare il codice riga per riga, andando a rimuovere l’indentazione e compilando una classe che ne rappresenti il codice.  
Il tutto è supportato da un’apposita classe, in grado di memorizzare eventuali errori e messaggi.

### Elaborazione

Comprende la parte più corposa del programma, in essa, infatti, abbiamo diversi sistemi che si occupano di analizzare il codice ricorsivamente, andando a individuare le parti comuni e richiamando le funzioni in grado di risolverle.  
L’elaborazione, inoltre, ha il compito attivo di andare a controllare, e segnalare, eventuali errori nella sintassi del linguaggio.  
L’elaborazione inizia, infatti, con l’analisi dei pacchetti richiamati nel programma e con la loro apertura e integrazione da file “*\*.lmp*”, questo viene seguito da un breve controllo per verificare il giusto tipo di compilazione fra pacchetto e programma eseguibile.

Svolto ciò il traduttore inizia a richiamare una funzione per volta per analizzarne le righe, risolvendo ed elaborandone i contenuti, e convertendole in C#.

Durante questa elaborazione, viene controllata anche l’esistenza o meno degli elementi richiamati.

Una volta ottenuto un codice in C# che rispecchi appieno le funzionalità originalmente scritte, si procede alla compilazione.

### Compilazione

Detta anche “Post-Elaborazione”, richiama il compilatore del C# “csc.exe” internamente alla classe e procede con la compilazione.

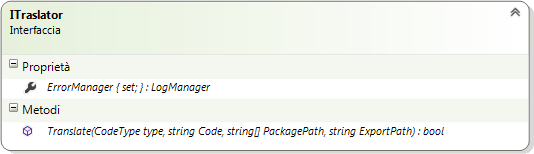
Fatto ciò ne analizza gli errori cercando di segnalarli all’utente, dove possibile, e se necessario avvertendo della necessità di procedere nuovamente con il processo di traduzione.

Infine impacchetta e deposita dove indicato i file generati.

### Problematiche

Durante la creazione del traduttore, sono sorte innumerevoli problematiche, ci siamo accorti, infatti, che non esistevano classi di supporto per il nostro progetto e abbiano dunque dovuto procedere alla loro creazione da zero.

In primo luogo, visto la divisione del progetto, ci siamo accordati per la creazione d’interfacce comuni, in modo da avere un oggetti comuni e fissi.

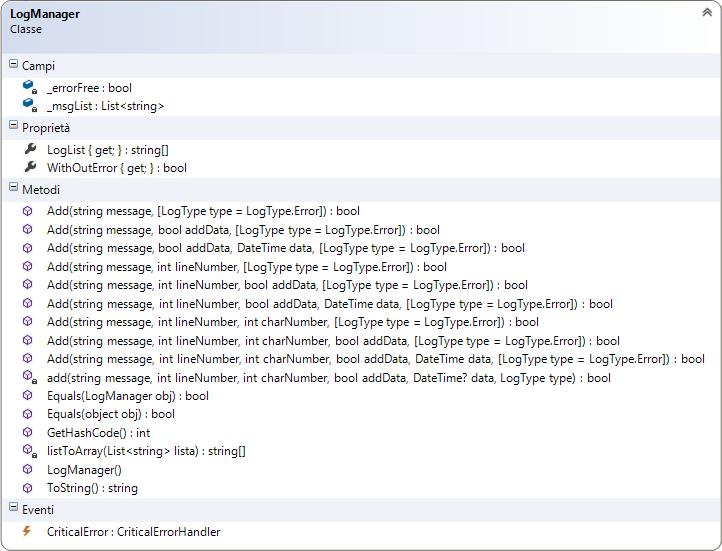


Interfacci del Traduttore

Oltre alle interfacce, un altro elemento in comune che ci siamo trovati obbligati a gestire, sono stati gli errori e i log.

Il traduttore, infatti, durante la sua elaborazione compila un report molto dettagliato di tutti gli errori e le possibili correzioni applicabili al codice.

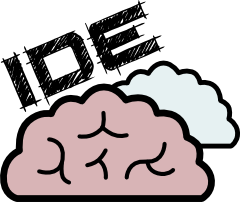
Sfortunatamente questo report non è immediatamente fruibile dall’utente, perciò siamo stati costretti a elaborare una classe intermedia che, trascinata lungo tutti i processi di conversione, immagazzina tutte le informazioni riguardanti i messaggi del traduttore.



UML classe gestione messaggi

Il traduttore non si è dimostrato complesso nella sua realizzazione; il problema più grande riscontrato sono state le tempistiche molto lunghe di lavorazione che ci hanno portato a lavorare sul progetto sino al giorno dell’esposizione.

## Realizzazione “IDE di sviluppo”



La parte di progetto legata all’ambiente di sviluppo è stata curata da Garabello Giacomo

L'ambiente di sviluppo, che nell’informatica viene comunemente denominato come IDE (Integrated Develop Environment), rispecchia nelle sue funzionalità i programmi usati all’interno dell’istituto. Tra queste possibilità si possono effettuare le principali azioni input, di output e la colorazione automatica del testo. Tutto ciò è stato sviluppato rispettando gli standard di Windows per facilitarne l'utilizzo a chi non lo avesse mai utilizzato questo programma.

Ecco i principali punti su cui si è basata la stesura del codice:

* Gestione dei File: salvare e caricare i dati dal disco fisso oppure la possibilità di stampare il documento appena redatto.
* Modifica del testo: copiare e incollare gli appunti o di annullare e ripristinare le azioni appena eseguite.
* Numerazione automatica: creare un’area dove poter formattare il codice in modo corretto con la numerazione delle linee di codice.
* Colorazione dinamica del testo: tramite un file di configurazione opportunamente redatto, la modifica della colorazione di alcune parole chiave del linguaggio.
* Collegamento al progetto del traduttore: Integrazione del sistema di traduzione e completamento della sua interfaccia utente.
* Funzione Auto-completamento: per semplificare ulteriormente l’utilizzo del linguaggio, l’IDE fornisce un aiuto in linea per il completamento delle parole.

Qui di seguito verranno trattate nel dettaglio alcune delle più gravi problematiche che si sono presentate durante lo sviluppo, ovvero:

* Stampa di RTF;
* Numerazione delle linee di codice;
* Colorazione dinamica.

### Stampa di RTF

La stampa di file formattati da una “*RichTextBox*”, componente principale di programmi per l’editing di testo, richiede molte più competenze di quanto si possa pensare.

Per proceder alla stampa abbiamo dovuto in primo luogo calcolare le dimensioni del codice da stampare e di conseguenza elaborarlo tramite la libreria di sistema “*System.Drawing*”.

Il codice risultante è questo:

|  |
| --- |
| StringReader reader = new StringReader(RTBText.Text);  float LinesPerPage = 0;  float YPosition = 0;  int Count = 0;  float LeftMargin = e.MarginBounds.Left;  float TopMargin = e.MarginBounds.Top;  string Line = null;  Font PrintFont = this.RTBText.Font;  SolidBrush PrintBrush = new SolidBrush(Color.Black);  LinesPerPage = e.MarginBounds.Height / PrintFont.GetHeight(e.Graphics);  while (Count < LinesPerPage && ((Line = reader.ReadLine()) != null))  {  YPosition = TopMargin + (Count \* PrintFont.GetHeight(e.Graphics));  e.Graphics.DrawString(Line, PrintFont, PrintBrush, LeftMargin,  YPosition, new StringFormat());  Count++;  }  if (Line != null)  e.HasMorePages = true  else  e.HasMorePages = false;  PrintBrush.Dispose(); |

Si potrebbe notare la presenza della classe *PrintDocument* e pensare che essa basti alla stampa, sfortunatamente è una classe generalizzata a tutti gli usi di stampa e il metodo *Print()* non permette una diretta formattazione.

### Numerazione delle linee di codice

Per eseguire la numerazione delle linee di codice è stato necessario implementare una nuova versione del controllo denominato “*RichTextBox*”.

Le maggiori problematiche riscontrate sono state il ridimensionamento del Font e l’eccessivo tempo di elaborazione del processo per la numerazione.

Per ridurre le tempistiche abbiamo preferito utilizzare come base una classe trovata in rete, la classe è disponibile all’indirizzo: http://GitHub.com/zippy1981/XPathVisualizer.

Sfortunatamente la classe presentava numerose incorrettezze che ne rallentavano l’esecuzione, per questo motivo è stato necessario riscrivere gran parte del codice.

Inoltre ho aggiunto la gestione dello zoom e ho predisposto la classe per la colorazione della parole chiave riguardanti il linguaggio di programmazione.

Di seguito allego un piccolo frammento di codice presente nella classe per la colorazione e scrittura delle zone in cui viene effettuata la numerazione.

|  |
| --- |
| private void PaintLineNumbers()  {  int w = (LineNumberWidth) + 3;  if (w != \_lastWidth)  {  SetLeftMargin(w + Margin.Left);  \_lastWidth = w;  return;  }  Bitmap buffer = new Bitmap(w, this.Bounds.Height);  Graphics g = Graphics.FromImage(buffer);  var rect = new Rectangle(0, 0, w, this.Bounds.Height);  g.FillRectangle(new SolidBrush(Color.LightGray), rect);  int n = NumberOfVisibleTextLines  int first = FirstVisibleTextLine  int py = 0;  int w2 = w - 2 - (int)NumberBorderThickness;  Pen dividerPen = new Pen(NumberColor);  for (int i = 0; i <= n; i++)  {  int ix = first + i;  int c = GetCharIndexForTextLine(ix)  var p = GetPosFromCharIndex(c + 1);  Rectangle r4 = Rectangle.Empty;  if (i == n)  {  if (this.Bounds.Height <= py) continue;  r4 = new Rectangle(1, py, w2, Font.Height);  }  else  {  if (p.Y <= py) continue;  r4 = new Rectangle(1, py, w2, Font.Height);  }  g.FillRectangle(new SolidBrush(Color.LightGray), r4);  if (NumberLineCounting == LineCounting.CRLF) ix++;  if (NumberAlignment == StringAlignment.Near)  rect.Offset(0, 3);  var s = ix.ToString();  g.DrawString(s, NumberFont,  new SolidBrush(Color.DarkGray), r4, \_stringDrawingFormat);  py = p.Y;  } |

### Colorazione dinamica

Il problema più grande, riscontrato nella creazione della “*Form”*, è stata la colorazione delle parole chiave del linguaggio.

Per eseguire questa operazione è stato necessario creare un nuovo formato di file per la configurazione.

È stato quindi introdotto un nuovo tipo di file con estensione “*\*.lmc*” con cui è stato possibile configurare il programma per la corretta colorazione delle parole chiave tramite riscontro diretto o Regular Expression, anche dette Regex.

Qui di seguito è presente una porzione del file “*ColorsConfig.lmc*”:

|  |
| --- |
| #Controlli Condizionali    se=255;204;51  altrimenti=255;204;51  controlla=255;204;51  ripeti=255;204;51  volte=255;204;51  quando=255;204;51  #regex commenti  ^\|.+?\|=173;255;47 |

La configurazione della colorazione è ispirata ai “file INI”, comunemente usati nell’ambiente Windows.  
È possibile commentare alcune righe del file tramite l’utilizzo del simbolo “#” e indicare le Regex tramite “^”.

I colori vanno scritti con notazione RGB, ma con i valori espressi in decimale

Di seguito riporto il metodo per la colorazione.

|  |
| --- |
| public static void ColoraTesto(RichTextBoxEx rtb,List<ColorConfig>color)  { Color col = rtb.ForeColor;  string rex = @"[\w]+[.\s]($)?";  MatchCollection Collections = Regex.Matches(rtb.Text, rex);  foreach (ColorConfig i in color)  {  if (i.IsRegex)  {  MatchCollection rexres = Regex.Matches(rtb.Text, i.Value);  foreach (Match n in rexres)  {  rtb.Select(n.Index, n.Length);  rtb.SelectionColor = i.color;  }  else  {  foreach (Match m in Collections)  {  if (i.Value.ToLower() == m.ToString().Trim().ToLower())  {  rtb.Select(m.Index, m.Length);  rtb.SelectionColor = i.color;   }}}  }} |

# Manuale di programmazione

## Sintassi del linguaggio

Il linguaggio “Leggermente” s’ispira, per la sintassi, a differenti linguaggi di programmazione che ultimamente si stanno diffondendo nel mondo della programmazione professionale e non, come il Lua e il Python; il linguaggio prende liberamente ispirazione da diversi linguaggi e le sue strutture sintattiche sono ispirate a essi.

La sintassi sfrutta la tabulazione per suddividere i comandi in regioni facilmente riconoscibili.

Inoltre abbiamo deciso di eliminare la possibilità di concatenare più comandi, per permettere una più facile comprensione.

Le funzioni sono indicate con il simbolo “**#**” seguite dal nome, il codice delle funzioni deve essere scritto tra parentesi quadre per permetterne l’identificazione.

Qualunque funzione restituisce un valore non tipizzato e i parametri sono inseriti senza specificazione di tipo.

I parametri dovranno essere indicati tra parentesi tonde con il loro nome e nell’utilizzo della funzione andranno inseriti nell’esatta sequenza.

La funzione principale, comunemente denominata “Main”, qui sarà nominata come:

|  |
| --- |
| **#programma**[  …  ] |

### I file e le loro estensioni

In “Leggermente” esistono due tipi di file:

* “\*.lm”: Sono i sorgenti del nostro progetto, essi sono identificabili per la presenza del tag “**#programma**”, produrranno i file eseguibili.
* “\*.plm”: Sono delle raccolte di funzioni a cui viene assegnato un nome, essi si possono riconoscere per il tag “**pacchetto**” al loro interno e per la mancanza del tag “**#programma**”.
* “\*.lmc”: Sono i file di configurazione del progetto.

### Commenti

“Leggermente”, come tutti i linguaggi di programmazione, implementa la possibilità di scrivere dei commenti direttamente nel codice.  
Viene considerato commento tutto ciò che è presente fra “*|*”

|  |
| --- |
| **#Funzione**[  **se** 3 **maggiore** 1  **ritorna** vero *|questo è un commento|*  **altrimenti**  **ritorna** falso *|anche questo|*  ] |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * I commenti sono essenziali per un codice, poiché permettono a distanza di anni una sua semplice comprensione. * La scelta del carattere è stata pensata in relazione alla configurazione standard sulla tastiera italiana e americana, infatti il pipe è per entrambe un carattere velocemente accessibile. |

### Funzioni

Di seguito presentiamo un esempio di funzione:

|  |
| --- |
| **#NumeroPari**(**numero**)[  **se** **MATE**.**Resto** **di** numero,2 **uguale** 0  **ritorna** vero  **altrimenti**  **ritorna** falso  ] |

Come si può osservare, la funzione viene dichiarata con il simbolo “**#**”, i parametri (all’interno delle parentesi tonde) sono separati da una virgola.

Infine si può specificare il valore di ritorno della funzione.

Le Procedure presentano una sintassi perfettamente uguale alle funzioni, senza però ritornare alcun valore.

Esempio:

|  |
| --- |
| **#Esempio**(**numero**)[  **se** **MATE**.**Resto** **di** numero,2 **uguale** 0  … *|codice|*  **altrimenti**  … *|codice|*  ] |

**N.B.** Se in una funzione o in un ramo di una struttura di controllo non viene specificato un valore di ritorno, viene restituito NULL.

I parametri delle funzioni e procedure sono sempre passati per referenza e non per valore.

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * Nel linguaggio, non essendoci variabili tipizzate, non è più necessario specificare le tipologie in uscita. * Le parole chiave (***di****,* ***con****,* ***a****,* ***per***) sono intercambiabili fra loro e la scelta ricade solo per semplificare la lettura del codice |

### Pacchetti

Il pacchetto è un insieme di funzioni e/o procedure che viene definito su un file esterno e che può essere incluso in qualunque altro progetto.

Nella dichiarazione del pacchetto è necessario dichiarate il tag “**pacchetto**” e al suo interno non deve essere presente il tag “**#programma**”.

La dichiarazione avviene come nell’esempio:

|  |
| --- |
| **pacchetto** MATE  **#Pari**(**numero**)[  …  ]  **#Media**(**vettore**)[  …  ] |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * I pacchetti permettono una rapida esportazione e riuso delle funzioni precedentemente scritte. |

### Variabili e Vettori

Nel linguaggio le variabili non sono state tipizzate, esse possono contenere un qualsiasi valore.

Sarà possibile sapere il tipo di dato contenuto mediante i metodi della libreria “**VAR**” come riportato nelle referenze in fondo al manuale.

I comandi standard nella gestione delle variabili sono:

|  |
| --- |
| **crea** pippo → Crea una variabile  **crea** pluto **a** 0 → Crea una variabile con un valore indicato  **cambia** pippo **a** 1 → Cambia la variabile al valore indicato  **scambia** pluto **con** pippo → Scambia due variabili fra loro |

Anche se non utilizzate all’atto pratico, il linguaggio ragiona su quattro tipi di variabili:

* Double: classico numero con la virgola
* String: stringa di testo
* Bool: variabile vera o falsa
* Vector: insieme di valori vettoriali.

Il vettore non è infatti altro che una lista di variabili accessibili tramite indice singolo.

Ogni variabile, a sua volta, può diventare un vettore, di conseguenza è possibile creare una matrice con gli appositi comandi:

|  |
| --- |
| **crea** **vettore** vet **di** 5 → Crea un vettore di 5  **cambia** vet(1) **a “gatto”** → Cambia il primo elemento del vettore a “gatto”  **crea** **vettore** vet(2) **di** 3 → Crea un vettore di 3 nel terzo elemento  **cambia** vet(2)(1) **a** 5 → Cambia il primo elemento del secondo a 5 |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * Le variabili sono state detipizzate per permettere una più libero utilizzo, in relazione anche a quello riscontrato dalle interviste. Infatti, gli studenti intervistati, hanno confermato la semplicità delle variabili di Scratch. |

### Comparatori logici e Simboli matematici

Oltre ai classici simboli quali:

|  |
| --- |
| + → Somma e concatenazione stringhe  - → Sottrazione  / → Divisione  \* → Moltiplicazione  “” → Definizione stringa |

Abbiamo quelli in parole chiave, quali:

|  |
| --- |
| **maggiore** → Comparatore di maggioranza  **minore** → Comparatore di minoranza  **uguale** → Comparatore di eguaglianza  **diverso** → Comparatore di diseguaglianza  **non** → Negazione di un Booleano |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * La riduzione dei simbolismi porta a una più semplice e immediata comprensione, a svantaggio di una più veloce scrittura del codice. |

## Strutture di controllo

Come in tutti i linguaggi, anche il “Leggermente” sfrutta delle strutture di controllo per condizionare il codice.

### Strutture condizionali

Nel “Leggermente” è possibile concatenare, nelle strutture condizionali, una sequenza di condizioni tramite l’utilizzo delle parentesi tonde e la concatenazione booleana fornita dai blocchi “**e**” ed “**o**” per ognuno dei seguenti casi:

* **se**: condizionale semplice, permette di eseguire un determinato blocco di azioni se la condizione primaria indicata viene rispettata:

|  |
| --- |
| **se** (numero **maggiore** 0) **e** (x **diverso** -1)  **cambia** x **a** numero |

* **se**-**altrimenti**: condizionale composto, permette l’esecuzione di un blocco “se” con un annesso blocco nel caso la condizione primaria risulti falsa:

|  |
| --- |
| **se** numero **maggiore** 0  **cambia** x a 0  **altrimenti**  **cambia** x a 1 |

* **controlla**: condizionale multiplo, in questo elemento è possibile confrontare una variabile con una serie di condizioni ed eseguire diversi blocchi di codice in relazione alla veridicità di essi:

|  |
| --- |
| **controlla** numero  **se** **minore** 0  **cambia** x a 1  **se** **uguale** 0  **cambia** x a 2  **se** **maggiore** 0  **cambia** x a 3 |

Come visto negli esempi precedenti, ognuna delle strutture contiene le proprie sequenze di comandi tramite la tabulazione.

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * La tabulazione viene usata normalmente per permettere una rapida lettura del codice, per questo motivo abbiamo deciso di usare la tabulazione stessa per delimitare le sequenze dei comandi. * “**controlla”** è un simulacro della funzione “switch” presente nei linguaggi di programmazione, però, sotto consiglio degli intervistati, abbiamo preferito lasciare la possibilità di definire il tipo di controllo da usare sulla variabile. |

### Strutture d’iterazione

Le strutture d’iterazione sono strutture nel linguaggio che permettono di ripetere un determinato blocco di comando in una specifica situazione.

Queste strutture sono:

* **ripeti volte**: struttura che permette di ripetere un blocco di codice un determinato numero di volte senza una condizioni che ne determini l’interruzione o il procedimento:

|  |
| --- |
| **ripeti** 5 **volte**  **schermo** “LeggerMente\n” |

* **quando ripeti**: strutture che permettono la ripetizione di un blocco di codice sotto una determinata condizione:

|  |
| --- |
| **ripeti quando** x **maggiore** 0  **schermo** “Minore” |

* **ripeti quando**: struttura che permette l’esecuzione di un blocco di codice e solo dopo il controllo di una condizione che ne determini la continuazione

|  |
| --- |
| **ripeti**  **schermo** “controllo minimo”  **quando** x **minore** 0 |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * Le strutture d’iterazione sono studiate e riproposte da Scratch, ma per essere conformi ai linguaggi di programmazione standard, tutte ciclano per una condizione vera. |

### Indice nelle strutture ad iterazione

Nel “Leggermente” ogni struttura d’iterazione durante l’esecuzione del suo blocco d’istruzioni, genera una variabile normalmente denominata “***Indice***” contenente il numero di esecuzioni che si sono svolte su quel blocco.  
E possibile ridefinire il nome di questa variabile mediante il tag “**con**”:

|  |
| --- |
| **ripeti** x **volte con** i  **cambia** vet(i) **a** vet(i)+5 |

|  |  |
| --- | --- |
|  | **Vantaggi** |
| * La ridefinizione dell’indice permette, oltre che l’utilizzo in cascata, una maggiore possibilità di personalizzazione del codice. |

## Esempi

Qui riportiamo alcuni esempi di codice scritto in “Leggermente”:

### Bubble Sort

|  |
| --- |
| **aggiungi** MATE    **#programma[**  **crea** n **a** 5  **crea vettore** v **a** n  **crea** sup  **crea** check    **ripeti** 5 **volte**  **cambia** v**(**indice**) a MATE.random di** 0**,**100    **ripeti**  **cambia** check **a falso**  **ripeti se** indice **minore di** n**-**1  **se** v**(**indice**) minore** v**(**indice**+**1**)**  **scambia** v(indice) **a** v(indice+1)  **cambia** check **a vero**  **cambia** n **a** n**-**1  **quando** check    **ripeti** 5 **volte con** i  **schermo** v**(**i**)+”\n”**  **]** |

## Referenze Pacchetti

Qui di seguito aggiungiamo le referenze di alcuni fra i pacchetti base concepiti per il linguaggio “Leggermente”.

|  |
| --- |
| IO |
| schermo → Ripulisce lo schermo  schermo di var → Esegue la stampa a schermo della variabile  tastiera → Legge gli input sino alla pressione dell’invio  tasto → Procede alla lettura di un singolo tasto  tasto di → Procede alla lettura di “var” tasti |

|  |
| --- |
| VAR |
| tipo → Restituisce il tipo della variabile  lunghezzaVect → Restituisce la lunghezza del vettore  lunghezzaStringa → Restituisce la lunghezza della stringa  substringa di var → Esegue una sotto stringa di una stringa per “var” caratteri  substringa di var1, var2 → Esegue una sotto stringa da “var1” per “var2” caratteri |

|  |
| --- |
| MATE |
| resto di var1, var2 → Riporta il resto di “var1” diviso per “var2”  random di var1, var2 → Restituisce un numero random da “var1” a “var2”  sqrt di var1, var2 → Esegue la radice di “var1” alla “var2”  exp di var1, var2 → Esegue l’esponenziale di “var1” alla “var2” |

# Risultati progetto

## Raccolta risultati

Vista la complessità del progetto e la possibilità di incrementare il lavoro negli anni successi, grazie alla collaborazione con le classi quarte, abbiamo deciso innanzi tutto di capire quanto il progetto sia effettivamente riuscito.

Per far ciò abbiamo portato a termine un’esercitazione, prendendo in esame dieci soggetti, di cui cinque che abbiano seguito, almeno in parte, il corso di studi informatici del “G.Vallauri” e altrettanti soggetti che invece abbiano seguito un altro tipo di corso di studi.

Nell’esercitazione, abbiamo richiesto ai ragazzi di creare un programma che generi una sequenza di Fibonacci lunga in proporzione al numero in input.

I risultati sono qui allegati, sono stati raccolti dai ragazzi:

* Canale Francesca (2c Mec – I.I.S. “G.Vallauri”)
* Enzo Saccà (3B Mec – I.I.S. “G.Vallauri”)
* Hachenberg Rainer (3B Eltr – I.I.S. “G.Vallauri”)
* Edoardo Farello (3G – Istituto Alberghiero “V.Mucci”)
* Noemi Revello (5D  – I.P.C “P.Cillario Ferrero”)
* Serafino Stefano (4C Info – I.I.S. “G.Vallauri”)
* Andrea Gilardi (1D Info – I.I.S. “G.Vallauri”)
* Russo Mattia (1B Info – I.I.S. “G.Vallauri”)
* Torrengo Federico (3E Info – I.I.S. “G.Vallauri”)
* Panetta Andrea (2A Info – I.I.S. “G.Vallauri”)

## Analisi risultati

Dopo la raccolta dei risultati è evidente il successo del linguaggio.

Si sono rilevati dei problemi a causa della scarsa conoscenza del linguaggio da parte dei soggetti che, qualche volta, hanno commesso errori.  
Ciò nonostante, i risultati sono incoraggianti poiché tutti i soggetti sono riusciti a completare gli esercizi con un buon grado di correttezza e senza commettere gravi errori riguardanti la sintassi del linguaggio.

Inoltre, i soggetti al di fuori del corso di studi informatici, si sono dimostrati capaci di assimilare le basi del linguaggio con il minimo intervento da parte nostra.

## Ringraziamenti

Per l’aiuto e collaborazione al progetto si ringraziano:

* Giuseppe Borra
* Oscar Cambieri
* Giovanni Tosello
* Tancredi Canonico

Ringraziamo, inoltre, tutti i ragazzi che hanno aiutato nelle ricerche del linguaggio di programmazione e tutti i professori che hanno concesso le ore di lezione per lo sviluppo dei software.