



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Progetto e sviluppo del linguaggio di programmazione DLK

Relatore

Prof. Gian Franco Lamperti

Laureando

Stefano Frati

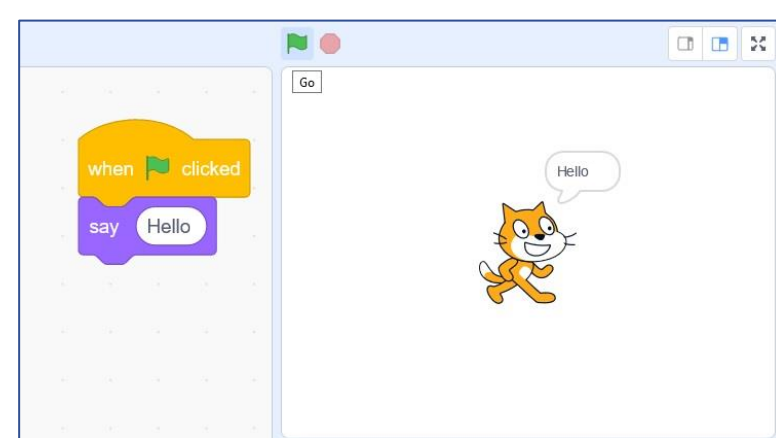
➔ **DLK** (***Didactical Language for Kids***)

- Linguaggio di programmazione a scopo didattico
- Rivolto ad un pubblico giovane



- Keyword in italiano
- Costrutti semplificati rispetto ai principali linguaggi di programmazione

- ➔ Si pone ad un livello intermedio fra i linguaggi di programmazione per bambini che utilizzano il paradigma di programmazione visuale e i linguaggi di programmazione standard come *C*, *Pascal*, ecc...
- ➔ L'obiettivo è quello di introdurre i giovani alla programmazione di alto livello, senza perdere la componente di comprensibilità del codice tipica dei linguaggi per bambini



Scratch



```
inizio  
    scrivi ("Hello");  
fine.
```

DLK



```
#include <stdio.h>  
  
int main(void)  
{  
    printf("Hello");  
    return 0;  
}
```

C

Il progetto

- ➔ Il progetto di creazione del DLK si può suddividere in due fasi:
- **Specifica** del linguaggio di programmazione (regole: lessicali, sintattiche e semantiche)
 - **Implementazione** del linguaggio di programmazione
- ➔ **Interprete DLK** ➔ Permette di eseguire programmi scritti in DLK su di un elaboratore

Specifica: caratteristiche principali del DLK

- ➔ Adotta il paradigma di programmazione **imperativo**
- ➔ Le istruzioni DLK terminano sempre con « ; »
- ➔ È possibile introdurre commenti nel codice

commento \rightarrow `//(~[\n])* \n`

```
// questo è un commento, verrà ignorato dall'interprete
```

Sezioni di un programma DLK

➔ Un programma DLK è suddiviso in **due** sezioni

sezione dedicata alla dichiarazione di variabili

inizio

istruzioni

fine.

corpo del programma

Sezione di dichiarazione delle variabili

➔ Lista, anche vuota, di dichiarazioni di variabili

decl-list → {*decl* ;}

decl → *type* : *id-list*

type → **intero** | **decimale** | **stringa** | **boolean**

id-list → **id** {, **id**}



tipo: *lista di nomi di variabili;*



intero: a, b, c;
boolean: d;

Corpo del programma

- ➔ Lista di istruzioni racchiuse fra le due keyword «**inizio**» e «**fine.**»

body → **inizio** *stat-list* **fine.**

stat-list → {*stat* ;}

stat → *assign-stat* | *se-stat* | *ripeti-stat* | **stop** | *scrivi-stat* | *inserisci-stat* |
inc-dec-stat

inizio

istruzioni

fine.

Istruzione di assegnamento

assign-stat \rightarrow **id** = *rhs-assign-stat*

rhs-assign-stat \rightarrow *math-expr* | *bool-const* | **string-const** | **id**



```
pi = 3.14;  
str = "ciao";  
bool = vero;  
res = 3*(2-a);
```

- ➡ Il **tipo** di valore assegnato alla variabile deve essere coerente con quello dichiarato in precedenza (coercizione per tipi numerici)
- ➡ Il **nome** della variabile a cui viene assegnato un valore deve esser stato dichiarato in precedenza

Incremento e decremento di una variabile

- ➔ È possibile **incrementare** e **decrementare** una variabile di un'unità attraverso gli operatori «++» e «--»
- ➔ La variabile deve esser stata dichiarata «**intero**» o «**decimale**» e inizializzata in precedenza

```
a = a + 1;  
a = a - 1;
```

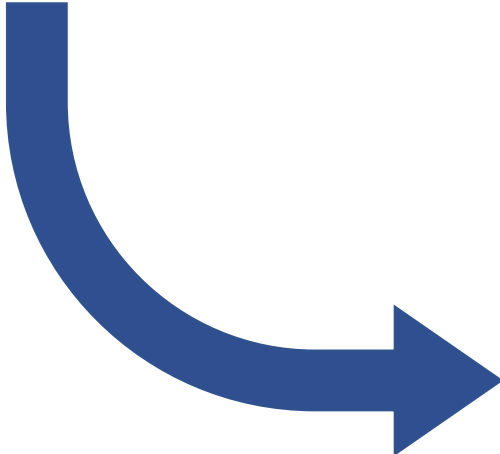


```
a++;  
a--;
```

L'istruzione se-altrimenti

- ➔ Istruzione **condizionale**
- ➔ Due sezioni, una obbligatoria e l'altra facoltativa

se-stat → **se** (*logical-expr*) **vero fai** : *stat-list* [*altrimenti-stat*] **fine**
altrimenti-stat → **altrimenti** : *stat-list*



```
se ( espressione logica ) vero fai :  
istruzioni  
altrimenti :  
istruzioni } Facoltativo  
fine ;
```

L'istruzione se-altrimenti

```
se (a > b) vero fai:  
    a--;  
fine;
```

```
se (a > b) vero fai:  
    a--;  
altrimenti:  
    a++;  
fine;
```

- ➔ Gli operatori logici sono «**e**» ed «**o**» (**and** e **or** logici)
- ➔ Differenza nel confronto fra stringhe rispetto ad altri linguaggi:
 - Con gli operatori relazionali si confronta la **lunghezza** delle stringhe
 - Con gli operatori di uguaglianza si confronta il **contenuto**

"Alice" > "Bob"



vero

"Paolo" == "Irene"



falso

L'istruzione ripeti

ripeti-stat → **ripeti** *math-expr* **volte** : *stat-list* **fine**



```
ripeti espressione matematica volte:  
istruzioni  
fine;
```

```
ripeti 10 volte:  
    scrivi("Ciao");  
fine;
```

```
ripeti n volte:  
    scrivi("Ciao");  
fine;
```

```
ripeti n+4 volte:  
    scrivi("Ciao");  
fine;
```

- ➡ Numero di **iterazioni** dato dal risultato dell'espressione matematica
- ➡ L'espressione matematica deve risultare positiva
- ➡ Nel caso in cui l'espressione matematica risulti un numero decimale, le cifre dopo la virgola verranno troncate

L'istruzione stop

- ➔ Permette di abbandonare un ciclo **ripeti** prima della sua naturale terminazione
- ➔ Può essere utilizzata solamente all'interno di un ciclo **ripeti**

```
a = 2;  
ripeti 10 volte:  
    a++;  
    se (a == 3) vero fai:  
        stop;  
    fine;  
fine;
```

L'istruzione **scrivi**

➔ Permette di **stampare** a terminale un valore

scrivi-stat → **scrivi** (*scrivi-arg*)
scrivi-arg → **strconst** | *expr* | **id**
expr → *math-expr* | *logical-expr*



scrivi (*espressione*) ;

`scrivi ((2+2) / 4) ;`



1

`scrivi (3>2) ;`



vero

`scrivi ("Ciao\nTutto bene?\n") ;`



Ciao
Tutto bene?

L'istruzione inserisci

inserisci-stat → **inserisci (id)**



inserisci (*variabile*) ;

- ➔ Permette di **inserire** da tastiera il valore da assegnare ad una variabile
- ➔ Il valore inserito da tastiera deve esser coerente col tipo dichiarato della variabile a cui si vuole assegnare il valore

`inserisci(a) ;`

Implementazione: l'interprete DLK

- ➔ Scritto in Python, in modo da garantire la massima **portabilità**
- ➔ Durante lo sviluppo dell'interprete si è prestata molta attenzione alla sua **facilità** di utilizzo, creando un'interfaccia guidata in italiano

```
#####  
#####  
#####          BENVENUTO NELL'INTERPRETE DLK          #####  
#####  
#####  
  
Inserire il nome del file contenente il programma da interpretare (ESC per uscire) --> |
```

Schermata d'avvio dell'interprete

L'interprete DLK: gli errori

- ➔ **Riconosce e segnala errori:** lessicali, sintattici e semantici
- ➔ Si è posta grande attenzione nel cercare di rendere la segnalazione degli errori *user-friendly*

```
Inserire il nome del file contenente il programma da interpretare (ESC per uscire) --> prova.dlk  
Impossibile trovare il file: 'prova.dlk'  
Controllare che il file sia nella stessa cartella dell'interprete
```

```
ERRORE DI SINTASSI:  
Riga 5, colonna 1 --> ';' mancante  
  
ERRORE DURANTE L'ESECUZIONE DEL PROGRAMMA:  
Alla riga 7 --> L'argomento della 'radice' deve essere positivo  
  
ERRORE DI SINTASSI:  
Riga 7, colonna 11 --> Chiudere la parentesi ')'
```

L'interprete DLK: esempio

- ➔ Esempio di utilizzo dell'interprete
- ➔ Verrà eseguito un programma scritto in DLK chiamato: «esempio.dlk»
- ➔ Il programma «esempio.dlk» permette di calcolare l'area di alcune figure geometriche

L'interprete DLK: esempio

```
stefano@DESKTOP-G7T0JLF:/mnt/c/Users/Stefano/desktop$ python3 DLK.py ← Avvio interprete

#####
#####
#####          BENVENUTO NELL'INTERPRETE DLK          #####
#####
#####

Inserire il nome del file contenente il programma da interpretare (ESC per uscire) --> |
```



```
Inserire il nome del file contenente il programma da interpretare (ESC per uscire) --> esempio.dlk

Inserire:
1 per calcolare l'area di un quadrato
2 per calcolare l'area di un rettangolo
3 per calcolare l'area di un triangolo
4 per calcolare l'area di un cerchio
0 per uscire
--> 2

Inserire la lunghezza della base --> 5
Inserire la lunghezza dell'altezza --> 10

L'area è: 50.0
```

↑
Inserimento del file
da interpretare

L'interprete DLK: esempio

```
Inserire:
1 per calcolare l'area di un quadrato
2 per calcolare l'area di un rettangolo
3 per calcolare l'area di un triangolo
4 per calcolare l'area di un cerchio
0 per uscire
--> 0

Arrivederci, alla prossima!

Programma 'esempio.dlk' terminato, premi invio per continuare|
```

Avviso di
terminazione del
programma



```
Programma 'esempio.dlk' terminato, premi invio per continuare

Inserire il nome del file contenente il programma da interpretare (ESC per uscire) --> ESC

Arrivederci, a presto!

stefano@DESKTOP-G7T0JLF:/mnt/c/Users/Stefano/desktop$ |
```

Terminazione
dell'esecuzione
dell'interprete

- ➔ Estendere il linguaggio con nuovi **costrutti** e **strutture dati** (ciclo *while*, potenze, *array*, ...)
- ➔ Creare un **IDE** apposito per il DLK
- ➔ Creare un **sito web** dove trovare tutte le informazioni sul linguaggio e dei video tutorial
- ➔ Consultare un **pedagogista** per individuare le migliorie da apportare al DLK e all'interprete in futuro

Grazie per l'attenzione!