**Progetto del corso Linguaggi di Programmazione a.a. 22/23**

**Anthony Di Pietro – 1960447**

**Mini-C: un linguaggio con puntatori**

Tavola dei contenuti

[Introduzione 2](#_Toc127098843)

[Inizializzazione 3](#_Toc127098844)

[Valutazione 5](#_Toc127098845)

[Utilizzo dei puntatori 6](#_Toc127098846)

[Free e garbage collector 8](#_Toc127098847)

[Funzioni che lavorano sui tipi del linguaggio 9](#_Toc127098848)

[Conclusioni 10](#_Toc127098849)

# Introduzione

Il progetto simula un mini-linguaggio che fa uso dei puntatori adottando l’approccio dei bucket per la gestione della memoria: una funzione hash identifica univocamente un certo nome a un certo bucket e, nella lista dei bucket, gli elementi sono inseriti in coda man mano che vengono inizializzati. Definiamo tre diversi ambienti:

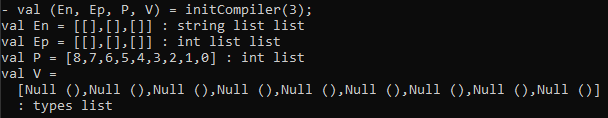
* **L’ambiente dei nomi *En*:** una lista di liste lunga *n*, in cui *n* è un valore definito inizialmente dalla costante *const\_ncells.* Le liste più interne, chiamate bucket, sono di lunghezza variabile in funzione del numero di elementi inizializzati in un dato momento che sono stati assegnati dalla funzione hash a quel bucket. *En* conterrà i nomi delle variabili inizializzate dall’utente alle quali sarà sempre associato un puntatore.
* **L’ambiente dei puntatori *Ep*:** una lista di liste speculare a *En* che associa un puntatore a un elemento di *En* univocamente. In Ep[i,j] è presente il puntatore che riferisce alla variabile En[i,j]. I puntatori sono rappresentati come interi e, un certo puntatore *p*, punta alla zona di memoria *V[p]* dell’ambiente dei valori *V*.
* **L’ambiente dei valori** ***V*:** array monodimensionale di lunghezza *n²* nel quale saranno salvati i valori associati a delle variabili.

Durante l’esecuzione del progetto, per un uso più efficiente in termini computazionali del linguaggio, sarà anche sempre definito *P* che rappresenta la lista di puntatori liberi in un certo istante.

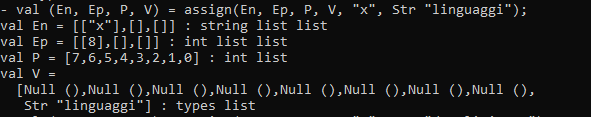
# Inizializzazione

In questa sezione mostriamo l’inizializzazione del compilatore e di alcune variabili.

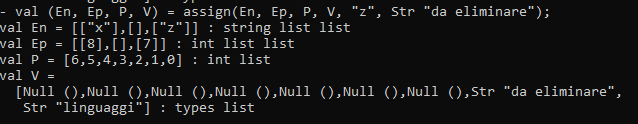
Come prima cosa inizializziamo il compilatore attraverso la funzione *initCompiler*. Essa ci restituirà l’ambiente dei nomi *En*, l’ambiente dei puntatori *Ep*, la lista dei puntatori liberi *P* e l’ambiente dei valori *V*.



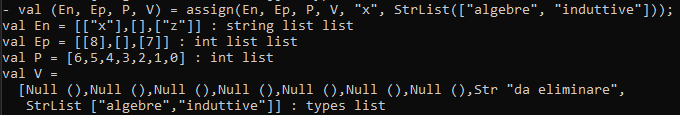
Inizializziamo una variabile *x* attraverso la funzione *assign* e le assegniamo il tipo Str, un tipo definito all’interno del progetto attraverso il costrutto datatype, e come valore la stringa “linguaggi”.



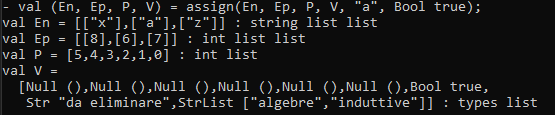
Facciamo la stessa cosa con una variabile *z* e le assegniamo la stringa “da eliminare”.



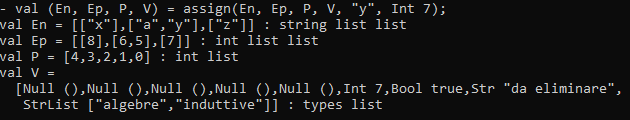
Ora assegniamo alla variabile già definita *x* una lista di stringhe StrList. È da osservare che, poiché *x* è già definita, non viene creata una nuova entry nell’ambiente dei nomi o dei puntatori, ma viene aggiornato solamente il contenuto nell’ambiente dei valori sostituendo il vecchio valore associato a *x*.



Ora inizializziamo una nuova variabile *a* con un valore di tipo bool.



E, infine, inizializziamo la variabile *y* con un intero 7.



**Conclusione:** è da osservare come la funzione *hash* cerca di distribuire le diverse variabili all’interno di ogni bucket.

# Valutazione

In questa sezione mostreremo, data una variabile già inizializzata, il valore associato a quella variabile.

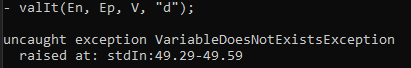
La funzione *valIt*, dati i tre ambienti e un certo nome di variabile, ci restituisce il valore associato a quella variabile. Proviamo a vedere quale sia il valore associato a *x*.



Facciamo un altro esempio: stavolta vogliamo sapere quale valore è associato a *y*.



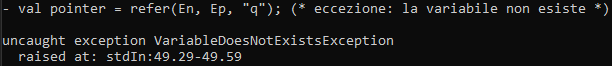
Se a *valIt* passiamo il nome di una variabile non inizializzata, riceveremo un’eccezione.



# Utilizzo dei puntatori

In questa sezione mostreremo alcuni utilizzi dei puntatori e le funzioni che ci permettono di lavorare con essi.

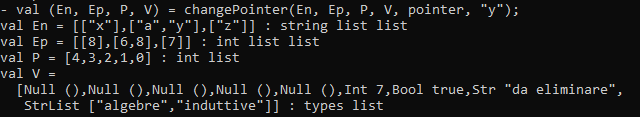
Proviamo subito con un non-esempio: chiediamo al compilatore quale sia il puntatore associato a una variabile non inizializzata; esso risponderà con un’eccezione.



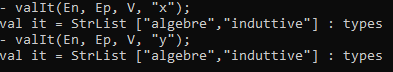
Ora proviamo a chiedere quale sia il puntatore della variabile *x*, correttamente inizializzata, e lo salviamo dentro una variabile di nome *pointer*.



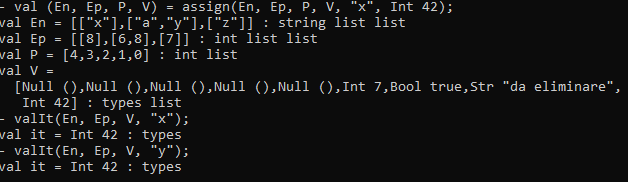
Adesso vogliamo assegnare *pointer*, variabile che contiene il puntatore di *x*, a un’altra variabile *y*.



Utilizzando *valIt* possiamo osservare come alle due variabili sia associato lo stesso valore.



Ora proviamo a cambiare il valore assegnato a *x* con l’intero *42* e osserviamo come viene mutato anche il contenuto di *y*.

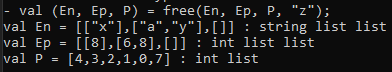


Supponiamo di voler sapere a che cella di memoria punta un certo puntatore *p*. Utilizziamo la funzione *defer*.



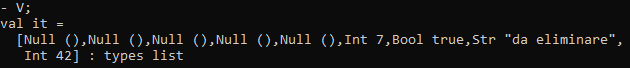
# Free e garbage collector

Ora supponiamo di non necessitare più della variabile dapprima inizializzata *z* e che vogliamo liberare la memoria e il puntatore a essa associata. Per far ciò usiamo la funzione *free*.



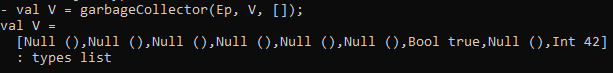
Possiamo osservare come il vecchio puntatore associato a *z* sia tornato in coda a *P*.

Ora osserviamo cosa c’è dentro *V*.



Dentro l’ambiente dei valori *V* c’è ancora la stringa “da eliminare” alla posizione 7, il vecchio valore associato precedentemente a *z*. Possiamo quindi osservare che la funzione *free* produce garbage.

Per pulire l’ambiente dei valori dopo la funzione *free*, ci serviamo di *garbageCollector*. Osserviamo come, dopo la sua esecuzione, la stringa “da eliminare” è stata correttamente rimossa dall’ambiente dei valori.



# Funzioni che lavorano sui tipi del linguaggio

In questa sezione mostriamo qualche esempio di funzione che lavora sui tipi definiti all’interno del mini-linguaggio.

La funzione *sumInt*, data una lista di elementi di tipo Int, restituisce la somma dei valori della lista. Possiamo osservare che il tipo di ciò che restituisce è un tipo di ML int.



La funzione *concatString*, data una lista di elementi Str, restituisce la concatenazione degli elementi della lista. Anche qui il tipo del valore restituito è un tipo di ML string.



# Conclusioni

Riassumiamo le funzionalità del mini-linguaggio:

* Inizializzazione corretta degli ambienti.
* Assegnazione di una variabile *x* definendo due coordinate *i,j* in cui *i* è decisa da una funzione *hash* e *j* è decisa dal numero di elementi assegnati al bucket in posizione *i* prima di *x*.
* Modifica del valore di una variabile già esistente.
* Valutazione di variabili.
* Ottenere il puntatore di una variabile inizializzata.
* Modificare il puntatore di una variabile inizializzata, rendendo possibile che due variabili abbiano associati puntatori che puntano alla stessa cella di memoria
* Valutazione tramite puntatore.
* Implementazione della funzione *free(x)* che libera l’ambiente dei nomi e dei puntatori degli elementi associati a *x*.
* Implementazione di un garbage collector che pulisce l’ambiente dei valori da elementi non referenziati da alcun puntatore.
* Implementazione di una piccola libreria che permette di lavorare su tipi definiti nel linguaggio.