RELAZIONE SECONDO PROGETTO – GIANLUCA PANZANI – CORSO A:

SUDDIVISIONE DEI FILE:
- `interprete.ml` → file contenente l'interprete del linguaggio e la relativa batteria dei test.
- `staticTypechecker.ml` → file contenente l'interprete del tipo delle espressioni definite nel linguaggio e la relativa batteria dei test.
- `RegoleOperazionali.pdf` → file contenente le regole operazionali relative alle funzionalitá introdotte nel linguaggio.

FILE `interprete.ml`:

IN GENERALE:
Contiene l'interprete di linguaggio con scoping statico introdotto a lezione ed esteso con il tipo Set. Questo valuta dinamicamente le espressioni restituiendo i valori ottenuti dalla valutazione

TIPI:
I tipi che ho definito nel linguaggio (oltre a quelli già presenti precedentemente) sono `setTypes` ed `args`. Il primo serve per definire i tipi dei valori che un insieme può contenere. Il secondo invece è un tipo ricorsivo che serve alla funzione `set_of` per riprodurre una rorta di funzione ad argomenti variabili. Infatti il costruttore `Of` ha come argomenti un valore di tipo `setTypes` e uno di tipo `args`. In questo modo si rende possibile l'inserimento di molteplici elementi all'interno di un insieme con un unico comando.
Inoltre ho aggiunto tra i valori esprimibili (`evT`) i costuttori:
```
String of string
Set of string * evT list
```

FUNZIONI:
Le principali funzioni che ho definito nel codice sono le seguenti (ce ne sono anche altre ma di secondaria importanza):
`switch_evTexp v:` permette di trasformare il valore passato come parametro nell'espressione corrispondente (di tipo `exp`).
`switch_setTypesString t:` permette di trasformare il tipo passato come parametro (di tipo `setTypes`) nella stringa corrispondente che permette di identificare il tipo dell'insieme affinchè possa essere creato.
`switch_evTsetTypes v:` permette di trasformare il valore passato come parametro nel tipo (`setTypes`) corrispondente.
`checkList t l:` controlla che il tipo dei valori nella lista associata all'insieme siano compatibili col tipo passato al costruttore dell'insieme stesso (viene chiamata dalla funzione `typecheck`).
`set_isEmpty set:` ritorna `true` se l'insieme passato come parametro è vuoto, `false` altrimenti.
`set_existsIn v set:` ritorna `true` se il valore `v` è presente nell'insieme `set`, `false` altrimenti.
`set_put v set:` inserisce il valore `v` nell'insieme `set`, se non presente.
`set_singleton t v:` crea un nuovo `set` di tipo `t` contenente il valore `v`.
`set_remove v set:` rimuove il valore `v` dall'insieme `set`, se presente.
`set_containsSet set set1:` ritorna `true` se tutti i valori presenti nell'insieme `set1` sono presenti nell'insieme `set`, `false` altrimenti.
`set_of t l:` crea un insieme di tipo `t` con gli elementi presenti nella lista `l`. Elimina possibili duplicati presenti nella lista chiamando la funzione `deleteDuplicates`.

`set_maxOf/set_minOf set:` ritorna il valore massimo/minimo presente nell'insieme, se di tipo intero.
`set_union set1 set2:` ritorna un insieme che è il risultato dell'unione insiemistica degli insieme `set1` e `set2`.
`set_intersection set1 set2:` ritorna un insieme che è il risultato dell'intersezione insiemistica dei due insiemi `set1` e `set2`.
`set_difference set1 set2:` ritorna un insieme che è il risultato della differenza insiemistica dei due insiemi `set1` e `set2`.

FILE `staticTypechecker.ml`:

IN GENERALE:
Contiene il typechecker statico che, dopo aver valutato le espressioni, restituisce il tipo ottenuto dalla valutazione stessa (il tutto staticamente).

IMPLEMENTAZIONE:
L'unica funzione presente, oltre a quelle per la gestione dell'ambiente, è un typechecker per il tipo associato agli insiemi, `setTypeCheck`, affinché si eviti l'uso improprio dei tipi con conseguente errore per tipo non valido.
Inoltre ho definito i valori esprimibili `tval`, l'albero di sintassi astratta `texp` e l'interprete `teval` per la valutazione del tipo delle espressioni.

ESECUZIONE:

COMPILAZIONE ED ESECUZIONE:
Per eseguire il codice basterà avviare ocaml con il comando `ocaml` e successivamente digitare:
```
#use "interprete.ml";;
```
oppure
```
#use "staticTypechecker.ml";;
```
in base a quale dei due file si vuole eseguire i test.

FORMATO OUTPUT:
All'output standard di ocaml ho aggiunto dei `print_string` che ne semplificano la leggibilità affiancando il codice eseguito all'output vero e proprio.
Possiamo distinguere le righe di output dalle altre per i "-" che precedono l'output stesso.
Inoltre aggiungerei il fatto che le batterie di test sono equivalenti per entrambi i file, cambia soltanto il tipo di output.

FINE