Progetto Linguaggi e Compilatori 2
Parte 3
Gruppo 1

Studenti:
Foschiani Luca
Marinato Riccardo
Passalenti Andrea

Assunzioni

Nel seguente progetto abbiamo considerato le scelte di sintassi concreta definite dal linguaggio *Go*. A seguire vengono riassunte una serie di assunzioni.

- Variabili e operazioni:
  - non è consientito dichiarare delle variabili con lo stesso identificatore nello stesso blocco, mentre è possibile se avviene su blocchi (e sottoblocchi) diversi;
  - non è possibile utilizzare delle right-expression senza assegnare il loro valore;
  - è consentita la dichiarazione di variabili globali al di fuori delle funzioni, ma non tramite la sintassi "short": ":=";
  - sono resi compatibili i seguenti tipi:
    - *float* ← *interi*
    - *interi* ← *booleani*
    - *float* ← *booleani*
- Funzioni e procedure:
  - non è possibile dichiarare più di una funzione/procedura con lo stesso identificatore;
  - nel corpo delle procedure non deve essere presente l'istruzione return, sempre richiesta invece nei corpi delle funzioni;
  - è consentito effettuare dei passaggi di parametri per valore (*val*) e per riferimento (*ref*).
- Comandi di controllo sequenza:
  - è stato usato l'operatore 'not' nelle condizioni degli if per veicolare correttamente i salti condizionati;
  - sono stati implementati i comandi per il controllo della sequenza richiesti nel testo del progetto (*condizionali semplici*, *iterazione determinata* (costrutto for) *e indeterminata* (nelle sue due forme: while e do-while)). Per quanto riguarda la sintassi utilizziamo sempre la stringa "for", in quanto Go non prevede alcuna sintassi

con il “while”.
Per quanto riguarda i costrutti di iterazione abbiamo inoltre i comandi `break` e `continue`;

- le guardie booleane dei controlli di sequenza vengono gestite tramite short-cut mentre le altre espressioni booleane vengono gestite senza short-cut;
- non è possibile definire controlli di sequenza al di fuori delle funzioni/procedure;
- è stato implementato il costrutto *try-catch* avente sintassi **try** *Statement* **catch** *Statement* come da specifica.

Scelte implementative

- Abbiamo generato *lexer* e *parser* tramite il tool BNFC, partendo dalla definizione di una gramatica iniziale;
- la gestione del *Type Checker* e del *Three Address Code* vengono fatte all'interno del parser;
- le funzioni *write* (*writeInt* eccetera) sono trattate come statement che prendono come argomento una right-expression, mentre le funzioni *read* (*readInt* e via dicendo) vengono viste come right-expressions e hanno una lista di parametri di input vuota.

Grammatica

Di seguito viene riportata la grammatica di partenza.

```
comment "//";
comment "/*" "*/";

entrypoints Start;

rules Boolean ::= "true" | "false";

ExpAnd.          RExp ::= RExp "&&" RExp;
ExpOr.           RExp ::= RExp "||" RExp;
```

```
ExpNot.          REXp ::= "!" REXp;
ExpEq.           REXp ::= REXp "==" REXp;
ExpNeq.          REXp ::= REXp "!=" REXp;
ExpLt.           REXp ::= REXp "<" REXp;
ExpLtE.          REXp ::= REXp "<=" REXp;
ExpGt.           REXp ::= REXp ">" REXp;
ExpGtE.          REXp ::= REXp ">=" REXp;
ExpAdd.          REXp ::= REXp "+" REXp;
ExpSub.          REXp ::= REXp "-" REXp;
ExpMul.          REXp ::= REXp "*" REXp;
ExpDiv.          REXp ::= REXp "/" REXp;
ExpMod.          REXp ::= REXp "%" REXp;
ExpNeg.          REXp ::= "-" REXp;
ExpRef.          REXp ::= "&" LExp;
ExpFuncEmpty.    REXp ::= Id "(" ")";
ExpFunc.         REXp ::= Id "(" [REXp] ")";
ExpVal.          REXp ::= Val;
ExpLExp.         REXp ::= LExp;
ExpPar.          REXp ::= "(" REXp ")";
StRead.          REXp ::= ReadT "(" ")";

Int .            Val ::= Integer;
Float .          Val ::= Double;
Char .           Val  ::= Char;
String .         Val ::= String;
Bool .           Val  ::= Boolean;

rules ReadT  ::= "readInt" | "readFloat" | "readChar" | "readString";

ExpId.           LExp ::= Id;
ExpArr.          LExp ::= LExp "[" REXp "]";
ExpDeref.        LExp ::= "*" REXp;

separator nonempty REXp ",";

Entry.           Start ::= "package" Id [Decl];

separator Decl "";
```

```
DeclVar.          Decl ::= "var"  [Id] Type;
DeclVarInit.      Decl ::= "var"  [Id] "=" [RExp];
DeclVarTypeInit.  Decl ::= "var"  [Id] Type "=" [RExp];
DeclVarShort.     ShortVarDecl ::= [Id] ":=" [RExp];

TVoid.            Type ::= "void";
TInt.             Type ::= "int";
TBool.            Type ::= "bool";
TFloat.           Type ::= "float";
TChar.            Type ::= "char";
TString.          Type ::= "string";
TArray.           Type ::=  "[" Integer "]" Type;
TPointer.         Type ::= "*" Type;

token Id (letter | '_')(letter | digit | '_')*;
separator nonempty Id "," ;

DeclFun.          Decl ::= "func" Id "(" [Param] ")" Type Block;
DeclProc.         Decl ::= "func" Id "(" [Param] ")" "void" Block;

separator Param ",";

Parameter.        Param ::=  [Id] Type;
ParameterPass.    Param ::=  Pass [Id] Type;

PassVal.          Pass  ::=  "val";
PassRef.          Pass  ::=  "ref";

separator Stmt "";

BodyBlock.        Block ::= "{" [Stmt] "}";

StDecl.           Stmt ::= Decl;
StBlock.          Stmt ::= Block;
StSmpl.           Stmt ::= StmtSmpl;
StIf.             Stmt ::= "if" RExp Block;
StIfElse.         Stmt ::= "if" RExp Block "else" Block;
```

```
StWhile.          Stmt ::= "for" RExp Block;
StDoWhile.        Stmt ::= "do" Block "for" RExp;
StFor.            Stmt ::= "for" [StmtSmpl] ";" RExp ";" [StmtSmpl] Block;
StBreak.          Stmt ::= "break";
StContinue.       Stmt ::= "continue";
StReturn.         Stmt ::= "return" RExp;
StTryCatch.       Stmt ::= "try" Stmt "catch" Stmt;
StWrite.          Stmt ::= WriteT "(" RExp ")";

rules WriteT ::= "writeInt" | "writeFloat" | "writeChar" | "writeString";

StShortVarDecl.   StmtSmpl ::= ShortVarDecl;
StExp.            StmtSmpl ::= RExp;
StAsgn.           StmtSmpl ::= LExp "=" RExp;

[].              [StmtSmpl] ::= ;
(:[]).           [StmtSmpl] ::= StmtSmpl;
```

Parser

Gran parte del progetto è stato sviluppato all'interno del parser.
Abbiamo definito una serie di attributi relativi alla *grammatica attributata*.
Gli attributi sono stati scelti in base alle nostre esigenze, che erano fondamentalmente il type checking (e gli altri tipi di controlli) e la generazione del three-address code.
Gli enviroment sono gestiti attraverso gli attributi **EnvVar**/**EnvVarNew** ed **EnvFun**/**EnvFunNew** e corrispondono a liste di tipi di dato **ElemVar** ed **ElemFun**. In particolare, gli attributi **EnvVar** ed **EnvFun** rappresentano, per ogni nodo, l'ambiente derivante dal nodo padre, mentre **EnvVarNew** ed **EnvFunNew** definiscono l'ambiente ottenuto da tutti i nodi figli.
La gestione/modifica degli Environment viene gestita dalle strutture e dalle funzioni presenti in *Env.hs*.
Per quanto riguarda la dichiarazione delle precedenze e l'associatività degli operatori, sono state gestite a livello di parser, nel file *ParGo.y*.

Type Checker

Il Type Checker viene gestito interamente all'interno del parser. In particolare, per ogni espressione valutata si effettua un controllo diretto attraverso **Ok** e **Bad** che, in caso di errore, gestiscono l'errore tramite la stampa di un messaggio in output indicante la linea dell'errore e la sua causa.
Tutte le funzioni di controllo utilizzate dal Type Checker sono state definite internamente al parser per praticità.
Tra le diverse assunzioni fatte in fase di type checking, ricordiamo che i booleani sono compatibili con interi e float, mentre gli interi sono compatibili con i float.

Three Address Code

Il Three Address Code viene generato interamente nel parser, utilizzando un modulo esterno (*TAC.hs*) contenente la struttura dati per gestirne la generazione e le funzioni addette alla stampa.

Type System

A seguire il *Type System* relativo al linguaggio implementato.
Definiamo Γ essere l'ambiente e τ essere l'insieme dei tipi di base da noi considerati: `int, float, char, string` e `boolean`.

Basic Judgments

Γ è un ambiente ben formato:
$$\Gamma \vdash \diamond$$

Type è un tipo ben formato in Γ
$$\Gamma \vdash \text{Type}$$

Stmt è un tipo ben formato in Γ
$$\Gamma \vdash \text{Stmt}$$

RExp è un tipo ben formato in Γ
$$\Gamma \vdash \text{RExp} : \text{Type}$$

Regole

$$(\textbf{empty})\ \frac{}{\emptyset \vdash \diamond}$$

$$(\textbf{Id})\ \frac{\Gamma \vdash \diamond, \texttt{ident} \notin dom(\Gamma)}{\Gamma, \texttt{ident} \vdash \diamond}$$

Tipi Base

Vediamo di seguito in maniera esplicita tutti i tipi considerati.

$$(\textbf{Int})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash n : \texttt{int}}$$

$$(\textbf{Float})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash f : \texttt{float}}$$

$$(\textbf{Char})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash c : \texttt{char}}$$

$$(\textbf{String})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash s : \texttt{string}}$$

$$(\textbf{Boolean})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash b : \texttt{boolean}}$$

Tipi Composti

$$(\textbf{Ref})\ \frac{\Gamma \vdash \texttt{ident}}{\Gamma \vdash \&\texttt{ident}}$$

- Per $\tau \in \{$ `int, float, char, string, boolean` $\}$

$$(\textbf{Deref})\ \frac{\Gamma \vdash \tau}{\Gamma \vdash *_\tau}$$

- Per $\tau \in \{$ `int` $\}$

$$(\textbf{Arr})\ \frac{\Gamma \vdash \texttt{ident} \quad \Gamma \vdash RExp : \tau}{\Gamma \vdash \texttt{ident}\ [\ RExp\ ]}$$

Assegnamento

- Per $\tau \in \{$ `int, float, char, string, boolean` $\}$

$$(\textbf{Asgn})\ \frac{\Gamma \vdash LExp : \tau \quad \Gamma \vdash RExp : \tau}{\Gamma \vdash LExp \texttt{ = } RExp}$$

Per l'assegnamento inoltre abbiamo le usuali compatibilità fra tipi, `boolean` $\rightarrow$ `int` $\rightarrow$ `float`

Operatori Relazionali

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{And})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ \&\& } RExp2 : \tau}$$

$$(\textbf{Or})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ || } RExp2 : \tau}$$

$$(\textbf{Not})\ \frac{\Gamma \vdash RExp : \tau}{\Gamma \vdash \texttt{not } RExp : \tau}$$

- Per $\tau \in \{$ `int, float, char, string, boolean` $\}$

$$(\textbf{Eq})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ == } RExp2 : \texttt{boolean}}$$

$$(\textbf{Neq})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ != } RExp2 : \texttt{boolean}}$$

$$(\textbf{Lt})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ < } RExp2 : \texttt{boolean}}$$

$$(\textbf{LtE})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ <= } RExp2 : \texttt{boolean}}$$

$$(\textbf{Gt})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ > } RExp2 : \texttt{boolean}}$$

$$(\textbf{GtE})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ >= } RExp2 : \texttt{boolean}}$$

Operazioni Aritmetiche

Addizione

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{Add 1})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `int` $\}$

$$(\textbf{Add 2})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \tau}$$

- Per $\tau \in \{$ `float` $\}$

$$(\textbf{Add 3})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \tau}$$

Addizione con casting implicito

- Per $\tau \in \{$ `bool, int` $\}$ con una delle due espressioni `int` e l'altra `boolean`

$$(\textbf{Add 4})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `bool, float` $\}$ con una delle due espressioni `float` e l'altra `boolean`

$$(\textbf{Add 5})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \texttt{float}}$$

- Per $\tau \in \{$ `int, float` $\}$ con una delle due espressioni `float` e l'altra `int`

$$(\textbf{Add 6})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ + } RExp2 : \texttt{float}}$$

Sottrazione

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{Sub 1})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `int` $\}$

$$(\textbf{Sub 2})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \tau}$$

- Per $\tau \in \{$ `float` $\}$

$$(\textbf{Sub 3})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \tau}$$

Sottrazione con casting implicito

- Per $\tau \in \{$ `boolean, int` $\}$ con una delle due espressioni `int` e l'altra `boolean`

$$(\textbf{Sub 4})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `boolean, float` $\}$ con una delle due espressioni `float` e l'altra `boolean`

$$(\textbf{Sub 5})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \texttt{float}}$$

- Per $\tau \in \{$ `int, float` $\}$ con una delle due espressioni `float` e l'altra `int`

$$(\textbf{Sub 6})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ - } RExp2 : \texttt{float}}$$

Moltiplicazione

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{Mul 1})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `int` $\}$

$$(\textbf{Mul 2})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \tau}$$

- Per $\tau \in \{$ `float` $\}$

$$(\textbf{Mul 3})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \tau}$$

Moltiplicazione con casting implicito

- Per $\tau \in \{$ `boolean, int` $\}$ con una delle due espressioni `boolean` e l'altra `int`

$$(\textbf{Mul 4})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \texttt{int}}$$

- Per $\tau \in \{$ `boolean, float` $\}$ con una delle due espressioni `boolean` e l'altra `float`

$$(\textbf{Mul 5})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \texttt{float}}$$

- Per $\tau \in \{$ `int, float` $\}$ con una delle due espressioni `float` e l'altra `int`

$$(\textbf{Mul 6})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ * } RExp2 : \texttt{float}}$$

Divisione

- Per $\tau_1, \tau_2 \in \{$ `boolean`, `int`, `float` $\}$

$$(\textbf{Div})\ \frac{\Gamma \vdash RExp1 : \tau_1 \quad \Gamma \vdash RExp2 : \tau_2}{\Gamma \vdash RExp1 \texttt{ / } RExp2 : \texttt{float}}$$

Modulo

- Per $\tau_1, \tau_2 \in \{$ `boolean`, `int` $\}$

$$(\textbf{Mod 2})\ \frac{\Gamma \vdash RExp1 : \tau \quad \Gamma \vdash RExp2 : \tau}{\Gamma \vdash RExp1 \texttt{ \% } RExp2 : \texttt{int}}$$

Neg

- Per $\tau \in \{$ `int` $\}$

$$(\textbf{Neg 1})\ \frac{\Gamma \vdash RExp : \tau}{\Gamma \vdash \texttt{-}RExp : \tau}$$

- Per $\tau \in \{$ `float` $\}$

$$(\textbf{Neg 2})\ \frac{\Gamma \vdash RExp : \tau}{\Gamma \vdash \texttt{-}RExp : \tau}$$

Controlli di sequenza

If-then

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{If})\ \frac{\Gamma \vdash RExp : \tau \quad \Gamma \vdash Block}{\Gamma \vdash \texttt{if}\ RExp\ \{Block\}}$$

If-then-else

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{IfElse})\ \frac{\Gamma \vdash RExp : \tau \quad \Gamma \vdash Block1 \quad \Gamma \vdash Block2}{\Gamma \vdash \texttt{if}\ RExp\ \{Block1\}\ \texttt{else}\ \{Block1\}}$$

While

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{While})\ \frac{\Gamma \vdash RExp : \tau \quad \Gamma \vdash Block}{\Gamma \vdash \texttt{for}\ RExp\ \{Block\}}$$

Do-while

- Per $\tau \in \{$ `boolean` $\}$

$$(\textbf{DoWhile})\ \frac{\Gamma \vdash RExp : \tau \quad \Gamma \vdash Block}{\Gamma \vdash \texttt{do}\ \{Block\}\ \texttt{for}\ RExp}$$

For

- Per $\tau \in \{$ `boolean` $\}$, con $\texttt{stmtSmpl}_1$ e $\texttt{stmtSmpl}_2$ statementSimple

$$(\textbf{For})\ \frac{\Gamma \vdash stmtSmpl_1 \quad \Gamma \vdash RExp : \tau \quad \Gamma \vdash stmtSmpl_2 \quad \Gamma \vdash Block}{\Gamma \vdash \texttt{for}\ \ stmtSmpl_1\ RExp\ stmtSmpl_2\ \{Block\}}$$

Eccezioni

- Con $\texttt{stmt}_1$ e $\texttt{stmt}_2$ statement (`Stmt`)

$$(\textbf{Try-Catch})\ \frac{\Gamma \vdash stmt_1 \quad \Gamma \vdash tmt_2}{\Gamma \vdash \texttt{try}\ \ stmt_1\ \texttt{catch}\ stmt_2}$$

Funzioni e procedure

- Con τ qualunque tipo (semplice o composto) e `params` una variabile di tipo `TypeList` (lista di tipi)

$$(\textbf{DeclFun})\ \frac{\Gamma \vdash \texttt{ident} \quad \Gamma \vdash params : TypeList \quad \Gamma \vdash Block}{\Gamma \vdash func\ \texttt{ident}\ (TypeList)\ \tau\ \{Block\}}$$

$$(\textbf{DeclProc})\ \frac{\Gamma \vdash \texttt{ident} \quad \Gamma \vdash params : TypeList \quad \Gamma \vdash Block}{\Gamma \vdash func\ \texttt{ident}\ (TypeList)\ void\ \{Block\}}$$

$$(\textbf{WriteInt})\ \frac{\Gamma \vdash RExp : \texttt{int}}{\Gamma \vdash \texttt{writeInt}\ (\ RExp\ )}$$

$$(\textbf{WriteFloat})\ \frac{\Gamma \vdash RExp : \texttt{float}}{\Gamma \vdash \texttt{writeFloat}\ (\ RExp\ )}$$

$$(\textbf{WriteChar})\ \frac{\Gamma \vdash RExp : \texttt{char}}{\Gamma \vdash \texttt{writeChar}\ (\ RExp\ )}$$

$$(\textbf{WriteString})\ \frac{\Gamma \vdash RExp : \texttt{string}}{\Gamma \vdash \texttt{writeString}\ (\ RExp\ )}$$

$$(\textbf{ReadInt})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{readInt}\ (\ )\ :\ \texttt{int}}$$

$$(\textbf{ReadFloat})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{readFloat}\ (\ )\ :\ \texttt{float}}$$

$$(\textbf{ReadChar})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{readChar}\ (\ )\ :\ \texttt{char}}$$

$$(\textbf{ReadString})\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{readString}\ (\ )\ :\ \texttt{string}}$$

Test Case

Sono stati preparati dei test-case significativi che possono essere eseguiti utilizzando i comandi **make demo1**, **make demo2**, **make demo3** e **make demo4**. Altri comandi per la compilazione del codice sono presenti all'interno del makefile ma non sono necessari al fine di runnare i test.