Complementi di Linguaggi

Report Finale

Name: Lorenzo Campidelli
StudentID: 0001099608

Name: Federico Augelli
StudentID: 0001097518

15th July 2023

Contents

1 Esercizio 1 **3**

2 Esercizio 2 **4**

3 Esercizio 3 **4**

4 Esercizio 4 **11**

1 Esercizio 1

Il codice da eseguire viene scritto sul file input.txt. Quando il programma trova un errore lessicale o sintattico fa una stampa su linea di comando:

```
Syntax errors: check out/errors.txt for details
```

e stampa gli errori relativi sul file errors.txt.
3 esempi di errori lessicali:

- singolo carattere non valido.
  input.txt:

  ```
  '
  ```

  output.txt:

  ```
  Line 1 in position 0 --Error: invalid character '
  ```

- carattere non valido dentro il codice.
  input.txt:

  ```
  int x;
  x = %5;
  x
  ```

  output.txt:

  ```
  Line 2 in position 4 --Error: invalid character %
  ```

- serie di caratteri non validi.
  input.txt:

  ```
  int$ x;
  x = %5;
  _'x
  ```

  output.txt:

  ```
  Line 1 in position 3 --Error: invalid character $
  Line 2 in position 4 --Error: invalid character %
  Line 3 in position 0 --Error: invalid character _
  Line 3 in position 1 --Error: invalid character '
  ```

2 Esercizio 2

Il programma crea l'abstract syntax tree tramite i metodi visit dei token nel parser. Una volta generato l'albero viene creata la symbol table vuota e riempita tramite il controllo semantico con le funzioni checkSemantics dei nodi dell'albero. In particolare:

- il controllo di identificatori non dichiarati viene effettuato in IdNode.
- il controllo di funzioni non dichiarate viene effettuato in CallNode.
- il controllo di identificatori dichiarati piú volte nell'ambiente viene effettuato in DecNode.
- il controllo di funzioni dichiarate piú volte nell'ambiente viene effettuato in FunNode.

3 Esercizio 3

La correttezza dei tipi viene controllata dalle funzioni typeCheck dei nodi dell'ast. In particolare il controllo di numero e tipi dei parametri attuali rispetto ai parametri formali viene effettuato in CallNode.

Il compilatore controlla anche l'inizializzazione di una variabile. Per farlo é stato aggiunto un campo alla STEntry che indica lo stato di inizializzazione. Un identificatore ha campo inizializzazione falso appena dichiarato, quando viene fatto un assegnamento (nodo AsgNode) il campo diventa vero. Se l'identificatore viene inizializzato all'interno di un If ci sono due casi:

- se l'istruzione if ha solo il ramo then all'uscita dall'istruzione l'identificatore non viene considerato inizializzato perché non si puó sapere a priori se la condizione dell'if é vera o falsa.
- se l'istruzione if ha sia ramo then che ramo else l'identificatore viene considerato inizializzato all'uscita dall'istruzione solo se é stato inizializzato in entrambi i rami, poiché non si puó sapere a priori quale ramo verrá percorso dal programma.

Per l'assunzione dell'esercizio non viene trattato il caso di inizializzazione o utilizzo di identificatore in una funzione. L'utilizzo di un identificatore non inizializzato viene controllato nel nodo IdNode.

Il compilatore utilizza le seguenti regole semantiche:

Program:

$$\text{ProgExp}\frac{\varnothing \quad \vdash \quad EXP : T}{\varnothing \vdash EXP : T}$$

Il programma non ha dichiarazioni né statements quindi non viene creata la tabella dei simboli. Tipo e risultato del programma sono quelli dell'espressione.

$$\text{ProgFull}\frac{\varnothing[] \vdash DEC : \Gamma \quad \Gamma \vdash STM : \Gamma' \quad \Gamma' \vdash EXP : T}{\varnothing \vdash DEC \quad STM \quad EXP : T}$$

Programma completo. Tipo e risultato del programma sono quelli dell'espressione finale.

$$\text{ProgFull1}\frac{\varnothing[] \vdash DEC : \Gamma \quad \Gamma \vdash STM : \Gamma'}{\varnothing \vdash DEC \quad STM : VOID}$$

Programma senza espressione finale, crea e modifica la tabella dei simboli e ritorna il tipo void. Per implementazione i programmi void hanno come risultato 0.

$$\text{ProgFull2}\frac{\varnothing[] \vdash DEC : \Gamma \quad \Gamma \vdash EXP : T}{\varnothing \vdash DEC \quad EXP : T}$$

Programma con solo dichiarazioni e espressione finale. Tipo e risultato del programma sono quelli dell'espressione finale.

$$\text{ProgDec}\frac{\varnothing[] \vdash DEC : \Gamma}{\varnothing \vdash DEC : VOID}$$

Programma di solo dichiarazioni. Ritorna tipo void con risultato 0.

Dichiarazioni:

$$\text{VarDec}\frac{ID \notin DOM(TOP(\Gamma))}{\Gamma \vdash T \quad ID; : \Gamma'}$$

Dichiarazione di un identificatore, Γ' é la tabella dei simboli con il nuovo identificatore inserito.

$$\text{FunDec}\frac{f \notin DOM(TOP(\Gamma)) \quad \Gamma[p_1 \to T_1..p_n \to T_n, f(T_1..T_n) \to T] \vdash \{BODY\} : T}{\Gamma \vdash T \quad f(T_1 p_1..T_n p_n)\,\{BODY\} : \Gamma'}$$

Dichiarazione di una funzione, Γ' é la tabella dei simboli con la nuova funzione inserita.

$$\text{SeqDec}\frac{\Gamma \vdash DEC_1 : \Gamma' \quad \Gamma' \vdash DEC_2 : \Gamma''}{\Gamma \vdash DEC_1 \quad DEC_2 : \Gamma''}$$

Sequenza di dichiarazioni.

Body:

$$\text{Body}\frac{\Gamma[] \vdash DEC : \Gamma' \quad \Gamma' \vdash STM : \Gamma'' \quad \Gamma'' \vdash EXP : T}{\Gamma \vdash \{DEC \quad STM \quad EXP\} : T}$$

Body completo di una funzione. Il tipo di ritorno é quello dell'espressione finale.

$$\text{Body1}\frac{\Gamma[] \vdash DEC : \Gamma' \quad \Gamma' \vdash STM : \Gamma''}{\Gamma \vdash \{DEC \quad STM\} : VOID}$$

Body di funzione senza espressione, la funzione é di tipo void.

$$\text{Body2}\frac{\Gamma[] \vdash DEC : \Gamma' \quad \Gamma' \vdash EXP : T}{\Gamma \vdash \{DEC \quad EXP\} : T}$$

Body di funzione senza statements. Il tipo della funzione é quello dell'espressione finale.

$$\text{Body3}\frac{\Gamma \vdash STM : \Gamma' \quad \Gamma' \vdash EXP : T}{\Gamma \vdash \{STM \quad EXP\} : T}$$

Body di funzione senza dichiarazioni locali. Il tipo della funzione é quello dell'espressione finale.

$$\text{BodyDec}\frac{\Gamma[] \vdash DEC : \Gamma'}{\Gamma \vdash \{DEC\} : VOID}$$

Body di funzione con solo dichiarazioni locali. Il tipo della funzione é void.

$$\text{BodyStm}\frac{\Gamma \vdash STM : \Gamma'}{\Gamma \vdash \{STM\} : VOID}$$

Body di funzione con solo statements. Il tipo della funzione é void.

$$\text{BodyExp}\frac{\Gamma \vdash EXP : T}{\Gamma \vdash \{EXP\} : T}$$

Body di funzione con solo un'espressione. Il tipo della funzione é lo stesso dell'espressione.

$$\text{BodyVoid}\frac{}{\Gamma \vdash \{\} : VOID}$$

Body di funzione vuoto, il tipo della funzione é void.

Statements:

$$\text{Asg}\frac{\Gamma \vdash ID : T \quad \Gamma \vdash EXP : T \quad f(\Gamma) = \Gamma'}{\Gamma \vdash ID = EXP; : \Gamma'}$$

Assegnamento di un valore ad un identificatore. Γ' é il risultato di una funzione che controlla se l'identificatore viene inizializzato, se é il primo assegnamento dell'identificatore viene aggiornato il suo stato di inizializzazione nella tabella dei simboli, se era giá inizializzato Γ' é uguale a Γ.

$$\text{FunStm}\frac{\Gamma \vdash f : T_1 \times ... \times T_\text{n} \to T \quad (\Gamma \vdash EXP_\text{i} : T_\text{i})_\text{i} \in {}_{1..\text{n}} \quad f(\Gamma) = \Gamma'}{\Gamma \vdash f(EXP_1..EXP_\text{n}); : \Gamma'}$$

Chiamata di funzione come statement. Γ viene aggiornato a Γ' se nel corpo della funzione ci sono inizializzazioni di identificatori, nel caso non ci siano Γ' é uguale a Γ.

$$\text{IfStm}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash STM_1 : \Gamma' \quad \Gamma \vdash STM_2 : \Gamma'' \quad f(\Gamma', \Gamma'') = \Gamma''' = \Gamma' \cap \Gamma''}{\Gamma \vdash if(EXP)\,\{STM_1\}\,else\,\{STM_2\} : \Gamma'''}$$

Istruzione if come statement. Γ''' é uguale all'intersezione fra Γ' dal ramo then e Γ'' dal ramo else, cioé é uguale a Γ in cui gli identificatori inizializzati in entrambi i rami dell'if sono inizializzati.

$$\text{IfThenStm}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash STM : \Gamma'}{\Gamma \vdash if(EXP)\,\{STM\} : \Gamma}$$

Istruzione if statement con solo il ramo then. Γ finale é uguale a quello iniziale perché la condizione potrebbe non essere vera, quindi se ci sono inizializzazioni all'interno del ramo then non vengono considerate valide.

$$\text{SeqStm}\frac{\Gamma \vdash STM_1 : \Gamma' \quad \Gamma' \vdash STM_2 : \Gamma''}{\Gamma \vdash STM_1 \quad STM_2 : \Gamma''}$$

Sequenza di statements.

Expressions:

$$\text{Not}\frac{\Gamma \vdash EXP : BOOL}{\Gamma \vdash !EXP : BOOL}$$

Espressione not.

$$\text{Mult}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad * : INT \times INT \to INT}{\Gamma \vdash EXP_1 * EXP_2 : INT}$$

Espressione moltiplicazione.

$$\text{Div}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad / : INT \times INT \to INT}{\Gamma \vdash EXP_1 / EXP_2 : INT}$$

Espressione divisione.

$$\text{Plus}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad + : INT \times INT \to INT}{\Gamma \vdash EXP_1 + EXP_2 : INT}$$

Espressione somma.

$$\text{Minus}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad - : INT \times INT \to INT}{\Gamma \vdash EXP_1 - EXP_2 : INT}$$

Espressione sottrazione.

$$\text{Great}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad > : INT \times INT \to BOOL}{\Gamma \vdash EXP_1 > EXP_2 : BOOL}$$

Espressione comparazione maggiore.

$$\text{Less}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad < : INT \times INT \to BOOL}{\Gamma \vdash EXP_1 < EXP_2 : BOOL}$$

Espressione comparazione minore.

$$\text{Geq}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad >= : INT \times INT \to BOOL}{\Gamma \vdash EXP_1 >= EXP_2 : BOOL}$$

Espressione comparazione maggiore uguale.

$$\text{Leq}\frac{\Gamma \vdash EXP_1 : INT \quad \Gamma \vdash EXP_2 : INT \quad <=: INT \times INT \to BOOL}{\Gamma \vdash EXP_1 <= EXP_2 : BOOL}$$

Espressione comparazione minore uguale.

$$\text{Equals}\frac{\Gamma \vdash EXP_1 : T \quad \Gamma \vdash EXP_2 : T \quad ==: T \times T \to BOOL}{\Gamma \vdash EXP_1 == EXP_2 : BOOL}$$

Espressione di uguaglianza fra due elementi. Il tipo degli elementi é polivalente ma deve essere uguale fra i due.

$$\text{And}\frac{\Gamma \vdash EXP_1 : BOOL \quad \Gamma \vdash EXP_2 : BOOL \quad \&\& : BOOL \times BOOL \to BOOL}{\Gamma \vdash EXP_1 \&\& EXP_2 : BOOL}$$

Espressione and.

$$\text{Or}\frac{\Gamma \vdash EXP_1 : BOOL \quad \Gamma \vdash EXP_2 : BOOL \quad || : BOOL \times BOOL \to BOOL}{\Gamma \vdash EXP_1 || EXP_2 : BOOL}$$

Espressione or.

$$\text{IfExpSimple}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash EXP_1 : T \quad \Gamma \vdash EXP_2 : T}{\Gamma \vdash if(EXP) \quad then\,\{EXP_1\} \quad else\,\{EXP_2\} : T}$$

Istruzione if come expression. Il tipo di ritorno é quello dei rami then ed else, che devono essere dello stesso tipo.

$$\text{IfExpFull}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash STM_1 : \Gamma' \quad \Gamma' \vdash EXP_1 : T \quad \Gamma \vdash STM_2 : \Gamma'' \quad \Gamma'' \vdash EXP_2 : T}{\Gamma \vdash \quad if(EXP) \quad then\,\{STM_1 \quad EXP_1\} \quad else\,\{STM_2 \quad EXP_2\} : T}$$

Istruzione if expression completa di statements. Il tipo di ritorno é quello dei rami then ed else, che devono essere dello stesso tipo.

$$\text{IfExpStmThen}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash STM : \Gamma' \quad \Gamma' \vdash EXP_1 : T \quad \Gamma \vdash EXP_2 : T}{\Gamma \vdash \quad if(EXP) \quad then\,\{STM \quad EXP_1\} \quad else\,\{EXP_2\} : T}$$

Istruzione if expression con statements solo nel ramo then. Il tipo di ritorno é quello dei rami then ed else, che devono essere dello stesso tipo.

$$\text{IfExpStmElse}\frac{\Gamma \vdash EXP : BOOL \quad \Gamma \vdash EXP_1 : T \quad \Gamma \vdash STM : \Gamma' \quad \Gamma' \vdash EXP_2 : T}{\Gamma \vdash \quad if(EXP) \quad then\,\{EXP_1\} \quad else\,\{STM \quad EXP_2\} : T}$$

Istruzione if expression con statements solo nel ramo else. Il tipo di ritorno é quello dei rami then ed else, che devono essere dello stesso tipo.

$$\text{Paren}\frac{\Gamma \vdash EXP : T}{\Gamma \vdash (EXP) : T}$$

Espressione compresa fra parentesi.

$$\text{FunExp}\frac{\Gamma : f : T_1 \times .. \times T_n \to T \qquad (\Gamma \vdash EXP_i : T_i)_i \in {}_{1\ ..\ n}}{\Gamma \vdash f(EXP_1..EXP_n) : T}$$

Chiamata di funzione come espressione.
Assiomi:

$$\text{Id}\frac{\Gamma(ID) = T}{\Gamma \vdash ID : T}$$

$$\text{Int}\frac{}{\Gamma \vdash NUM : INT}$$

$$\text{True}\frac{}{\Gamma \vdash TRUE : BOOL}$$

$$\text{False}\frac{}{\Gamma \vdash FALSE : BOOL}$$

Assiomi riconosciuti.

Codici da verificare:

Codice 1:

```
int a; int b; int c ; c = 2 ;
    if (c > 1) { b = c ; } else { a = b ; }
```

Il codice é errato perché b viene utilizzato come espressione in un assegnamento senza mai essere stato inizializzato. Viene stampato sul terminale:

```
Id b not initialized in else branch
```

Codice 2:

```
int a; int b; int c ;
void f(int n){
    int x ; int y ;
    if (n > 0) { x = n ;} else { y = n+x ;}
}
c = 1 ; f(0) ;
```

L'errore é l'utilizzo di x come espressione in una somma senza essere stato inizializzato. Viene stampato sul terminale:

```
Id x not initialized in else branch
```

Codice 3:

```
void h(int n){
    int x ; int y ;
    if (n==0){ x = n+1 ;} else { h(n-1) ; x = n ; y = x ;}
}
h(5) ;
```

Il codice é corretto, il programma é di tipo void (quindi il risultato finale é 0).

Codice 4:

```
int a;
void h(int n){
    int x ; int y ;
    if (n==0){ x = n+1 ;} else { h(n-1) ; y = x ;}
}
h(5) ;
```

L'errore é l'utilizzo di x come espressione in un assegnamento senza essere stato inizializzato. Viene stampato sul terminale:

```
Id x not initialized in else branch
```

4 Esercizio 4

Codice 1:

```
int x ;
void f(int n){
    if (n == 0) { n = 0 ; }       // n e‘ gia‘ uguale a 0; equivale a fare skip
    else { x = x * n ; f(n-1) ; }
}
x = 1 ;
f(10)
```

per come é stato implementato il codice la variabile x non viene considerata inizializzata (per l'assunzione di non accedere a variabili globali all'interno di una funzione). Per farlo funzionare basta una piccola modifica:

```
int x ;
void f(int n){
    x = 1 ;
    if (n == 0) { n = 0 ; }       // n e‘ gia‘ uguale a 0; equivale a fare skip
    else { x = x * n ; f(n-1) ; }
}
f(10)
```

mettendo l'inizializzazione di x nel corpo della funzione non c'é piú errore. Il programma é di tipo void e il risultato finale é 0.

Codice 2:

```
int u ;
int f(int n){
    int y ;
    y = 1 ;
    if (n == 0) { y }
    else { y = f(n-1) ; y*n }
}
u = 6 ;
f(u)
```

Il programma é di tipo int e il risultato finale é 720.

Codice 3:

```
int u ;
void f(int m, int n){
    if (m>n) { u = m+n ;}
    else { int x ; x = 1 ; f(m+1,n+1) ; }
}
f(5,4) ; u
```

il codice scritto cosí risulta in un syntax error. Nel file errors.txt compare l'errore:

```
Line 4 position 8 --Error: extraneous input 'int' expecting {'if', ID}
```

per correggerlo basta modificare il codice come segue:

```
int u ;
void f(int m, int n){
    int x ;
    if (m>n) { u = m+n ;}
    else { x = 1 ; f(m+1,n+1) ; }
}
f(5,4) ; u
```

questo codice ha un errore semantico, la variabile u non viene considerata inizializzata in quanto l'assegnamento viene eseguito solo nel ramo then dell'if. Per correggere l'errore serve un'ulteriore modifica:

```
int u ;
void f(int m, int n){
    int x ;
    u = 0;
    if (m>n) { u = m+n ;}
    else { x = 1 ; f(m+1,n+1) ; }
}
f(5,4) ; u
```

inizializzando la variabile u fuori dall'if (il valore non importa) il codice viene eseguito. Il programma é di tipo int e il risultato é 9.

Se la funzione viene invocata come f(4, 5) il calcolo non arriva mai a termine, risultando nell'errore:

```
ArrayIndexOutOfBoundsException: Index -1 out of bounds for length 1000
```