|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Report     |  |  | | --- | --- | | Riferimento |  | | Versione | 1.1 | | Data | 02/10/19 | |

RevisionHistory

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 30/09/2019 | 1.0 | Prima stesura | F. Mazzotta |
| 02/10/2019 | 1.1 | Aggiunto testing di regressione | F. Mazzotta |
| 14/10/2019 | 1.2 | Aggiornamento documento | F. Mazzotta |

Sommario

[RevisionHistory 1](#_Toc22408219)

[1. Descrizione del documento. 1](#_Toc22408220)

[2. Test Case Specification 1](#_Toc22408221)

[2.1 Test case Data flow 1](#_Toc22408222)

[3 Testing di regressione 1](#_Toc22408232)

[3.1 Test case specification per testing di regressione 1](#_Toc22408234)

[4 Report del testing 1](#_Toc22408244)

1. Descrizione del documento.

Il presente documento viene redatto per descrivere i test case eseguiti sul compilatore Yaspl. Di seguito vengono riportati i test case per le modifiche aggiunte e i test case per il testing di regressione.

2. Test Case Specification

2.1 Test case Data flow

TC\_DF\_1

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_1** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome media | Nodo 10: media=sum+n | | Variabile di nome media | Nodo 11: “Valore della media %f”, media -> | | Variabile di nome media | Nodo 15: esito=media>0 | | Variabile di nome media | Nodo 16: media=i+n | | Variabile di nome media | Nodo 17: sum=media+96 | | Variabile di nome media | Nodo 18: n=n+media | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(10,11,media),(10,15,media),(16,17,media),(16,18,media)} e il token finale (16,media). | |

TC\_DF\_2

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_2** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome n | Nodo 3: n <- double | | Variabile di nome n | Nodo 5: while(i<n) | | Variabile di nome n | Nodo 6: (lato true): n=50 | | Variabile di nome n | Nodo 8: i=i+n | | Variabile di nome n | Nodo 9: n=i+50 | | Variabile di nome n | Nodo 13: media=sum+n | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(3,5,n),(6,8,n),(9,13,n)} e il token finale (9,n) | |

TC\_DF\_3

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_3** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome x | Nodo 7: x <- double | | Variabile di nome x | Nodo 8: sum=sum+x | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(7,8,x)} e il token finale (7,x) | |

TC\_DF\_4

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_4** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome y | Nodo 5: y=1 | | Variabile di nome y | Nodo 8: y=y+n | | Variabile di nome y | Nodo 10 :”Risultato %d” , y -> | | Variabile di nome y | Nodo 12: y <-double | | Variabile di nome y | Nodo 15: raggio=raggio+y | | Variabile di nome y | Nodo 20: n=n+y | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(5,8,y),(5,10,y),(12,15,y),(12,20,y)} e il token finale (12,y) | |

TC\_DF\_5

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_5** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome a | Nodo10: a<-int | | Variabile di nome a | Nodo 11: “Valore della media %f”, a -> | | Variabile di nome a | Nodo 15: y=a | | Variabile di nome a | Nodo 16 a=b+96 | | Variabile di nome a | Nodo 19: “Valore finale: %d”, a -> | | |
| **Oracle** | |
| Il compilatore produce le seguenti triple {(10,11,a),(10,15,a),(16,19,a)} e il token finale (19,a) | |

TC\_DF\_6

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_6** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome area | Nodo 9: area=3.14\*(raggio\*raggio) | | Variabile di nome area | Nodo 11: “Area %lf”, area -> | | Variabile di nome area | Nodo 13: area <-double | | Variabile di nome area | Nodo 16: sum=area+89 | | Variabile di nome area | Nodo 19: “Nuova area %d”,area | | Variabile di nome area | Nodo 20: while(i<area) | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(9,11,area),(13,16,area),(13,19,area),(13,20,area)} e il token finale (13,area) | |

TC\_DF\_7

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_7** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome raggio | Nodo 7: raggio <- double | | Variabile di nome raggio | Nodo 8: if(raggio <0) | | Variabile di nome raggio | Nodo 10: area=3.14\*(raggio\*raggio) | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(7,8,raggio),(7,10,raggio)} e il token finale (7,raggio) | |

TC\_DF\_8

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_8** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome  i | Nodo 4: i=1 | | Variabile di nome  i | Nodo 6: “cerchio n. %d raggio:”, i -> | | Variabile di nome  i | Nodo13: i=i+1 | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(4,6,i),(4,13,i)} e il token finale (4,i) | |

TC\_DF\_9

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_DF\_9** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto il control flow graph del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’algoritmo di reaching defintion sul control flow graph (CFG)   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome sum | Nodo 5: sum=0 | | Variabile di nome sum | Nodo 9: sum=sum+x | | Variabile di nome sum | Nodo 11: media =sum+n | | Variabile di nome sum | Nodo 15: sum <- int | | Variabile di nome sum | Nodo 20: if(sum>0) | | |
| **Oracle:** | |
| Il compilatore produce le seguenti triple {(5,9,sum),(9,11,sum),(15,20,sum)} e il token finale (15,sum) | |

3 Testing di regressione

Il testing di regressione è stato effettuato per verificare che le componenti non modificate durante la manutenzione non abbiano subito alterazioni. Queste componenti sono le parti dell’AST che eseguono l’analisi sintattica, lessicale e semantica e la parte di generazione del codice.

3.1 Test case specification per testing di regressione

TC\_AS\_1

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_1** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome  i | Variabile non dichiarata | | Variabile di nome  i | Istruzione linea 10: i=0 | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che la variabile sum non risulta essere dichiarata | |

TC\_AS\_2

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_2** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome raggio | Istruzione linea 1: double raggio | | Variabile di nome raggio | Istruzione linea 8: raggio=true | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che alla variabile raggio è stato assegnato un valore non adeguato al suo tipo di dato | |

TC\_AS\_3

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_3** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome n | Istruzione linea 1: double n | | Variabile di nome n | Istruzione linea 6: n <- bool | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che nell’istruzione read è presente una non uguaglianza di tipi di dati tra la variabile e il convertitore. | |

TC\_AS\_4

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_4** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome flag | Istruzione linea 1: bool flag | | Variabile di nome flag | Istruzione linea 9: flag=true | | Variabile di nome flag | Istruzione linea 12: flag=(i<n) && (i==1) | | |
| **Oracle:** | |
| Il compilatore traduce correttamente il codice YASPL in linguaggio C senza segnalare errori | |

TC\_AS\_5

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_5** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome somma | Istruzione linea 1: def somma (int add1;int add2;): int somma;{ somma = add1+add2;}; | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che non è possibile chiamare una variabile con il nome dato ad una funzione | |

TC\_AS\_6

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_6** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Funzione di nome somma | Istruzione linea 1: def somma (int add1;int add2;): int risultato;{  risultato=add1+add2;  }; | | Funzione di nome somma | Istruzione linea 8: somma(7:n) | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che il numero di parametri inseriti nella chiamata della funzione somma sono di numero inferiore rispetto a quelli definiti nella definizione della funzione | |

TC\_AS\_7

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_7** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome toReturn | Istruzione linea 1: double toReturn;  def somma (int add1;int add2;): int risultato;{  risultato=add1+add2; }; | | Variabile di nome toReturn | Istruzione linea 7: somma(7,8,toReturn) | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che il tipo di dato restituito dalla funzione somma è incompatibile con il tipo di dato della variabile toReturn. | |

TC\_AS\_8

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_8** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome esito | Istruzione linea 1: bool esito; | | Variabile di nome esito | Istruzione linea 9: while(esito==0)do{} | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che c’è un incompatibilità di tipi di dati nell’operazione di relazione dell’istruzione while | |

TC\_AS\_9

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_AS\_9** |
| **Pre-condizione:** | |
| Il compilatore ha prodotto l’abstract syntax tree del programma datogli in input | |
| **Flusso degli eventi** | |
| 1.Il compilatore esegue l’analisi semantica sull’abstract syntax tree.   |  |  | | --- | --- | | **Input** | **Valore** | | Variabile di nome prodotto | Istruzione linea 1: int prodotto; | | Variabile di nome prodotto | Istruzione linea 4: somma(7,8:prodotto) | | |
| **Oracle:** | |
| Il compilatore lancia un eccezione segnalando che non è possibile assegnare il valore della funzione alla variabile prodotto perché la funzione di nome somma non esiste | |

4 Report del testing

Durante le attività di testing il sistema ha risposto correttamente a tutti i casi di test inerenti al data flow. Infatti i cammini calcolati sono risultati stampati correttamente. Anche il testing di regressione è stato soddisfacente perché il sistema risponde correttamente alle vecchie funzionalità e quindi le modifiche non hanno portato errori alle vecchie componenti.