Linguaggi e Traduttori
a.a. 2005/2006
Tesina n° 4

Titolo: Debug C
Descrizione

Si richiede di realizzare un traduttore che serve ad eseguire il debugging di un programma C. Le istruzioni di debugging devono essere inserite direttamente all'interno del codice C. Ogni istruzione di debugging inizia con un etichetta, seguita dal carattere ":", seguita da un numero (che può essere o 0 o 1: tale numero indicherà il tipo di output che dovrà fornire l'istruzione di debugging), seguita dal carattere ":" e seguita dall'istruzione vera e propria, terminata dal carattere ";" :
<etichetta> : <(0| 1)> : <istruzione> ;
Può esserci un ulteriore campo ("s"), opzionale, che indica se, a seguito della stampa di un istruzione di debugging, l'esecuzione del programma debba stopparsi, attendendo la pressione di un tasto da tastiera:
<etichetta> : <(0| 1)> : s : <istruzione> ;
Esistono due forme più compatte:

- *<istruzione>;* : Viene eseguita l'istruzione come un'istruzione di tipo 0
- *s : <istruzione>;* : Viene eseguita l'istruzione come un'istruzione di tipo 0 e con l'opzione "s" attivata

Il programma, lanciato da linea di comando, dovrà fornire le seguenti opzioni:
-o <(0|1)>: Tutte le istruzioni di debugging dovranno fornire un output del tipo specificato
-nd : nel file prodotto vengono commentante tutte le istruzioni di debugging
-d : tutte le istruzioni di debugging vengono attivate
Le opzioni *–nd* e *–d* hanno la possibilità di avere una serie di parametri passati da linea di comando che identificano le etichette dei commenti da commentare e da scommentare:
java <nome_prog> -d a,b,c –nd d,e <file_in> <file_out>
In questo caso verrebbero attivate le istruzioni di debugging etichettate con a, b, c e disattivate le istruzioni di debugging etichettate da d, e. *<file_in>* rappresenta il nome del file di ingresso, *<file_out>* rappresenta il nome del file di uscita, cioè del file C prodotto che conterrà le varie istruzioni di debugging trasformate in C.
Nel caso sia presente solo l'istruzione *–d* seguita da una lista di etichette si intende che tutte le istruzioni di debugging sono disattivati a meno di quelle segnalate nella lista di etichette. Nel caso, invece, sia presente solamente l'istruzione *–nd* seguita da una lista di etichette, si intende che tutte le istruzioni di debugging sono attive eccetto quelle indicate dall'istruzione *–nd*.
Esempi di istruzioni di debugging:
Le istruzioni di debugging servono per stampare a video il contenuto delle variabili:

- Tipo variabile: **int a;**
  Istruzione: **x : 0 : a;**
  output: *a={1}*
- Tipo variabile: **float a[3];**
  Istruzione: **x : 0 : a[1-2];** // Stampa i valore a[1] e a[2]
  output: *a={0.000432, -543,32313}*
- Tipo variabile: **int a[5][10];**
  Istruzione: **x : 0 : a[0-1][0-3];**
  output: *a={{1, 2, 4},{4, 5, 6}}*
- Tipo variabile: **int a;**
  Istruzione: **x : 1 : a;**
  output: *a=1*

- Tipo variabile: **float a[3];**
  Istruzione: **x : 1 : a[1-2];** // Stampa i valore a[1] e a[2]
  output: *a=0.000432 -543,32313*
- Tipo variabile: **int a[5][10];**
  Istruzione: **x : 1 : a[0-1][0-3];**
  output: *a=1 2 4*
  4 5 6
- Tipo variabile: **struct pippo {**
  int a;
  float b;
  int c[2][2];
  int b;**
  }prova;
  Istruzione: **x : 0 : prova;**
  output: prova=*{{3},{1.05},{{3,5},{6,7}},{0F6334}}*
- Tipo variabile: **struct pippo {**
  int a;
  float b;
  int c[2][2];
  }prova;
  Istruzione: **x : 1 : prova;**
  output: prova=*a: 3*
  b: 1.05
  c[0]: 3 5
  c[1]: 6 7
- Tipo variabile: **struct pippo {**
  int a;
  float b;
  int c[2][2];
  }prova;
  Istruzione: **x : 1 : prova.b;**
  output: prova=*b: 1.05*
- Tipo variabile: **struct pippo {**
  int a;
  float b;
  int c[2][2];
  }prova;
  Istruzione: **x : 1 : prova{.b,.c[0][0-1]};**
  output: prova=*a: 3*
  c[0]: 3 5
- Tipo variabile: **int *a;**
  Istruzione: **x : 0 : a;**
  output: *a={0F6578}*
- Tipo variabile: **int *a;**
  Istruzione: **x : 0 : &a;**
  output: *a={1}*
- Tipo variabile: **int *a;**
  Istruzione: **x : 0 : a[0-1];**
  output: *a={1 2}*
- Tipo variabile: **int **a;**
  Istruzione: **x : 0 : a[0][0-1];**

output: *a={1 2}*

- Tipo variabile: **char *a;**
  Istruzione: **x : 0 : &a;**
  output: *a={c}*
- Tipo variabile: **char *a;**
  Istruzione: **x : 0 : a[0-4]**
  output: *a={c, i, a, o}*
- Tipo variabile: **char **a;**
  Istruzione: **x : 0 : a[1][0-5]**
  output: *a={p, r, o, v, a}*
- Tipo variabile: **char a[2][5];**
  Istruzione: **x : 0 : a;**
  output: *a={{c, i, a, o, }, {p, r, o, v, a}}*

Quest'esercizio potrebbe sembrare complesso, in realtà è abbastanza modulare: in primo luogo si può separare la parte delle opzioni in ingresso al programma (e di commento delle istruzioni di debugging) dalla parte di debugging vera e propria. Nel modulo delle opzioni di ingresso basta creare una symbol table in cui si segna per ogni etichetta se deve essere commentata o meno (o se devono essere tutte commentate o scommentate). Nella parte di debugging bisogna analizzare la parte di dichiarazione dei tipi creando delle type-expression e una symbol table per ogni identificatore che contenga tutte le informazioni necessarie alle istruzioni di debugging. Quando viene eseguita l'istruzione di debugging la si interpreta e si possono reperire nella symbol table tutte le informazioni necessarie alla stampa dell'output.
Bisogna ricordarsi solo che lo stesso nome di variabile può essere dichiarato in funzioni precedenti, quindi conviene avere una symbol table in cui si memorizzano le variabili globali e una symbol table in cui si memorizzano le variabili interne alla funzione (tale symbol table dovrà essere svuotata alla fine delle funzione).

Si ricorda anche che la definizione di alcune variabili globali può essere fatta all'interno di un file *.h* richiamato dalla direttiva #include. Questo problema po' essere risolto senza complicazione con jflex utilizzando la tecnica spiegata nella prima esercitazione.

Per la grammatica del c consiglio di cercare in rete quella ufficiale.

Assieme al programma dovrà essere allegata una breve relazione contenente una veloce descrizione delle strutture dati utilizzate, di come il programma funziona e delle specifiche/limiti di funzionamento. Dovranno poi essere allegati dei **file di esempio** per verificare il corretto funzionamento del programma.