Parsing in C e traduzione in Assembler MIPS

Creazione di un compilatore semplificato per C (lessico, sintassi, semantica) usando ANTLR e traduzione semplificata in Assembler MIPS

Tool utile per chi vuole capire, analizzare e tradurre semplici istruzioni C in Assembler MIPS

DOCS COMPLETA DISPONIBILE

INDICE

INTRODUZIONE

L'obiettivo del progetto è generare un compilatore semplificato per il linguaggio C che svolga rispettivamente i seguenti compiti:

1. **Analisi lessicale (lexer):** il compilatore deve analizzare i token appartenenti al linguaggio e riuscire a individuare gli eventuali token errati non appartenenti all'alfabeto della grammatica.

2. **Analisi sintattica (parser):** il compilatore deve analizzare la struttura sintattica del linguaggio e individuare se corrisponde a una sequenza corretta o meno.

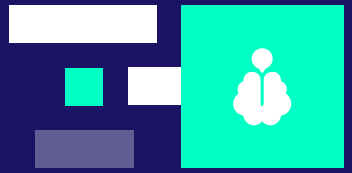
3. **Analisi semantica:** ricavare il significato associato alla struttura sintattica e verificare che le regole di impiego del linguaggio siano soddisfatte.

4. **Gestione degli errori:** riconoscere gli errori e gestire gli stessi indicandoli in maniera esplicativa.

5. **Traduzione:** infine procedere nella traduzione del linguaggio C in un linguaggio di più basso livello quale l'Assembler MIPS (versione semplificata).

Il parser ha prospezione LL(1), quindi con k=1.

TOOLS

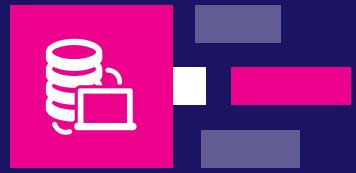


ANTLR

ANother Tool for Language Recognition

GitHub

Controllo di versione

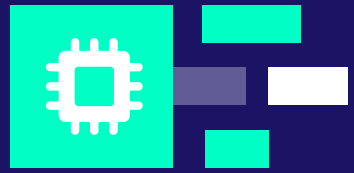


IntelliJ

Java IDE

Assembler MIPS

Linguaggio output della traduzione



ANALISI LESSICALE

Alcuni token riconosciuti

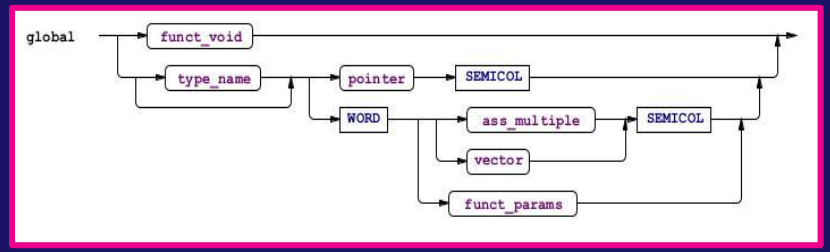
TOKEN	CODICE	TOKEN	CODICE
ELSE	18	RPAREN	44
EQ	19	SEMICOL	45
FLOAT	20	SLASHR	46
FOR	21	SPACE	47
GE	22	SUB	48
GT	23	S_QUOTE	49
HASHTAG	24	TAB	50
IF	25	TOKEN_ERROR	51
INCLUDE	26	UNDRSCR	52
INT	27	VOID	53
WHILE	54	WORD	55
WS	56		

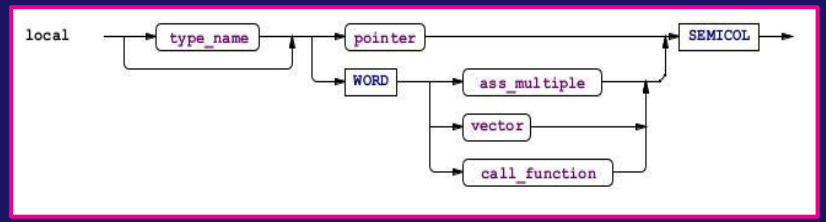
```
---Input---
#include<stdio.h>
#include<stdlib.h>

int a;
float b = 2.2;
char c;
...

---Test ANTLR lexer---
...
Token 13: (10,0)    TokenType:30:    int
Token 14: (10,4)    TokenType:55:    a
Token 15: (10,5)    TokenType:45:    ;
Token 16: (11,0)    TokenType:29:    float
Token 17: (11,6)    TokenType:55:    b
Token 18: (11,8)    TokenType:7:     =
Token 19: (11,10)    TokenType:20:    2.2
Token 20: (11,13)    TokenType:45:    ;
Token 21: (12,0)    TokenType:28:    char
Token 22: (12,5)    TokenType:55:    c
Token 23: (12,6)    TokenType:45:    ;
...
```

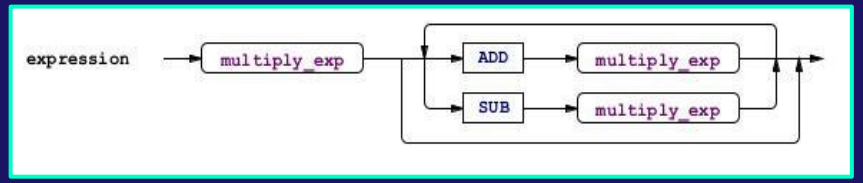
```
---Input---
#include<stdio.h>
#include<stdlib.h>

@
func(){?}
...

---Test ANTLR lexer---
...
Token 13: (4,0)    TokenType:51:    @ #Errore
Token 14: (6,4)    TokenType:55:    funct
Token 15: (6,9)    TokenType:34:    (
Token 16: (6,10)    TokenType:44:    )
Token 17: (6,11)    TokenType:32:    {
Token 18: (7,4)    TokenType:51:    ? #Errore
Token 19: (8,0)    TokenType:42:    }
...
```

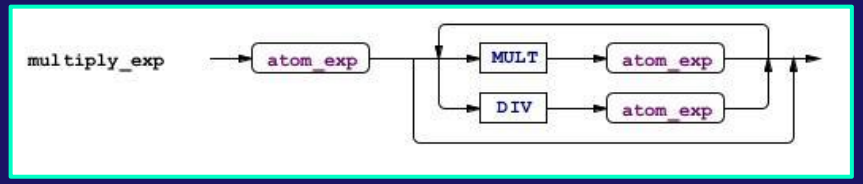
ANALISI SINTATTICA

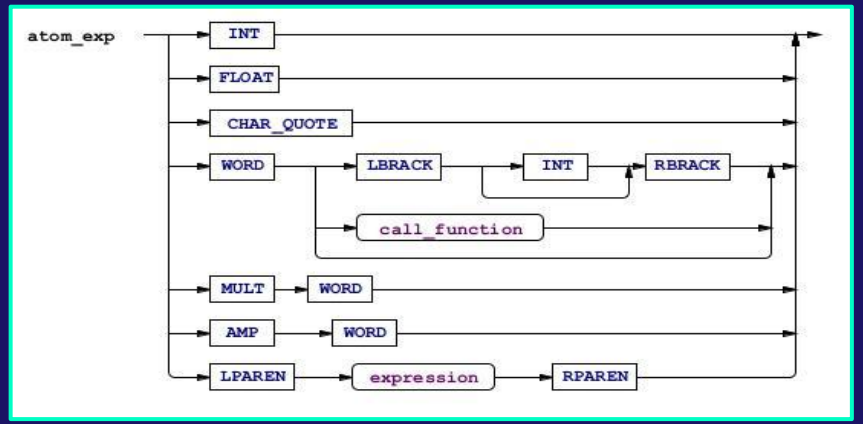
Alcuni diagrammi sintattici











ANALISI SEMANTICA

Symbol table globale

NAME	TYPE	VALUE	ADDRESS	isVAR	VECT
b	int	5	3996	True	X
main	int	null	3992	False	X
c	int	null	3888	True	2,10,7

Symbol table locale

NAME	TYPE	VALUE	ADDRESS	isVAR	VECT
f	int	5	3996	True	X
d	int	null	3888	True	2,10,7

```
#include <stdio.h>
#include <stdlib.h>

int a = 2;
int b = 3;

void funct1() {
        int d = 3;

        if (d == 2) {
        a = 3;
        } else {
        a = 4;
        }
}

void funct2() {
        int e = 3;

        while (e < 5) {
        e += 2;
        }
}
```

```
int main() {
        int c = 4;
        a = 5;

        funct1();

        return 0;
}
```

File di input

ANALISI SEMANTICA

File di output

```
-----------------------------------------
*****    Global Symbol Table    *****
-----------------------------------------
1:    b=Value{name='b', type='int', value='3', address='3996', isVar=true,
isVarPassed=false, isVect=false, vect=[]}
2:    a=Value{name='a', type='int', value='5', address='4000', isVar=true,
isVarPassed=false, isVect=false, vect=[]}
3:    main=Value{name='main', type='int', value='0', address='3976',
isVar=false, isVarPassed=false, isVect=false, vect=[]}
4:    funct2=Value{name='funct2', type='void', value='null', address='3984',
isVar=false, isVarPassed=false, isVect=false, vect=[]}
5:    funct1=Value{name='funct1', type='void', value='null', address='3992',
isVar=false, isVarPassed=false, isVect=false, vect=[]}

-----------------------------------------------
*****    Local Symbol Table: main    *****
-----------------------------------------------
1:    c=Value{name='c', type='int', value='4', address='3972', isVar=true,
isVarPassed=false, isVect=false, vect=[]}
```

```
-----------------------------------------------
*****    Local Symbol Table: funct1    *****
-----------------------------------------------
1:    d=Value{name='d', type='int', value='3', address='3988', isVar=true,
isVarPassed=false, isVect=false, vect=[]}

-----------------------------------------------
*****    Local Symbol Table: funct2    *****
-----------------------------------------------
1:    e=Value{name='e', type='int', value='5', address='3980', isVar=true,
isVarPassed=false, isVect=false, vect=[]}


-------------------------
*****    Debug    *****
-------------------------
Ho dichiarato la variabile 'a' come 'int'
Ho assegnato alla variabile 'a' il valore '2'
Ho dichiarato la variabile 'b' come 'int'
Ho assegnato alla variabile 'b' il valore '3'
Ho dichiarato la funzione 'funct1' con tipo 'void'
Ho dichiarato la variabile locale 'd' come 'int'
...
```

GESTIONE DEGLI ERRORI

Errore sintattico e semantico in contemporanea nella stessa istruzione

File di input

```
int a;
int a = 1 # Doppia dichiarazione e punto virgola mancante
...
```

File di output

```
*****************************************
*****    Parsing completato con 2 errori    *****
*****************************************
```

1: Errore semantico [10] in (13,4) - La variabile <a> è già stata dichiarata
2: Errore sintattico [1] in (14, 0) - Found K_INT ('int') - missing SEMICOL at 'int'

Errore semantico sui tipi

File di input

```
int *p1;
//*p1 = 4; // Commentarla per catturare l'errore alla riga sotto
int e = 5 + *p1;
...
```

File di output

```
*****************************************
*****    Parsing completato con 3 errori    *****
*****************************************
```

1: Errore semantico [12] in (25,13) - La variabile <p1> non è stata inizializzata
2: Errore semantico [13] in (25,10) - L'operazione <+> non può essere eseguita perché almeno uno dei due operandi non ha valore
3: Errore semantico [16] in (25,4) - Valore di tipo non compatibile (null)

TRADUZIONE

Assunzioni:

- Ogni variabile e costante intera (int) occupa 4 Byte, quindi una word nella memoria del MIPS.
- La memoria parte da 0x4000 e va a decrescere man mano che vengono salvate variabili.
- Sono state scritte istruzioni semplificate e simboliche per gestire i valori ritornati ($v0), il return address ($ra)...

File di input

```
#include <stdio.h>
#include <stdlib.h>

int a = 2;
int b = 35;

int main() {
        int c = 4;

        b = (a + b) * c;

        return 0;
}
```

File di output

```
A: .word 2
B: .word 35
MAIN:
   addiu $sp, $sp, -8       #Crea area nello stack pointer
   sw $ra, 4($sp)           #Salva return address
   lw $t1, 4
   sw $t1, 0x3988           #[c=4]
   lw $t2, 0x4000           #[a=2]
   lw $t1, 0x3996           #[b=35]
   add $t2, $t1, $t2
   lw $t1, 0x3988           #[c=4]
   lw $t2, 4
   mul $t1, $t1, $t2
   sw $t1, 0x3996           #[b=148]
   sw 0, 0($sp)             #Memorizza il valore del return
   lw $v0, 0($sp)           #Salva valore d'uscita
   lw $ra, 4($sp)           #Ripristina return address
   addiu $sp, $sp, 8        #Elimina area nello stack pointer
   jr $ra                   #Salta al return address
```

TRADUZIONE

Statements:
In base al comparatore sono state usate una combinazione di istruzioni tra le seguenti:

- beq $t1, $t2, LABEL: se $t1 è uguale a $t2 si salta alla label.
- bne $t1, $t2, LABEL: se $t1 è diverso da $t2 si salta alla label.
- slt $t8, $t1, $t2: se $t2 è minore di $t2 si setta $t8 a 1.

File di input

```
#include <stdio.h>
#include <stdlib.h>

int a = 2;
int b = 3;

int main() {
        int c = 4;

        for (b = 2; b < 5; b += 2) {
        c += 4;
        }
        return 0;
}
```

File di output

```
...
FOR1:
  lw $t2, 2
  sw $t2, 0x3996          #[b=2]
  lw $t1, 0x3996          #[b=2]
  lw $t2, 5
  slt $t8, $t1, $t2
  bne $t8, 1, ENDFOR1
  j ENDINCRFOR1
  INCRFOR1:
     lw $t1, 0x3996       #[b=2]
     lw $t2, 2
     add $t1, $t1, $t2
     sw $t1, 0x3996       #[b=4]
     j FOR1
  ENDINCRFOR1:
  lw $t2, 0x3988          #[c=4]
  lw $t1, 4
  add $t2, $t1, $t2
  sw $t2, 0x3988          #[c=8]
  j INCRFOR1
ENDFOR1:
...
```

Altri statements e chiamate di funzioni nella docs...

GRAZIE

Samuele Ferri [1045975]
Simone Sudati [1045936]