Algoritmo di Parsing Top-Down Predittivo – Istruzioni

Il programma prende come una grammatica LL(1) e una parola da verificare ed esegue il predictive top-down parsing

Premesse sull'input

Il file di input deve essere scritto usando le seguenti regole:

- Nella prima riga deve essere presente il numero totale delle produzioni
- Nella seconda riga deve essere presente la parola di cui si vuole effettuare il parsing
- Dalla terza riga in poi devono essere presenti le produzioni nella forma $A \rightarrow \alpha$ dove α può essere una qualsiasi sequenza di caratteri (non devono essere presenti spazi).
- I simboli devono essere espressi da al massimo un carattere, epsilon deve essere rappresentato con il carattere "#"
- Il driver di una produzione e il suo body devono essere separati dal simbolo ">", più produzioni aventi lo stesso driver possono essere raggruppate dividendole con il simbolo "|"
- La grammatica può essere ambigua, ma non left recursive

Riporto la grammatica d'esempio che viene fornita

```
8
d+d*d
E>TA
A>+TA|#
T>FB
B>*FB|#
F>(E)|d
```

Gli input dati come esempio sono i seguenti:

- Input1: grammatica vista a lezione, con parola appartenente al linguaggio
- Input2: stessa grammatica di input1, la parola non appartiene al linguaggio
- Input3: grammatica ambigua vista a lezione, viene dato errore nella creazione della tabella di parsing
- Input4: grammatica trovata su internet, con parola appartenente al linguaggio
- Input5: stessa grammatica di Input4, la parola non appartiene al linguaggio
- Input6: grammatica trovata su internet, con parola appartenente al linguaggio

Premesse sull'output

Nel file di output vengono elencate tutti i procedimenti svolti dal programma:

- Viene riportata la grammatica di partenza estrapolata dal file di input
- Viene riportata la parola di cui fare il parsing
- Vengono listati i First di ogni non terminale
- Vengono listati i Follow di ogni non terminale
- Vengono listate le caselle della tabella di parsing che sono state rimpite
- Viene riportato passo per passo il procedimento per il parsing predittivo
- Alla fine viene riportato il risultato dell'esecuzione

Al momento della consegna ogni output corrisponde ad ogni input con lo stesso numero, tuttavia è possibile creare output con nomi diversi o sovrascrivere quelli già creati. Per questo motivo all'inizio di ogni file di output viene scritto il nome del file di input che lo ha generato

Compilazione

Per compilare il programma eseguire dalla directory del progetto il comando:

```
1  make
```

Dove i file c usati sono rispettivamente:

- lf.c: file contenente tutte le funzioni relative agli algoritmi di parsing

- structures.c: file contenente tutte le funzioni di supporto all'utilizzo delle strutture dati impiegate

Nel caso non fosse possibile usare make si può usare in alternativa il comando:

```
1  gcc lf.c structures.c -o PredictiveParsing
```

Esecuzione

Una volta compilato il programma è possibile eseguirlo con il comando

```
1  ./PredictiveParsing [-i FILENAME][-o FILENAME]
```

Dove FILENAME è il nome di un file che si vuole usare, sono presenti nello Zip del progetto 6 esempi di input e i rispettivi output.
Per eseguire un file di input personalizzato aggiungerlo alla directory 209084/inputs e eseguire

```
1  ./PredictiveParsing -i custominput
```

Per salvare l'output di questo file in un file personalizzato usare il comando

```
1  ./PredictiveParsing -i custominput -o customoutput
```

Sarà possibile trovare l'output nella directory 209084/outputs.
Se i parametri non vengono specificati verranno usati input1.txt e output1.txt come default.

Premesse sulle strutture dati

Symbol

Struct usata per rappresentare tutti i simboli della grammatica

```
struct symbol{
    char letter; //lettera per identificare il simbolo
    int terminal; // terminal=1 true, terminal=0 false
    struct symbol** firsts; //set dei first
    struct symbol** follow; //set dei follow
    struct symbol** tofollow; //set dei simboli da cui copiare i follow
};
```

Production

Struct usata per rappresentare le produzioni della grammatica

```
struct production{
    struct symbol* sym; //driver
    struct symbol** body; //body
    int size; //lunghezza del body
};
```

Record

Struct usata per identificare una entry della tabella di parsing

```
struct record{
    struct symbol* terminal; //simbolo terminale
    struct symbol* nonterminal; //simbolo non terminale
    struct production* prod; //produzione associata alla cella della
};                           //tabella
```

Stack

Struct usata per implementare uno stack per l'algoritmo di parsing predittivo

```
struct stack{
    int size; //numero di simboli massimi nello stack
    int head; //indice per la testa della pila
    struct symbol** pila; //array di simboli
};
```

Set

Il set è stato implementato come puntatore a puntatore di simboli. Il set viene inizializzato come un vettore di 93 elementi (i caratteri ASCII stampabili), in modo da ottenere un hashset con distribuzione delle chiavi uniforme. Grazie a questa scelta le funzioni che riguardano i set hanno le seguenti complessità:

- Creazione: O(1)
- Aggiunta di un simbolo: O(1)
- Ricerca di un determinato simbolo: O(1)
- Merge di due set: O(n)
- Rimozione di un determinato simbolo: O(1)
- Print: O(n)
- Equals tra due set: O(n)
- Copia di un set: O(n)