Relazione COMPILATORi

Cingillo Claudio & Alduina Alessio

Introduzione:

Il presente rapporto descrive in dettaglio un progetto di traduttore guidato dalla sintassi in grado di riconoscere un linguaggio che descrive le prenotazioni da parte di varie agenzie di diverse camere di hotel.

Strumenti Utilizzati:

Bison:

Bison è un generatore di parser, usato per la creazione di analizzatori sintattici. Bison analizza la struttura sintattica del codice sorgente, fondamentalmente seguendo le specifiche della grammatica del linguaggio target.  
La sezione iniziale del codice contiene delle direttive di pre-processore e l'inclusione di alcune librerie necessarie per il corretto funzionamento del programma.  
Nel nostro caso, il file contiene diverse sezioni:

1. **Definizione delle intestazioni e inclusioni**: La sezione iniziale tra %{ e %} è utilizzata per includere librerie, definire variabili globali e altre istruzioni che devono essere direttamente incluse nel codice generato da Bison. In questo caso, vengono inclusi header standard come <stdio.h>, <stdlib.h>, e anche il file di intestazione "SymbolTable.h".
2. **Definizione di union**: La sezione %union definisce i tipi di dati che possono essere restituiti dai token del lexer. In questo caso, viene definita un'unione contenente un puntatore a stringa (char\*), un intero (int), e un array di interi (int \*).
3. **Definizione dei token**: La sezione %token definisce i token che il lexer può generare. I token sono suddivisi in categorie specifiche di dati (<string>, <intero>, <int\_vec>) utilizzando l'unione definita in precedenza. Ogni token ha un tipo associato.
4. **Definizione della grammatica**: La sezione che inizia con %start specifica il simbolo di partenza della grammatica, che nel nostro caso è INIZIO. Ogni produzione è composta da terminali e non terminali
5. **Implementazione delle Actions**: Il codice C all'interno delle parentesi graffe {} all'interno delle produzioni della grammatica esegue azioni specifiche quando una produzione viene riconosciuta. Ad esempio, ci sono chiamate a funzioni esterne come aggiungiStanza e aggiungiAgenzia.
6. **Funzione yyerror**: Questa funzione viene chiamata in caso di errore durante l'analisi sintattica e stampa un messaggio di errore.
7. **Funzione main**: La funzione main apre un file di input, imposta lo standard input per il lexer (yyin), inizializza alcune variabili (come i, y, e z), chiama la funzione init() (presumibilmente per inizializzare qualcosa), esegue l'analisi sintattica con yyparse(), e stampa alcune informazioni in caso di successo.

Flex:

Flex permette di definire delle regole che specificano i token che verranno riconosciuti nella fase di analisi lessicale. Queste regole possono essere espresse sia con espressioni regolari che con codice C. Flex genera poi un programma in linguaggio C che implementa l'analizzatore lessicale basandosi sulle regole definite.  
Nel nostro caso il file contiene diverse sezioni:

1. **Intestazione e Inclusioni**: La sezione iniziale del file FLEX, tra %{ e %}, contiene le inclusioni delle librerie standard necessarie e la dichiarazione dell'header del parser generato da Bison (parser.tab.h).
2. **Definizione di Espressioni Regolari**:
   * Sono definite diverse espressioni regolari come cifre, mese, anno, DATA, numeri, caratteri, CARATTERI, ecc., utilizzate per riconoscere parti specifiche del testo di input.
   * DATA riconosce il formato MM-AAAA, e numeri riconosce una sequenza di cifre.
   * SEP1 e SEP2 sono utilizzati per riconoscere le delimitazioni tra le sezioni del file di input.
   * STANZA riconosce una tipologia di stanza con eventuali spazi.
   * AGENZIA riconosce il nome del tour operator tra virgolette.
3. **Regole di Riconoscimento**:
   * Le regole di riconoscimento seguono il formato <espressione> { azione; return token; }.
   * Ad esempio, {SEP1} riconosce la sequenza "&&&" e restituisce il token sep1.
   * {DATA} riconosce una data e converte i componenti in un array di interi, restituito come DATA.

Struttura del Backend

La struttura del Backend è costituita da due file, un header file .h dove vengono inizializzate le funzioni e le strutture dove andranno i dati, e un file .c dove viene creato il corpo delle funzioni

File Header:

1. **Struttura date Stanza:**
   * Contiene le informazioni su una stanza, rappresentata da un tipo di stanza e il suo prezzo.
2. **Struttura dati Prenotazione:**
   * Contiene le informazioni relative a una prenotazione, come il tipo di stanza e il numero di stanze prenotate.
3. **Struttura dati Agenzia:**
   * Rappresenta un'agenzia turistica con un nome, un codice, il numero di partecipanti, e il periodo della prenotazione.
   * Contiene un array di puntatori a strutture prenotazione per gestire le prenotazioni associate all'agenzia.

Backend:

Il vero core dell’analizzatore lessicale è scritto in C, un linguaggio sia di alto livello che di basso livello e in questo caso, nel nostro file avviene tutta la gestione delle aggiunzioni delle stanze, delle prenotazioni e delle agenzie inizializzate come array e allocati dinamicamente nell’ heap.

Inoltre viene gestito anche il prezzo totale e il prezzo scontato nel caso i partecipanti siano compresi tra 30 e 50, in questo caso lo sconto applicato sarà del 15%, nel caso i partecipanti siano maggiori di 50, lo sconto applicato al prezzo totale sarà del 25%.

Infine avverrà la stampa finale con tutti i prezzi per agenzia e successivamente una stampa riguardante il guadagno totale dell’hotel.

Output:

Una volta compilato tutto il codice sorgente e messo in atto il compilatore, esso potrà essere eseguito tramite un terminale con un file in input dove saranno presenti tutte le specifiche, ma per nostra sfortuna, il programma termina senza dare in output quello che ci aspettavamo, nonostante non dia problemi di alcun tipo