**Relazione Progetto Compilatori 20/21**

Gruppo: Nicola Levorato, Marco Alessio

La sintassi del linguaggio progettato è molto simile a quella del linguaggio C con alcune differenze:

1. La funzione per stampare in output stringhe o variabili è “print” anziché “printf”;
2. Gli operatori logici sono rappresentati tramite i token “and”, “or”, “not” e “xor” anziché “&&”, “||” e “!”;
3. I valori booleani sono rappresentati tramite i token “true” e “false” anziché 1 o 0;
4. Le stringhe sono comprese anche tra apici (‘’) oltre che le virgolette (“”).

Nell’implementazione dell’analizzatore lessicale abbiamo creato la macro “PROJECT\_LOGGING” con la direttiva #define insieme alla funzione “LOG\_FL” per determinare se abilitare la stampa del logging in output per debuggare più facilmente il codice o per disabilitarla in modo da ottenere in output solamente il three address code generato dal compilatore.

Tra i vari token riconosciuti abbiamo incluso le espressioni regolari per le due tipologie di commenti (// e /\* \*/) che vengono ignorati dall’analizzatore lessicale e le due tipologie di stringhe delimitate dagli apici (‘’) o dalle virgolette (“”).

Inoltre abbiamo incluso gli operatori di assegnazione come +=, -=, \*= e /=, gli operatori di incremento e decremento (++ e --) e gli operatori di shift sinistro, destro e complemento unario (<<, >> e ~).

Nell’implementazione del compilatore abbiamo utilizzato la tabella dei simboli, implementata tramite uthash.h, per gestire le variabili dichiarate, il loro tipo e il contenuto della loro assegnazione.

Inoltre come per l’analizzatore lessicale abbiamo usato la macro “PROJECT\_LOGGING” insieme alla funzione “LOG\_Y” per abilitare o disabilitare la stampa del logging in output.

Il tipo associato ai token e ai simboli non terminali è una struct chiamata “address” che contiene diversi campi. In questo modo in base al token/simbolo utilizzato posso assegnare un valore solo ai campi interessati e allo stesso tempo dichiarare tutti i token/simboli dello stesso tipo. Questa soluzione semplifica di molto la gestione dei tipi.

Per gestire la dichiarazione delle variabili e l’assegnazione del tipo corrispondente abbiamo utilizzato la tecnica dello stack descritta in classe anche se in realtà l’unico tipo disponibile nel nostro linguaggio è il tipo intero. Questo metodo però permette di semplificare la gestione di più tipi nel caso si volesse estendere il linguaggio e renderlo più complesso.

Inoltre, in uno statement di dichiarazione è possibile dichiarare più variabili e assegnarle un valore di inizializzazione.

Il passaggio degli attributi ereditati è gestito basandosi su un trucco. Invece di creare l’etichetta next prima di ogni statement questa viene creata direttamente all’inizio di ogni statement che ne ha bisogno come ad esempio l’if, l’if-else e il while. In questo modo si evita di creare etichette inutilmente come per gli statement di dichiarazione o assegnazione di variabile. Poi alla fine di ogni statement viene stampata l’etichetta next creata inizialmente, invece di stamparla una volta aver ridotto lo statement.

Il difetto di questa tecnica è che si creano dei salti incondizionati aggiuntivi che potrebbero essere evitati ma, nonostante ciò, il 3AC ottenuto è corretto.

Inoltre, tutti gli altri attributi ereditati come true\_label, false\_label e begin\_label insieme a next\_label vengono creati tramite delle mid-rule action o dei non terminali fittizi che eseguono un’azione. La stampa di queste etichette avviene sempre usando dei non terminali fittizi e sfruttando la tecnica dello stack vista in classe.

Le maggiori difficoltà riscontrate durante lo sviluppo del compilatore sono state:

1. la gestione degli attributi ereditati: inizialmente abbiamo provato a implementarli usando la SDD presente nelle slide viste a lezione ma abbiamo riscontrato difficoltà riguardo la gestione dello stack;
2. ambiguità dovuta al dangling-else: a causa del fatto che le due istruzioni if e if-else condividono la stessa parte iniziale, se si usano mid-rule action differenti per implementare gli attributi ereditati che sono diversi nei due casi (false\_label nel primo caso coincide con next mentre nel secondo caso viene creato) si verificano dei conflitti reduce/reduce dato che i token sono gli stessi ma bison non sa che azione eseguire. Per risolvere questo problema abbiamo usato un trucco: la prima parte dell’if è la stessa per entrambi i casi quindi next\_label coincide con false\_label, dopo aver processato il token “ELSE” devo assegnare a next\_label una nuova label. Questo è possibile grazie al fatto che nell’if-else la label next viene stampata solo dopo aver processato il token “ELSE”.