Progetto di Linguaggi di Programmazione

Anno accademico 2023-2024

Versione n. 03 del 10 Gennaio 2024

In questo documento trovate le informazioni necessarie sul progetto. Vi ricordo che il progetto deve essere fatto da gruppi di 3-4 persone. Ogni gruppo ha una o un *responsabile* che è anche colei/colui che deve comunicarmi, via email, i nomi dei componenti del gruppo. Il termine per la formazione dei gruppi sarà il **10 gennaio 2024**. I gruppi costituiti successivamente non hanno diritto ad alcun bonus e lo stesso vale per quei gruppi in cui la composizione viene modificata. Per chiarimenti scrivetemi.

Il presente documento può essere modificato. Controllate quindi regolarmente il documento. Le modifiche verranno comunque segnalate (sul canale Discord). Per accedere al documento (visualizzazione) seguite il link:

https://docs.google.com/document/d/1EDOtqgRIoyS3aM5VRlUq5LH9il2A-MtnZHAV1xCHxCo/edit?usp=sharing

Consegna del progetto

Il o la responsabile deve creare un repository privato su GitHub, e invitare come collaboratori tutti i componenti del gruppo, i docenti (**gmpinna** e **bitbart**) ed il tutor (**dalps**). Per poter avere il bonus il progetto va consegnato entro le 23:59 del **15 febbraio 2024**.

Il repository deve contenere tutti i file sorgenti del vostro progetto, generato con

```
dune init proj lip23
```

Il progetto deve essere compilabile con dune build senza generare errori o warning.

Inoltre, il progetto deve usare i files presenti nel repository:

https://github.com/informatica-unica/lip/tree/main/imp/lip23

Questi files *non* possono essere modificati, ad eccezione del file test/tests.

Discussioni e chiarimenti

Domande e richieste di chiarimenti o spiegazioni vanno inoltrate via email possibilmente dal(la) responsabile del gruppo. Esiste un canale su Discord dedicato a domande, dubbi e discussioni sul progetto.

Descrizione del progetto

Lo scopo del progetto è implementare CROBOTS (crobots.deepthought.it/home.php), uno scontro tra robots programmati in un sottoinsieme del linguaggio C. Di seguito trovate la descrizione del linguaggio e alcuni esempi. Dovete scrivere il lexer, il parser e l'interprete small step. Non dovete invece preoccuparvi della logica del gioco, né della parte grafica.

Il linguaggio

Illustriamo la grammatica del linguaggio dei programmi di CROBOTS nella forma BNF. Nelle regole seguenti, una stringa racchiusa da doppie virgolette rappresenta un simbolo terminale; tutti gli altri simboli sono non-terminali. Inoltre:

- un simbolo seguito da ? è un opzione
- un simbolo seguito da * è la sequenza di zero o più occorrenze del simbolo
- un simbolo seguito da + è la sequenza di una o più occorrenze del simbolo.

Un programma di CROBOTS è una lista di dichiarazioni di variabili globali, funzioni e procedure.

Una dichiarazione di variabile globale è formata dalla stringa "int" seguita da una lista di identificatori non vuota e un punto e virgola. Ogni identificatore presente nella lista può essere opzionalmente seguito da un "=" e un'espressione di inizializzazione; gli elementi della lista sono separati da una virgola.

declaration ::= "int" init_declarator_list ";"

init_declarator_list ::=

| init_declarator

| init_declarator "," init-declarator_list

init-declarator ::= identifier ("=" expression)?

Una definizione di funzione o procedura è formata da un identificatore seguito una lista di parametri e

infine un blocco sintattico contenente il corpo della funzione. Gli elementi della lista di parametri sono identificatori; ogni parametro è separato dal successivo con una virgola. La lista di parametri è racchiusa tra parentesi tonde.

function_definition ::= identifier "(" parameter_list ")" compound_statement

Un blocco sintattico è una lista (possibilmente vuota) di dichiarazioni di variabili locali, seguita da una lista (possibilmente vuota) di statement. L'intero blocco è racchiuso tra parentesi graffe. Le dichiarazioni locali hanno la stessa forma delle dichiarazioni globali.

compound_statement ::= "{" declaration* statement* "}"

Uno statement può essere uno tra: il null statement “;”, un'espressione seguita da un punto e virgola, un'istruzione condizionale, un'istruzione iterativa, un'istruzione di salto o un blocco sintattico.

statement ::=

| “;”

| expression “;”

| conditional_stat

| iterative_stat

| jump_stat

| compound_stat

Un'istruzione condizionale è formata dalla keyword "if" seguita da un'espressione racchiusa tra parentesi, uno statement, e dopo, opzionalmente, la keyword "else" seguita da un secondo statement.

conditional_stat ::=

| "if" "(" expression ")" statement

| "if" "(" expression ")" statement "else" statement"

Il costrutto if con ramo else ha priorità sul primo.

Un'istruzione iterativa è formata dalla keyword “while” seguita da un'espressione tra parentesi e uno

statement. Lo statement può essere preposto (ed eseguito in anticipo) al controllo dell’espressione tramite la seconda forma.

iterative_stat ::=

| "while" "(" expression ")" statement

| “do” statement “while” “(“ expression “)” “;”

In assenza di blocchi sintattici, gli statement condizionali e iterativi hanno priorità rispetto alle sequenze di statement.

Un'istruzione di salto è formata dalla keyword "return" seguita da un'espressione opzionale e un punto e virgola.

jump_stat ::= "return" expression? ";"

Si considerano espressioni gli assegnamenti, le espressioni binarie, le espressioni unarie e le espressioni primarie: quest’ultime sono le costanti numeriche, le chiamate di funzione/procedura e gli identificatori.

Un'espressione può essere un'espressione binaria oppure un identificatore seguito dal simbolo "=" e da un'altra espressione.

expression ::=

| binary_expr

| identifier "=" expr

Un'espressione binaria è un'espressione unaria oppure la combinazione di due espressioni binarie e un operatore infisso.

binary_expr ::=

| unary_expr

| binary_expr binary_op binary_expr

La seguente grammatica descrive gli operatori binari disponibili:

binary_op ::= "*" | "/" | "%" | "+" | "-" | "<" | ">" | "<=" | "=>" | "==" | "!=" | "&&" | "||"

Un'espressione unaria è un'espressione primaria oppure la combinazione di un operatore unario seguito da un'espressione unaria.

unary_expr ::=

| primary_expr

| unary_op unary_expr

L'unico operatore unario è quello di negazione di un numero.

unary_op ::= "-"

Un'espressione primaria è data da un atomo:

primary_expr ::=

| identifier

| constant

| identifier "(" argument_list ")"

| "(" expression ")"

Nella chiamata di funzione/procedura, la lista di argomenti è una lista (possibilmente vuota) di espressioni separate da una virgola.

Le priorità degli operatori binari e unari sono le stesse del linguaggio C. In particolare, gli operatori unari hanno precedenza sugli operatori binari; le operazioni aritmetiche hanno precedenza sulle operazioni di confronto "==", "!=", "<", "<=", ">" ">=", le quali hanno a loro volta precedenza sulle operazioni booleane.

La sintassi concreta del linguaggio comprende anche commenti su singola linea, aperti dalla stringa “//” e commenti multilinea, racchiusi tra “/*” e “*/”. I commenti non hanno alcun effetto su l'albero di sintassi astratta di un programma.

Un identificatore è una stringa che inizia con una lettera minuscola, dopodiché può contenere lettere maiuscole e minuscole, cifre e underscore.

Una costante numerica è una qualsiasi sequenza di cifre decimali che non inizi con uno zero.

Un esempio di programma è

```
main()
    {
        int range, x;
        while(1)
        {
            if (scan(x,10)!=0)
            {
                x = x + 5-(scan(x-5,5) != 0)*10;
                x = x + 3-(scan(x-3,3) != 0)*6;
                if ((range=scan(x,10))>20)
                {
                    cannon(x,range);
                    if (range>200)
                        drive(x,50);
                    else
                        drive(180-x,0);
                }
                x = x - 20;
            }
            else
                x = x + 20;
        }
    }
```

Non vi è un limite alla lunghezza di un programma.

Semantica small step

L’entry point di un programma è la routine main senza argomenti. L’interprete rifiuta quei programmi che non definiscono la funzione main, o che definiscono main ma con uno o più parametri.

Tutte le dichiarazioni globali presenti nel programma sono visibili all’interno di main al momento della sua esecuzione.

La memoria dell’interprete simula un array di interi di lunghezza arbitraria. Quando un programma dichiara una nuova variabile, l’interprete modifica l’ambiente locale della funzione con una nuova associazione del nome della variabile a una locazione di memoria disponibile, e assegna alla locazione un valore di default arbitrario in assenza di un inizializzatore.

Gli argomenti alle funzioni o procedure sono passati per valore. Le espressioni passate come argomento devono essere riscritte a delle costanti intere. La funzione non può essere invocata finché tutti gli argomenti non sono stati ridotti a delle costanti intere. Se un argomento è un’espressione diversa da una costante, verrà riscritto in un passo ulteriore e l’invocazione della funzione è rimandata a un momento successivo.

L’interprete fallisce se il numero di argomenti passati alla funzione non coincide con il numero di parametri elencati nella definizione della funzione.

L’ambiente iniziale dell’interprete definisce le primitive scan, drive, cannon, damage, loc_x, loc_y, rand, sqrt, sin, cos, tan, atan e le lega alle loro implementazioni, rese disponibili dall’interfaccia robot.mli. **Non dovete modificare l’implementazione delle primitive**.

L’invocazione di una primitiva richiede un solo passo di semantica e segue le stesse regole dell’invocazione di funzioni definite dall’utente per quanto riguarda il passaggio degli argomenti.

Se un programma tenta di ridefinire una delle funzioni intrinseche, l’interprete solleva l’eccezione IntrinsicOverride.

Se un programma tenta di accedere a una variabile x che non è stata dichiarata, l’interprete solleva l’eccezione UndeclaredVariable x che riporta il nome della variabile errata.

La valutazione di un’espressione può rendere un numero intero oppure nulla. Un’espressione che

non rende nulla è data, per esempio, dalla chiamata di una procedura. L’interprete diverge se un programma tenta di applicare un operatore su interi all’espressione nulla.

Per valutare le espressioni booleane si adotta la seguente convenzione: un intero diverso da zero rappresenta il booleano true e zero rappresenta il booleano false.

Le regole della semantica non vengono riportate dato che sono quelle viste a lezione ed in laboratorio. Per dubbi chiedete.