**Interpreter języka Macchiato**

**Struktura programu**

Rozwiązanie skała się z plików wygenerowanych przez bnfc:

LexMacchiato.hs ParMacchiato.hs SkelMacchiato.hs PrintMacchiato.hs AbsMacchiato.hs ErrM.hs

Oraz z plików:

Main.hs Interpreter.hs DataStructures.hs InbuildFunctions.hs Utils.hs StaticTypeControl.hs

W pliku Main.hs znajduje się funkcja główna programu. Interpreter może wykonywać pojedynczy plik podany jako argument z linii poleceń jak również może wykonywać kod otrzymany ze standardowego wejścia (wymaga na końcu wpisania znaku EOF):

$ ./interpreter program.mc

$ ./interpreter

Interpreter.hs - Właściwy kod interpretera

StaticTypeControl.hs - statyczna kontrola typów

DataStructures.hs - definicje struktur danych

InbuildFunctions.hs - definicje funkcji wbudowanych w język

Utils.hs - różne funkcje pomocnicze wykorzystywane przez interpreter

**Interpreter**

Interpreter wykonywany jest za pomocą transformera StateT z monadą IO. Stanem monady StateT jest typ Scope przetrzymujący informacje o stanie programu. Scope zawiera:

* Środowisko zmiennych zadeklarowanych w aktualnym bloku
* Środowisko zmiennych zadeklarowanych na zewnątrz aktualnego bloku
* Środowisko funkcji zadeklarowanych w aktualnym bloku
* Środowisko funkcji zadeklarowanych na zewnątrz aktualnego bloku
* Skład
* Informację czy nastąpił return z funkcji oraz jaka była jego wartość

Mapowanie nazw zmiennych w wartości odbywa się za pośrednictwem lokacji. Natomiast nazwy funkcji mapowane są bezpośrednio w strukturę przetrzymującą definicję funkcji. Dzięki takiemu rozwiązaniu możliwe jest przysłanianie zmiennych zdefiniowanych w zewnętrznym bloku, ale nie funkcji.

Wyrażenia arytmetyczne w języku nie mają efektów ubocznych, zatem Wykonywanie ich nie modyfikuje Scope, Stąd też operacja usunięcia elementu z mapy, która pierwotnie była wyrażenie w moim języku i miała zwracać wartość usuwanego elementu, została zmieniona na instrukcję. Semantycznie operacja ta usuwa element z mapy lub nic nie robi jeśli element o zadanym kluczu nie należy do mapy.

W języku znalazły się cztery wbudowane funkcje: print, intToStr, strToInt, concatStr.

Wykonanie programu zaczyna się od funkcji ‘main’, która musi być typu int oraz nie może mieć żadnych argumentów. Zwrócona wartość 0 oznacza poprawne zakończenie się programu, zwrócenie przez ‘main’ innej wartości niż 0 zakończy się błędem.

Błędy w tej fazie przerywają natychmiastowo wykonywanie programu i wypisują krótki komunikat diagnostyczny.

**Statyczna kontrola typów**

Macchiato jest językiem silnie typowanym. Wykonanie programu poprzedza kontrola typów. Sprawdzanie typowania odbywa się za pomocą monady State, która jako stan przetrzymuje środowisko mapujące nazwy zdefiniowanych zmiennych w ich typy.

Typy istniejące w języku to:

* Int
* String
* Boolean
* Tuple
* Array
* Map
* Void
* Fun

Wyłącznie int, string, boolean, tuple są typami porównywalnymi i niezmiennymi, mogą służyć jako klucze dla typy map oraz należeć do krotek.

Krotki mogą składać się z elementów różnych typów, jak również mogą być zagnieżdżone. Indeksować je można wyłącznie literałami typu int, nie jest możliwe odwołane się do któregoś elementu krotki za pomocą wyrażenia arytmetycznego. Jest to problem na jaki natrafiłem pisząc statyczne typowanie. Sprawdzając typ wyrażenia będącego odwołaniem do danego elementu krotki za pomocą wyrażenia arytmetycznego musi być znany indeks elementu krotki, do którego odwołanie następuje. Nie jest to możliwe na etapie statycznej kontroli typów, ponieważ wartość tego wyrażenia arytmetycznego zostanie obliczona dopiero w fazie dynamicznej podczas interpretowania programu.

W przypadku wystąpienia błędu na tym etapie program jest przerywany, na wyjście wypisywane poddrzewo składni abstrakcyjnej, którym wystąpił błąd z ewentualnym dodatkowym opisem.

**Gramatyka**

W gramatyce języka zostały wprowadzone drobne zmiany: operacja usunięcia mapy została przeniesiona ze wyrażeń do instrukcji, a indeksy przy odwoływaniu do elementów krotek zostały ograniczone do literałów typu int.

-- programs ------------------------------------------------

entrypoints Program ;

Program. Program ::= [TopDef] ;

FnDef. TopDef ::= Type Ident "(" [Arg] ")" Block ;

separator nonempty TopDef "" ;

Arg. Arg ::= Type Ident;

separator Arg "," ;

-- statements ----------------------------------------------

Block. Block ::= "{" [Stmt] "}" ;

separator Stmt "" ;

Empty. Stmt ::= ";" ;

BStmt. Stmt ::= Block ;

FunLoc. Stmt ::= Type Ident "(" [Arg] ")" Stmt ;

Decl. Stmt ::= Type [Item] ";" ;

NoInit. Item ::= Ident ;

Init. Item ::= Ident "=" Expr ;

separator nonempty Item "," ;

Ass. Stmt ::= Ident "=" Expr ";" ;

ArrAss. Stmt ::= Ident "[" Expr "]" "=" Expr ";" ;

MapAss. Stmt ::= Ident "[[" Expr "]]" "=" Expr ";" ;

MapDel. Stmt ::= "del" Ident "[[" Expr "]]" ;

Incr. Stmt ::= Ident "++" ";" ;

Decr. Stmt ::= Ident "--" ";" ;

Ret. Stmt ::= "return" Expr ";" ;

VRet. Stmt ::= "return" ";" ;

Cond. Stmt ::= "if" "(" Expr ")" Stmt ;

CondElse. Stmt ::= "if" "(" Expr ")" Stmt "else" Stmt ;

While. Stmt ::= "while" "(" Expr ")" Stmt ;

ForUp. Stmt ::= "for" Ident "=" Expr "to" Expr "do" Stmt;

ForDown. Stmt ::= "for" Ident "=" Expr "downto" Expr "do" Stmt;

SExp. Stmt ::= Expr ";" ;

-- Types ---------------------------------------------------

Int. Type ::= "int" ;

Str. Type ::= "string" ;

Bool. Type ::= "boolean" ;

Void. Type ::= "void" ;

Tup. Type ::= "<" [Type] ">" ;

Arr. Type ::= Type "[]" ;

Map. Type ::= Type "[[" Type "]]" ;

internal Fun. Type ::= Type "(" [Type] ")" ;

separator Type "," ;

-- Expressions ---------------------------------------------

EVar. Expr6 ::= Ident ;

ELitInt. Expr6 ::= Integer ;

ELitTrue. Expr6 ::= "true" ;

ELitFalse. Expr6 ::= "false" ;

EApp. Expr6 ::= Ident "(" [Expr] ")" ;

EString. Expr6 ::= String ;

ENewTup. Expr6 ::= "<(" [Expr] ")>" ;

EAccTup. Expr6 ::= Ident "<(" Integer ")>" ;

ENewArr. Expr6 ::= "new" Type "[" Expr "]" ;

EAccArr. Expr6 ::= Ident "[" Expr "]" ;

ENewMap. Expr6 ::= "new" Type "[[" Type "]]" ;

EAccMap. Expr6 ::= Ident "[[" Expr "]]" ;

EHasMap. Expr6 ::= "has" Ident "[[" Expr "]]" ;

Neg. Expr5 ::= "-" Expr6 ;

Not. Expr5 ::= "!" Expr6 ;

EMul. Expr4 ::= Expr4 MulOp Expr5 ;

EAdd. Expr3 ::= Expr3 AddOp Expr4 ;

ERel. Expr2 ::= Expr2 RelOp Expr3 ;

EAnd. Expr1 ::= Expr2 "&&" Expr1 ;

EOr. Expr ::= Expr1 "||" Expr ;

coercions Expr 6 ;

separator Expr "," ;

-- operators -----------------------------------------------

Plus. AddOp ::= "+" ;

Minus. AddOp ::= "-" ;

Times. MulOp ::= "\*" ;

Div. MulOp ::= "/" ;

Mod. MulOp ::= "%" ;

LTH. RelOp ::= "<" ;

LE. RelOp ::= "<=" ;

GTH. RelOp ::= ">" ;

GE. RelOp ::= ">=" ;

EQU. RelOp ::= "==" ;

NE. RelOp ::= "!=" ;

-- comments ------------------------------------------------

comment "#" ;

comment "//" ;

comment "/\*" "\*/" ;