Język Latte

Język Latte jest językiem imperatywnym, niemal podzbiorem imperatywnego fragmentu języka Java i może być z łatwością nań przetłumaczony.

Niniejszy opis jest umyślnie nieprecyzyjny i oparty na przykładach; dospecyfikowanie języka jest częścią zadania. Dla ułatwienia udostępniamy gramatykę języka bazowego w formacie dla BNFC (dla C++ lepiej użyć innej wersji).

Archiwum z programami testowymi:

lattests121017.tgz z 17.10.2012, MD5:49ca5702ca9795fb8f33d52b0b3c6fc3

Przykład: Hello world

```
// Hello world

int main () {
  printString("hello world") ;
  return 0 ;
}
```

Przykład: wypisz liczby parzyste do 10

```
// wypisz liczby parzyste do 10

int main () {
  int i ;
  i = 0 ;
  while (i < 10){
    if (i % 2 == 0) printInt(i) ;
    i++ ;
  }
  printInt(i) ;
  return 0 ;
}
```

Przykład: silnia na dwa sposoby

```
int main () {
  printInt(fact(7)) ;
  printInt(factr(7)) ;
  return 0 ;
}

// iteracyjnie
int fact (int n) {
  int i,r ;
  i = 1 ;
  r = 1 ;
  while (i < n+1) {
    r = r * i ;
    i++ ;
  }
  return r ;
}

// rekurencyjnie
int factr (int n) {
  if (n < 2)
    return 1 ;
  else
    return (n * factr(n-1)) ;
}
```

Struktura programu

Program w języku *Latte* jest listą definicji funkcji. Na definicję funkcji składa się typ zwracanej wartości, nazwa, lista argumentów oraz ciało. Funkcje muszą mieć unikalne nazwy. W programie musi wystąpić funkcja o nazwie `main` zwracająca `int` i nie przyjmująca argumentów (od niej zaczyna się wykonanie programu). Funkcje o typie wyniku innym niż `void` muszą zwracać wartość za pomocą instrukcji return

Funkcje mogą być wzajemnie rekurencyjne; co za tym idzie mogą być definiowane w dowolnej kolejności (użycie funkcji może występować przed jej definicją)

```
Program.   Program ::= [TopDef] ;
```

```
FnDef.     TopDef ::= Type Ident "(" [Arg] ")" Block ;
separator nonempty TopDef "" ;
Arg.       Arg ::= Type Ident;
separator  Arg "," ;
```

Instrukcje

Instrukcje: pusta,złożona,if,while,return jak w C/Javie. Dodatkowo instrukcjami są przypisanie, postinkrementacja, postdekrementacja (w wersji podstawowej języka l-wartościami są tylko zmienne).

Deklaracje zmiennych mogą występować w dowolnym miejscu bloku, jednak każda zmienna musi być zadeklarowana przed użyciem. Jeśli zmienna nie jest jawnie inicjalizowana w momencie deklaracji, jest inicjalizowana wartością domyślną (0 dla int, "" dla string, false dla bool).

Zmienne zadeklarowane w bloku nie są widoczne poza nim i przesłaniają zmienne o tej samej nazwie spoza bloku. W obrębie bloku zmienne muszą mieć unikalne nazwy.

```
Block.     Block ::= "{" [Stmt] "}" ;
separator  Stmt "" ;
Empty.     Stmt ::= ";" ;
BStmt.     Stmt ::= Block ;
Decl.      Stmt ::= Type [Item] ";" ;
NoInit.    Item ::= Ident ;
Init.      Item ::= Ident "=" Expr ;
separator nonempty Item "," ;
Ass.       Stmt ::= 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 ;
SExp.      Stmt ::= Expr  ";" ;
```

Typy

Typy int,boolean,void jak w Javie; string odpowiada String. Nie ma konwersji pomiedzy typami. Wprowadzenie niejawnych konwersji (rzutowań) będzie traktowane jako błąd, nie zaś ulepszenie.

Dla potrzeb analizy semantycznej może być pożyteczne wprowadzenie typów funkcyjnych.

```
Int.       Type ::= "int" ;
Str.       Type ::= "string" ;
Bool.      Type ::= "boolean" ;
Void.      Type ::= "void" ;
internal   Fun. Type ::= Type "(" [Type] ")" ;
separator  Type "," ;
```

Wyrażenia

Podzbiór zbioru wyrażeń dostępnych w Javie:

```
EVar.      Expr6 ::= Ident ;
ELitInt.   Expr6 ::= Integer ;
ELitTrue.  Expr6 ::= "true" ;
ELitFalse. Expr6 ::= "false" ;
EApp.      Expr6 ::= Ident "(" [Expr] ")" ;
EString.   Expr6 ::= String ;
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 ;
```

Wyrażenie logiczne zwracają typ `boolean` i są obliczane leniwie (drugi argument nie jest wyliczany gdy pierwszy determinuje wartość wyrażenia).

Napisy

Napisy podobnie jak w Javie, czyli zmienne typu `string` zawierają referencję do napisu, zaalokowanego na stercie.

Napisy moga występować jako: literały, wartości zmiennych, argumentów i wyników funkcji

Napisy mogą być użyte jako argumenty wbudowanej funkcji `printString`

Napisy mogą być konkatenowane przy pomocy operatora +. Wynikiem tej operacji jest nowy napis będący konkatenacją argumentów

Predefiniowane funkcje

Są dostępne predefiniowane funkcje:

```
void printInt(int)
void printString(string)
void error()
int readInt()
string readString()
```

Funkcja `error` wypisuje `runtime error` i kończy wykonywanie programu.

Funkcja `readString` wczytuje jedną linię z wejścia i daje ją jako wynik.

Parametry funkcji

Wszystkie parametry są przekazywane przez wartość. Wewnątrz funkcji parametry formalne zachowują się jak zmienne lokalne (czyli przypisania na nie są dozwolone).

Przekazanie napisu jako parametru odbywa sie poprzez przekazanie przez wartość referencji do napisu.

Latte a Java

Programy w Latte mogą być bez większych trudności przetłumaczone na język Java, poprzez opakowanie wszystkich funkcji w klasę jako publiczne statyczne metody. Metoda `main` będzie musiała dostać inną sygnaturę. Pewną trudność mogą sprawić zagnieżdżone bloki. Niektóre rozszerzenia mogą oczywiście wymagać dodatkowych zabiegów.

Rozszerzenia

Tablice jednowymiarowe i pętle for

Tablice jak w Javie, czyli zmienne typu tablicowego zawierają referencję do tablicy, zaalokowanej na stercie. Tablice są tworzone jawnie, przy użyciu operatora `new`. Zmienne typu tablicowego mają atrybut length (np. `a.length`)

Przykłady deklaracji tablic

```
  int[] a;
  string[] b;
```

Tworzenie tablic może (ale nie musi) być połączone z deklaracją:

```
  a = new int[20];
  int[] c = new int[30];
```

Funkcje mogą przyjmować tablice jako argumenty i dawać je w wyniku:

```
int[] sum (int[] a, int[] b) {
  int[] res = new int [a.length];
  int i = 0;

  while (i < a.length) {
    res[i] = a[i] + b[i];
    i++;
  }
  return res;
}
```

W ramach tego rozszerzenia należy również zaimplementować prosty wariant pętli `foreach`:

```
    for (int x : a)
     printInt(x);
```

Testy dla tego rozszerzenia znajdują się w katalogu `extensions/arrays1`.

Struktury

Rudymentarna wersja obiektów, uwzględniająca jedynie atrybuty, bez metod i dziedziczenia. Zmienne typu strukturalnego zawierają referencję do struktury zaalokowanej na stercie. Przypisanie oznacza przypisanie referencji. Przekazanie struktury jako parametru odbywa sie poprzez przekazanie przez wartość referencji. Powyższe odnosi sie równiez do następnego rozszerzenia ("Obiekty")

```
class list {
  int elem;
  list next;
}

int main() {
  printInt(length(fromTo(1,50)));
  printInt(length2(fromTo(1,100)));
}

int head (list xs) {
  return xs . elem;
}

list cons (int x, list xs) {
  list n;
  n = new list;
  n.elem = x;
  n.next = xs;
  return n;
}

int length (list xs) {
  if (xs==(list)null)
    return 0;
  else
    return 1 + length (xs.next);
}

list fromTo (int m, int n) {
  if (m>n)
    return (list)null;
  else
    return cons (m,fromTo (m+1,n));
}

int length2 (list xs) {
  int res = 0;
  while (xs != (list)null) {
    res++;
    xs = xs.next;
  }
  return res;
}
```

Testy dla tego rozszerzenia znajdują się w katalogu `extensions/struct`.

Obiekty

Klasy i obiekty, z pojedynczym dziedziczeniem (ale bez zastępowania metod).

Przykład klasy:

```
  int main () {
  Counter c;
  c = new Counter;
  c.incr();
  c.incr();
  c.incr();
  int x = c.value();
  printInt(x);
  return 0;
}

class Counter {
  int val;

  void incr () {val++; return;}
  int value () {return val;}
}
```

Przykład dziedziczenia:

```
class Point2 {
  int x;
  int y;

  void move (int dx, int dy) {
     x = x + dx;
     y = y + dy;
  }

  int getX () { return x; }

  int getY () { return y; }
}

class Point3 extends Point2 {
  int z;

  void moveZ (int dz) {
    z = z + dz;
  }

  int getZ () { return z; }
}

class Point4 extends Point3 {
  int w;

  void moveW (int dw) {
    w = w + dw;
  }

  int getW () { return w; }
}

int main () {
  Point2 p = new Point3;
  Point3 q = new Point3;
  Point4 r = new Point4;

  q.move(2,4);
  q.moveZ(7);
  p = q;

  p.move(3,5);

  r.move(1,3);
  r.moveZ(6);
  r.moveW(2);

  printInt(p.getX());
  printInt(p.getY());
  printInt(q.getZ());
  printInt(r.getW());
  return 0;
}
```

Testy dla tego rozszerzenia znajdują się w katalogu `extensions/objects1`.

Metody wirtualne

Ograniczenie poprzedniego rozszerzenia, zabraniające zastępowania metod przy dziedziczeniu wyklucza realne programowanie obiektowe. Stąd w tym rozszerzeniu należy zrealizować zastępowanie metod, traktując wszystkie metody jako wirtualne (jak w Smalltalku).

Przykład:

```
  class Node {
  Shape elem;
  Node next;

  void setElem(Shape c) { elem = c; }

  void setNext(Node n) { next = n; }

  Shape getElem() { return elem; }

  Node getNext() { return next; }
```

```
}

class Stack {
  Node head;

  void push(Shape c) {
    Node newHead = new Node;
    newHead.setElem(c);
    newHead.setNext(head);
    head = newHead;
  }

  boolean isEmpty() {
    return head==(Node)null;
  }

  Shape top() {
    return head.getElem();
  }

  void pop() {
    head = head.getNext();
  }
}

class Shape {
  void tell () {
    printString("I'm a shape");
  }

  void tellAgain() {
     printString("I'm just a shape");
  }
}

class Rectangle extends Shape {
  void tellAgain() {
    printString("I'm really a rectangle");
  }
}

class Circle extends Shape {
  void tellAgain() {
    printString("I'm really a circle");
  }
}

class Square extends Rectangle {
  void tellAgain() {
    printString("I'm really a square");
  }
}

int main() {
  Stack stk = new Stack;
  Shape s = new Shape;
  stk.push(s);
  s = new Rectangle;
  stk.push(s);
  s = new Square;
  stk.push(s);
  s = new Circle;
  stk.push(s);
  while (!stk.isEmpty()) {
    s = stk.top();
    s.tell();
    s.tellAgain();
    stk.pop();
  }
  return 0;
}
```

Odśmiecanie

To rozszerzenie nie zmienia składni ani semantyki języka, a jedynie usprawnia zarządzanie pamięcią poprzez wprowadzenie zwalniania niepotrzebnych już napisów. Wystarczy prosty odśmiecacz oparty na zliczaniu odwołań (reference counting).