Techniki Kompilacji - Projekt wstępny

Autor: Konstantin Panov
Opiekun: Konrad Grochowski
Temat: Język do opisu gier planszowych

Opis funkcjonalny

Język służy wspomaganiu implementacji procesów używających heksagonalnych siatek. Język rozszerza standardowe funkcjonalności imperatywnego języka programowania (zmienne liczbowe, zmienne tekstowe, listy, instrukcje warunkowe, instrukcje pętli, definicje funkcji) o funkcjonalności ułatwiające działania na siatkach heksagonalnych.
Silne, statyczne typowanie zmiennych. Konwersje nie są dopuszczalne.

Funkcjonalności standardowe

Zmienne

Zmienne liczbowe stałoprzecinkowe, zmienne liczbowe zmiennoprzecinkowe i zmienne tekstowe. Przykład:

```
int int_var_name = 100;
float float_var_name = 100.123;
string string_var_name = "Hello, world!";
```

Operatory arytmetyczne

- Operator przypisywania - “=”
- Operator dodawania - “+”
- Operator odejmowania/negacji - “-”
- Operator mnożenia - “*”
- Operator dzielenia - “/”
- Operator reszty z dzielenia - “%”

Przykład:

```
int var = 100;
var = var + 200;
var = var - 100;
var = var * 3;
var = var / 6;
var = var %
var = -10;
```

Operatory porównania

- Operator równa się - “==”
- Operator nie równa się i - “!=”

- Operator mniejsze niż - “<”
- Operator większe niż - “>”
- Operator mniejsze lub równe - “<=”
- Operator większe lub równe - “>=”
- Operator negacji logicznej - “!”

Operatory logiczne

- Iloczyn logiczny - “and”
- Suma logiczna - “or"

Priorytety operatorów

Priorytet	Operator	Opis	Łączność
1	or	Operator logiczny Or	Od lewej do prawej
2	and	Operator logiczny And	Od lewej do prawej
3	<, <=, >, >=, ==, !=	Porównanie	Od lewej do prawej
4	at, by, beside	Operatory na hexgrid	Od prawej do lewej
5	+, -	Dodawanie, Odejmowanie	Od lewej do prawej
6	*, /, %	Mnożenie, Dzielenie, Reszta	Od lewej do prawej
7	!, -	Negacja	Od prawej do lewej
9	[]	Indeksowanie	Od lewej do prawej

Instrukcje warunkowe

Instrukcje warunkowe zaznaczane są słowem kluczowym *if*, warunek podaje się w nawiasach okrągłych, a block wykonywalny w nawiasach klamrowych. Przykład:

```
int var = 100;
if (var >= 50)
{
    var = var / 2;
}
elif (var < 50 and var > 25)
{
   var = var * 2;
}
else
{
   var = var * 4;
}
```

Listy

Listy o dynamicznej długości i jednorodnym typie. Początkowe wartości pozycji listy są wypełniane wartością *null*. Przykład:

```
int[] array_example = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
array_example[0] = 3;
array_example add 11;

int[][] matrix_example;
matrix_example = [[1, 2, 3], [4, 5, 6]]
matrix add [7, 8, 9];
matrix_example[0][1] = 10;
```

Instrukcje pętli

Instrukcje pętli zaznaczane są słowem kluczowym *foreach* po czym następuje nazwa zmiennej w której są przechowywane wartości z listy wskazanej po słowie *in*. Przykład:

```
string[] loop_array = ["red", "green", "blue", "blue", "red", "blue"];
int count_blue = 0;
int count_red = 0;
foreach string color in loop_array
{
    if (color == "blue")
    {
        count_blue = count_blue + 1;
    } else if (color == "red")
    {
        count_red = count_red + 1;
    }
}
```

Definicje funkcji

Definicje funkcji zaznaczane są typem wartości wyjściowej oraz słowem kluczowym *func*, Po słowie kluczowym idą nawiasy okrągłe w których przez przecinek wypisane są typy i nazwy argumentów. Po nawiasach idą nawiasy klamrowe oznaczające blok instrukcji, które zostaną wykonane. Wyjście z funkcji oznaczane jest słowem kluczowym return oraz zwracaną wartością. Funkcja nie może być zdefiniowana w środku innej funkcji. Przykład:

```
func int foo(int a, int b)
{
    return a + b;
}
```

Funkcjonalności dziedzinowe

Typ zmiennej *hexgrid*

Zmienna typu *hexgrid* reprezentuje siatkę heksagonalną. Do implementacji siatki heksagonalnej używa się współrzędnych sześciennych (cube coordinates). Więcej na temat reprezentacji siatki heksagonalnej i algorytmów operacji na niej można znaleźć na tej stronie. Współrzędne pozycji na siatce to [q, r, s]. Zmienna hexgrid może być modyfikowana za pomocą konstrukcji dodawania/usuwania z niej "klatek". Natomiast sama klatka w siatce nie jest modyfikowalna, ale można ją zdjąć z siatki, zmienić nazwę i wrócić na siatkę.

Definicja zmiennej typu *hexgrid*

Utworzenie nowej zmiennej typu hexgrid o nazwie *hexgrid_name*. Tworzy pustą siatkę. Do pustej siatki można dodawać nowe klatki.

```
hexgrid hexgrid_name;
```

Utworzenie niepustej siatki

```
hexgrid hexgrid_name = <"blue" at [0, 0, 0], "red" at [0, 0, 1]>;
```

Dodawanie klatki do siatki *hexgrid*

W celu dodania klatki do heksagonalnej siatki należy podać jej nazwę oraz położenie na klatce. Mając siatkę o nazwie *hexgrid_name* możemy dodać klatkę o nazwie *cell_name* na pozycji q, r, s w następujący sposób:

```
add cell_name to hexgrid_name at [q, r, s];
```

Pozycja klatki może być wskazana jako listy liczb stałoprzecinkowych o długości 3. Na przykład:

```
int[] pos = [1, 2, 3];
add cell_name to hexgrid_name at pos;
```

Usuwanie klatki z siatki *hexgrid*

W celu usunięcia klatki z heksagonalnej siatki należ podać wyłącznie pozycję klatki. Mając siatkę o nazwie *hexgrid_name* możemy usunąć klatkę na pozycji q, r, s w następujący sposób:

```
remove [q, r, s] from hexgrid_name;
```

Przemieszczenie klatki z siatki *hexgrid*

Nazwę usuniętej z siatki klatki można zapisać do zmiennej tekstowej:

```
string cell_name;
```

```
move [q, r, s] from hexgrid_name to cell_name;
```

Albo do innej siatki:

```
move [q, r, s] from hexgrid_1_name to hexgrid_2_name at [q, r, s];
```

Wyszukiwanie nazwy klatki w siatce według pozycji

Takie wyszukiwanie zwraca nazwę klatki, gdy pozycja jest używana w siatce lub wartość null, gdy na podanej pozycji nie ma klatki. Przykład:

```
string cell_name = hexgrid_name on [1, 2, 3];
```

Wyszukiwanie pozycji klatki w siatce według nazwy

Takie wyszukiwanie zwraca macierz będącą listą pozycji na siatce, w których znajdują się klatki o podanej nazwie. Jeśli na siatce nie ma klatek o podanej nazwie, zwracana zostaje pusta lista. Przykład:

```
int[][] positions = hexgrid_name by cell_name;
```

Wyszukiwanie klatek sąsiadujących z wybraną klatką na siatce.

Takie wyszukiwanie zwraca listę pozycji klatek na siatce, które sąsiadują z wybraną pozycją i nie są puste.
Przykład:

```
int[][] positions = hexgrid_name beside [1, 2, 3];
```

Przykłady użycia języka

Przykład 1

Utworzenie i wypełnienie siatki heksagonalnej. Zliczanie sąsiadów klatki według wartości.

```
hexgrid example_hexgrid = <"blue"   at [0, 0, 0],
                           "red"    at [0, 0, 1],
                           "blue"   at [0, 0, -1],
                           "red"    at [0, 1, 0],
                           "yellow" at [0, -1, 0],
                           "red"    at [1, 0, 0],
                           "blue"   at [-1, 0, 0]>;
int blue_count = 0;
foreach int[] pos in example_hexgrid beside [0, 0, 0]
{
    if (example_hexgrid at pos == "blue")
    {
        blue_count = blue_count + 1;
    }
```

```
}
```

Przykład 2

Wyszukiwanie na siatce heksagonalnej klatek o wybranej nazwie i usuwanie tych, co znajdują się na dodatniej stronie osi q.

```
hexgrid example_hexgrid = <"blue"   at [0, 0, 0],
                           "red"    at [0, 0, 1],
                           "blue"   at [0, 0, -1],
                           "red"    at [0, 1, 0],
                           "yellow" at [0, -1, 0],
                           "red"    at [1, 0, 0],
                           "blue"   at [-1, 0, 0]>;
foreach int[] pos in example_hexgrid by "blue"
{
    if (pos[0] > 0) {remove pos from example_hexgrid;}
}
```

Formalny opis gramatyki

```
(* Nonterminal *)
script = {stmnt_or_func_def};
stmnt_or_func_def = stmnt_with_semicolon
                  | func_def
                  ;
func_def = func_kw, declr, params_list, stmnt_block;
params_list = l_r_bracket, [{declr, comma}, declr], r_r_bracket;

  (* Statements *)
stmnt_block = l_c_bracket, {stmnt_with_semicolon}, r_c_bracket;
stmnt_with_semicolon = stmnt, semicolon;
stmnt = func_call_or_assignment
      | declr_or_init
      | if_stmnt
      | foreach_stmnt
      | return_stmnt
      | add_stmnt
      | remove_stmnt
      | move_stmnt
      ;

func_call_or_assignment = id, arg_list_or_assignment_expr;
declr_or_init = declr, [assignment_expr];
if_stmnt = if_block, {elif_block}, [else_block];
```

```
foreach_stmnt = foreach_kw, declr, in_kw, expr, stmnt_block;
return_stmnt = return_kw, expr;
add_stmnt = add_kw, expr, to_kw, id, at_kw, expr;
remove_stmnt = remove_kw, expr, from_kw, id;
move_stmnt = move_kw, expr, from_kw, id, to_kw, id, at_kw, expr;

  (* Statements' helpers *)
declr = var_type, id;
if_block = if_kw, l_r_bracket, expr, r_r_bracket, stmnt_block;
elif_block = elif_kw, l_r_bracket, expr, r_r_bracket, stmnt_block;
else_block = else_kw, stmnt_block;
arg_list_or_assignment_expr = arg_list
                            | assignment_expr
                            ;
assignment_expr = assignment_operator, expr;

  (* Expressions *)
expr = or_expr;
or_expr = and_expr, {or_operator, or_expr};
and_expr = comparison_expr, {and_operator, and_expr};
comparison_expr = hexgrid_expr, {comparison_operator, hexgrid_expr};
hexgrid_expr = {so_arithm_expr, hexgrid_operator}, so_arithm_expr;
so_arithm_expr = fo_arithm_expr, {so_arithm_operator, fo_arithm_expr};
fo_arithm_expr = negation_expr, {fo_arithm_operator, negation_expr};
negation_expr = [negation_operator], indexing_expr;
indexing_expr = term, {l_s_bracket, expr, r_s_bracket};

  (* Terms *)
term = literal
     | id_or_func_call
     | array
     | hexgrid
     | l_r_bracket, expr, r_r_bracket
     ;

literal = number_literal | text_literal;
id_or_func_call = id, [arg_list];
array = l_s_bracket, [expr, {comma, expr}], r_s_bracket;
hexgrid = l_a_bracket, [inside_hexgrid], r_a_bracket;
inside_hexgrid = hexgrid_cell, {comma, hexgrid_cell};

  (* Terms' helpers *)
number_literal = integer_part, [dot, fraction_part];
integer_part = zero_digit
             | nonzero_digit, {digit}
             ;
```

```
fraction_part = digit, {digit};
text_literal = quote, {in_text_character}, quote ;
arg_list = l_r_bracket, [{expr, comma}, expr], r_r_bracket;
hexgrid_cell = expr, at_kw, expr;

  (* Nonterminal helpers *)
var_type = single_var_type, [array_sign, {array_sign}];
single_var_type = number_type
                | text_type
                | hexgrid_type
                ;
number_type = integer_type
            | float_type
            ;
text_type = string_type;
id = letter, {in_id_character};
in_id_character = letter
                | digit
                | underscore
                ;
letter = upper_case_letter
       | lower_case_letter
       ;
digit = nonzero_digit
      | zero_digit
      ;
in_text_character = whitespace
                  | letter
                  | digit
                  ;

(* Terminal *)

  (* Characters *)
whitespace = "\u0009" | "\u000B" | "\u000C";
upper_case_letter = "[A-Z]";
lower_case_letter = "[a-z]";
nonzero_digit = "[1-9]";
zero_digit = "0";
underscore = "_";
semicolon = ";";
comma = ",";
dot = ".";
quote = '"';
l_c_bracket = "{";
r_c_bracket = "}";
```

```
l_r_bracket = "(";
r_r_bracket = ")";
l_s_bracket = "[";
r_s_bracket = "]";
l_a_bracket = "<";
r_a_bracket = ">";

  (* Types *)
integer_type = "int";
float_type = "float";
string_type = "string";
array_sign = "[]";
hexgrid_type = "hexgrid";

  (* Operators *)
and_operator = "and";
or_operator = "or";
comparison_operator = ">" | "<" | ">=" | "<=" | "==" | "!=";
hexgrid_operator = "beside", "on", "by";
fo_arithm_operator = "/" | "*";
so_arithm_operator = "+" | "-";
negation_operator = "-" | "!";
assignment_operator = "=";

  (* Keywords *)
func_kw = "func";
return_kw = "return";
if_kw = "if";
elif_kw = "elif";
else_kw = "else";
foreach_kw = "foreach";
in_kw = "in";
at_kw = "at";
add_kw = "add";
remove_kw = "remove";
move_kw = "move";
to_kw = "to";
from_kw = "from";
```

Opis techniczny realizacji

- Wybrany język - C++ z biblioteką Boost (Boost Test)
- Narzędzie do budowania programu - Scons
- Program wczytuje kod źródłowy ze standardowego wejścia np.

```
./hexgrider < source_code.txt
```

albo

```
cat source_code.txt | ./hexgrider
```