PROJEKT WSTĘPNY TKOM

MACIEJ KAPUŚCIŃSKI

**1. Temat projektu**

Stworzenie języka pozwalającego definiować własne struktury danych oraz interpretera.

**2. Opis języka**

Struktury będą definiowane w sposób następujący:

class Nazwa {

/\* ---zawartość--- \*/

}

Struktury będą posiadały definiowane konstruktory, destruktory, dane, funkcje.

Typy: int, string, char

instrukcja switch

pętla while

definicja funkcji globalnych i wywoływanie

operacje matematyczne o różnych priorytetach

ściśle zdefiniowany main

**3. Przykłady**

Przykład 1:

class Ulamek

{

int licznik;

int mianownik;

}

main

{

Ulamek i;

i.licznik = 1;

i.mianownik = 2;

}

Przykład 2:

class Queue

{

public constructor()

{

array = new int[2];

capacity = 2;

size = 0;

}

public destructor()

{

delete array;

}

private int array;

private int capacity;

private int size;

void push(int n)

{

array[size] = n;

size = size + 1;

switch(size)

{

if capacity:

realloc(2\*capacity);

break;

default:

break;

};

}

int pop()

{

switch(size)

{

if 0:

return

default:

size = size - 1;

return array[size];

};

}

void realloc(int new\_capacity)

{

int[] new\_array = new int[new\_capacity];

int i = 0;

while (i < size)

{

new\_array[i] = array[i];

i = i + 1;

};

delete array;

array = new\_array;

}

}

main

{

Queue queue;

queue = new Queue();

queue.push(5);

print(queue.pop());

}

**4. Gramatyka**

program = {deklaracje}, main ;

deklaracje = funkcja | klasa | (deklaracja\_zmiennej, ";") ;

funkcja = (typ | "void") , tekst, wspolne\_cialo ;

klasa = "class", tekst, "{", {ciało\_klasy}, "}" ;

ciało\_klasy = ["public" | "private"], (konstruktor | destruktor | deklaracje) ;

konstruktor = "constructor", wspolne\_cialo ;

destruktor = "destructor", wspolne\_cialo ;

wspolne\_cialo = "(", [ deklaracja\_zmiennej, {",", deklaracja\_zmiennej}], ")", "{", {polecenie}, "}" ;

deklaracja\_zmiennej = typ, tekst;

typ = "int" | "string" | "char" ;

main = "main", "{", {polecenie}, "}";

polecenie = (deklaracja\_wewnatrz | przypisanie | switch | petla | wywolanie\_fcji | "break" | ("return", [wartosc]) | ("delete", tekst) | ("print", "(", wartosc, ")")) , ";" ;

deklaracja\_wewnatrz = (typ, tekst) | (typ, przypisanie) ;

przypisanie = tekst, [ "[", wartosc, "]" ], "=", (wartosc | definicja);

definicja = "new", typ, [ ("[", wartosc, "]") | "(", [wartosc, {",", wartosc}], ")" ] ;

switch = "switch", "(", expression, ")", "{", {"if", constant | variable, ":", {polecenie}}, ["default", ":", {polecenie}], "}" ;

petla = "while", "(", logic, ")", "{", {polecenie}, "}" ;

wywolanie\_fcji = [tekst, "."], tekst, "(", [wartosc, {",", wartosc}], ")" ;

expression = term , { ("+" | "-") , expression} ;

term = factor, { ("\*" | "/") , term } ;

factor = constant | variable | "(" , expression, ")" ;

constant = integer ;

variable = [tekst, "."], tekst, [ "[", wartosc, "]" ] ;

logic = expression, logic\_operator, expression;

logic\_operator = "==" | "!=" | "<=" | ">=" | ">" | "<" ;

tekst = character, {character} ;

wartosc = expression | string | character | wywolanie\_fcji;

string = '"', {character}, '"' ;

letter = #'[A-Za-z]'

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

nonzerodigit= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

integer = ([ "-" ], nonzerodigit, { digit }) | "0" ;

character = letter | digit | special\_character ;

special\_character = ?znaki specjalne?;

W typach będą też uwzględnione nazwy klas zdefiniowanych przez użytkownika. Komentarze i białe znaki nie są uwzględnione w gramatyce, albowiem będą pomijane.

**5 Zarys struktury interpretera**

Programy będzie napisany w języku C++. Na wejście będzie podawana ścieżka do kodu do zinterpretowania. Interpreter będzie się składał z kilku części:

* analiza leksykalna - zmiana znaków na tokeny
* analiza składniowa - sprawdzenie, czy tokeny są zgodne z gramatyką i stworzenie struktury składniowej (prawdopodobnie drzewa)
* analiza semantyczna - sprawdzenie, czy w strukturze składniowej znajdują się poprawne składniowo, ale niemożliwe do wykonania operacje
* wykonanie kodu - wykonanie po kolei instrukcji ze struktury składniowej zgodnie z ich znaczeniem

Będzie też dostępny moduł obsługi błędów i zarządca tablicy symboli.