PROJEKT KOŃCOWY TKOM

MACIEJ KAPUŚCIŃSKI

1. Temat projektu

Projekt polega na stworzeniu interpretera prostego języka, pozwalającego na zdefiniowanie własnych struktur danych.

2. Opis języka

Struktury będą definiowane w sposób następujący:

```
class Nazwa {
        Constructor() {}
        Destructor() {}
        void fun(){}
        public int a;
        private int b;
/* ---zawartość--- */
}
```

- Struktury posiadają definiowane konstruktory, destruktory, dane, funkcje z kontrolą dostępu dla danych.
- Wbudowane typy: int, string, char, tablica intów, tablica charów
- Jako że sam język nie operuje na pamięci, nie jest wymagane niszczenie tablic
- Wbudowana instrukcja switch, pętla while, możliwość definicji zmiennych i funkcji globalnych, obsługa operacji matematycznych o różnych priorytetach, proste porównania dla instrukcji while.
- Zmienne przekazywane do funkcji jako referencje.
- Ściśle zdefiniowany blok Main, zawsze na końcu programu.
- Wbudowana funkcja print(), pozwalająca na wypisanie typów wbudowanych
- Błędy sygnalizowane używając throw

3. Przykłady

Przykład 1:

```
class Ulamek
{
        int licznik;
        int mianownik;
        void divide(Ulamek other)
        {
                licznik = licznik * other.mianownik;
                mianownik = mianownik * other.licznik;
```

```
	}
	void multiply(Ulamek other)
	{
		licznik = licznik * other.licznik;
		mianownik = mianownik * other.mianownik;
	}
}
main
{
	Ulamek i = new Ulamek();
	i.licznik = 2 + 2 * 2 / (1 + 1);
	i.mianownik = 8 / 2 - 1;
	Ulamek j = new Ulamek();
	j.licznik = 3;
	j.mianownik = 4;
	i.multiply(j);
	print("licznik:", i.licznik, "mianownik:", i.mianownik);
}
```

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 0;
			default:
				size = size - 1;
				return array[size];
		}
	}

	void realloc(int newCapacity)
	{
		int newArray = new int[newCapacity];
		int i = 0;
		while (i < size)
		{
			newArray[i] = array[i];
			i = i + 1;
		}
		delete array;
		array = newArray;
	}
}

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"], deklaracja_zmiennej, ";") | (konstruktor | destruktor | funkcja) ;
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 są też uwzględnione nazwy klas zdefiniowanych przez użytkownika. Komentarze i białe znaki nie są uwzględnione w gramatyce, albowiem są pomijane przez lekser.

5 Struktura interpretera

Programy jest napisany w języku C++, budowany jest przy pomocy IDE Code::Blocks. Plik wykonywalny jest aplikacją konsolową, na wejście jest podawana ścieżka do pliku z kodem do zinterpretowania. W wypadku błędu zostanie on wypisany na wyjście standardowe, w przeciwnym wypadku program wykona się wypisując jedynie zadane przez użytkownika polecenia print.