# Deklaracja Języka

Warszawa, 11 kwietnia 2023

Anna Stawiska

---


# MatchaLatte

Język MatchaLatte jest oparty o język C oraz gramatykę języka Latte wspomnianego w treści zadania.

1. Zawiera 3 typy: int, string oraz bool
   Deklaruje się je podobnie jak w C.
   Zauważmy też że podobnie jak w C mamy główną funkcję int main(), która zwraca typ int.
   Przypisania, operacje arytmetyczne i wypisywanie na wyjście również wygląda podobnie jak w C.
   Operacja "+" na stringach to poprostu ich konkatenacja.
   Komentarze również wyglądają jak w C (poprzez "//" lub /* */)

```
// pierwszy program
int main() {
    int example_var;
    bool example_bool = true;
    string example_string = "my string example";

    example_var = 5;

    return 0;
}

// drugi program
int main() {
    int a = 8;
    int b = 4;

    int c = a + b;
    c = a - b;
    b = c / a;
    a = b * c;
    a = b % c;

    bool check = true;
    check = (a == b);
    check = a != b;

    string s = "Ala";
    s = s + " ma";

    return 0;
```

```
    }

    // trzeci przyklad - wypisywanie na wyjscie
    int main() {
        print("Ala ma kota");

        int a = 6;
        bool t = true;
        string s = "Test";

        print(a);
        print(t);
        print(s);

        return 0;
    }

```

W MatchaLatte są również funkcje zwaracające typy int, string, bool (poprzez return [wyrażenie danego typu]) oraz funkcje typu void które nic nie zwracają (poprzez samo return, tak jak w C). Przekazywanie parametrów odbywa się przez zmienną i przez wartość. Istnieje również rekurencja.

Przykład ilustrujący powyższe rzeczy:

```
    string string_fun() {
        print("Hello world!");

        return "hello world";
    }

    int increment(int x) {
        x++;
        return x;
    }

    int increment_ref(int@ x) {
        x++;
        return 1;
    }

    int recursive(int x) {
        if (x == 0) {
            return 0;
        }
        else {
            recursive (x - 1);
        }

        return 1;
    }

```

2. Konstrukcja while oraz if/if-elsewygląda jak w C z jedynym wyjątkiem, że bloki kodu w if-ach i while są otoczone nawiasami klamrowymi.

```
    int main() {
        int i = 0;
```

```
3
4           while (true) {
5               if (i % 2 == 0) {
6                   i++;
7               }
8               else {
9                   if (i == 8) {
10                      i--;
11                  }
12                  i++;
13              }
14          }
15
16          return 0;
17      }
18
```

3. Istnieją zmienne lokalne i globalne oraz zagnieżdżone procedury/funkcje
4. W języku mamy statyczne typowanie (tj. zawsze terminująca faza kontroli typów przed rozpoczęciem wykonania programu)
5. Obsługa błędów (błędy typów pojawiają się przy kontroli typów - statyczne typowanie):

```
1       int main() {
2           int a = 8;
3           int b = 0;
4
5           string s = "Test";
6
7           int c = a / b; // "Error : Illegal operation - attempt to divide
    by 0."
8           int d = a / s; //  "Error : Illegal operation - operation on
    wrong types."
9
10          return 0;
11      }
12
```

6. Język obsługuje dowolnie zagnieżdżone funkcje z zachowaniem statycznego wiązania zmiennych.

```
1       int x = 0;
2
3       string string_fun() {
4
5           int increment(int x) {
6               x++;
7               return 1;
8           }
9
10          int increment_ref(int@ x) {
11              x++;
12              return x;
13          }
14
15          int x = 2;
16          int b = increment(x);
17
```

```
        increment_ref(x);

        print(x); // 3
        print(b); // 3

        return "hello world";
    }

    int main() {
        string s = string_fun(); // 3 3
        print(x); // 0

        return 0;
    }

```

7. Dodatkowy typ który występuje to "typ funkcyjny", który umożliwia tworzenie zmiennych które mogą przechowywać funkcje.
   Składnia jest następująca:

```
    fun [(typy argument w) -> typ zwracany] nazwa_funkcji = funkcja;

```

   Przykład:

```
    int increment(int x) {
        x++;
        return x;
    }

    int example(int x) {
        int add(int n) {
            return x + n;
        }

        return add(3);
    }

    int main() {
        fun [(int) -> int] f = increment;
        fun [(int, int) -> int] g = \(int a, int b) -> int => {return a +
    b;};
        fun [(int) -> int] add = example;

        int a  = example(1); // 4
        int b = example(5); // 8
        int c = g(4, 7); // 11
        int d = f(4); // 5

        return 0;
    }

```

   W powyższym przykładzie widzimy, że istnieją funkcje anonimowe (tzw. "lambdy"). Mogą być też zapisywane w zmiennej typu funkcyjnego.
   Funkcje anonimowe mogą przyjmować dowolną liczbę parametrów, które są zapisywane w nawiasach okrągłych. Funkcje anonimowe mogą zwracać dowolny typ, który jest zapisywa-

ny po strzałce.

Dodatkowo widzimy przykład (oraz działanie) zwracania funkcji w wyniku, jako domknięcia (funkcja example(int)).

8. Język dopuszcza również przekazywanie funkcji jako argumentu funkcji.

```
1     int test(fun [(int, int) -> int] f) {
2         int x = f(5, 6);
3         return x;
4     }
5
6     int main() {
7         fun [(int, int) -> int] g = \(int a, int b) -> int => {return a +
      b;};
8
9         test(g); // 11
10        return 0;
11    }
12
```