GTL - The Green Text Programming Language

Jakub Kierznowski, Mateusz Kotarba, Juliusz Kociński

5 czerwca 2025

1 Wstęp

Green Text Programming Language (GTL) to uniwersalny, imperatywny język programowania, którego głównym założeniem jest składnia możliwie zbliżona do Greenext - formatu krótkich, charakterystycznie pisanych historyjek internetowych, spopularyzowanego przez portal 4chan.org, a następnie różne fora. W Polsce szerzej jest znany jako klasyczny format pasty (i zapisywany czerwonym tekstem).

2 Komentarze

- Każda linia musi zaczynać się od znaku > poprzedzonym spacją. Jest ona traktowana jako kod. Po tym znaku można dać dowolną ilość spacji lub tabów, aby uczynić program bardziej czytelnym.
- Linia, która nie zaczyna się od >, jest traktowana jako komentarz.
- Komentarze w tej samej linijce muszą być poprzedzone spacją i znakiem #.

Składnia:

```
 > {To jest kod}
To jest komentarz
 > {To jest kod} # To jest komentarz w linii
```

3 Zmienne

- Zmienne muszą być deklarowane z podaniem typu: `type nazwa_zmiennej`.
- Nazwy zmiennych mogą być dowolnym znakiem z unicode (nawet może być to emoji), z wyjątkiem `'` i `,`. Zmienna nie może zaczynać się od cyfry. Zmienna nie może być słowem kluczowym.
- GTL obsługuje szereg typów prostych:

int	→	see
double	→	taste
string	→	hear
boolean	→	smell
struktury	→	spot

- Przypisanie do zmiennej odbywa się za pomocą słowa kluczowego `is` po czym następuje wyrażenie, lista wyrażeń lub wywołanie funkcji.

  Składnia definicji zmiennej i przypisania zmiennej:

```
> {typ zmiennej} {nazwa zmiennej}
> {typ zmiennej} {nazwa zmiennej} is {wywołanie funkcji, wyrażenie, lista wyrażeń}
> {nazwa zmiennej} is {wywołanie funkcji, wyrażenie, lista wyrażeń}
```

Przykład:

```
> see baddie is 9
> taste lunch is someone elses dinner
> smell flowers is c:
> flower is :c
```

- Dla typów złożonych:
  - `multiple` definiuje dynamiczną tablicę.
  - `about` i potem wyrażenie definiuje tablicę znanym rozmiarze.
  - `someone elses` definiuje referencję do zmiennej (nie jest tworzona jej kopia).
- Tablicę można inicjować słowem `is` i podaniem rozdzielonych przecinkami lub `and` wartości.
- Dostęp do tablicy jest przez poprzedzenie nazwy tablicy numerem elementu (zaczynając od 0), z dodatkiem `'th`. Przed `'th` może się pojawić tylko pojedyncza nazwa lub liczba całkowita. **Składnia dostępu i tworzenia tablicy:**

```
> {typ zmiennej} multiple {nazwa zmiennej}
> {typ zmiennej} about {wyrażenie} {nazwa zmiennej}
> {typ zmiennej} multiple {nazwa zmiennej} is {wyrażenie}
> and {wyrażenie}, {wyrażenie}
> {całkowity literał, pojedyncza nazwa}'th {nazwa tablicy}
```

Przykład dostępu:

```
> see multiple baddies
> see about 5 men is 1,2,3,4,5
> see man is 4'th men
> see best is 2
> man is best'th men
```

- Zmiana typu zmiennej "castowanie"odbywa się przez przypisanie do nowej zmiennej, zmiennej o innym typie.
- Przy operacji dodawania z zmienną typu `hear` (string), pozostałe zmienne są zamieniane na `hear` (string) i konkatenowane.

Składnia zmiany typu zmiennej:

```
> {typ zmiennej} {nazwa zmiennej} is {literał, funkcja, zmienna, wyrażenie (o innym typie)}
```

Przykład:

```
> taste baddie is 9.5
> see girl is baddie
> hear boy is joined by girl
```

4 Definiowanie struktur (typów)

Aby stworzyć nową strukturę/typ, należy użyć poniższej składni:

- `> look around` rozpoczyna definicję struktury.
- `> struct_name` to nazwa struktury.
- Następnie definiujemy zmienne, funkcje, pętle wewnątrz struktury.
- `> lose interest` oznacza zakończenie definicji struktury.
- Aby dostać się do zawartości pól zmiennych należy podać ich nazwę, a potem napisać `'s` i nazwę pola.
- Nazwy struktur muszą być poprawnymi nazwami (patrz nazwa zmiennej).
- Deklaracja struktury utrzymuje się w całym programie.

Składnia struktury:

```
> look around
> {nazwa_struktury}
> {typ_zmiennej} {nazwa_zmiennej}
> {deklaracja funkcji}
.
.
> lose interest
>
> spot {nazwa_struktury} {nazwa_zmiennej}
> {nazwa_zmiennej}'s {nazwa_pola} is {wyraźnie, funkcja}
> {inna_zmienna} is {nazwa_zmiennej}'s {nazwa_pola}
```

Przykład:

```
> look around
> Human
> see age
> hear name
> smell alive
> lose interest

> spot Human Anon
> Anon's age is 18
> see number is Anon's age
```

5 Zasięg zmiennych

- Zmienna obowiązuje do momentu napotkania pustej linii.
- Po napotkaniu pustej linii zmienne zdefiniowane powyżej przestają być dostępne.
- Nie stosuje się to do deklaracji funkcji i struktur. Ale sama zdefiniowana struktur (z danymi) już jest usuwana.

Przykład:

```
> see baddie
tutaj jest komentarz
> spit baddie  # baddie wciąż istnieje
drugi komentarz

> spit baddie  # błąd! baddie nie istnieje, ponieważ wystąpiła pusta linijka
```

6 Operacje logiczne

- GTL zawiera wartości logiczne:

false	→	:c
true	→	c:

- GTL zawiera operacje logiczne do porównywania zmiennych:

==	→	vibe with
!=	→	doesn't vibe with
<	→	beaten by
<=	→	doesn't beat
>	→	beats
=>	→	unbeaten by
&&	→	also
||	→	alternatively
zaprzeczenie	→	not

- GTL zawiera podstawowe operatory matematyczne:

X+Y	→	> X joined by Y
X+=Y	→	> X is joined by Y
X++	→	> X evolves
X*Y	→	> X breeding like Y times
X*=Y	→	> X is breeding like Y times
-Y	→	> the literal opposite of Y
X–	→	> X devolves
$\frac{1}{Y}$	→	> flipped Y
X%Y	→	> X whatever left from Y
X%=Y	→	> X is whatever left from Y

- Podobnie jak w innych językach operator modulo, ++, – można wykonać tylko na liczbach całkowitych.
- `evolves devolves` można użyć tylko w osobnej linijce, a nie w wyrażeniu. Na zmienne tekstowe działa tylko dodawanie, konkatenuje ono oba ciągi znaków. GTL nie zawiera operatorów odejmowania, ani dzielenia. Zamiast tego trzeba dodawać liczbę przeciwną oraz mnożyć przez odwrotność.

Przykład:

```
> see baddie is 9
> taste division is 5 breeding like flipped baddie times
> see subtraction is 7 joined by the literal opposite of baddie
```

- Kolejność wykonywania operatorów:
  1. flipped
  2. the literal opposite of ; breeding like {wyrażenie} times
  3. whatever left from
  4. joined by
  5. {operatory porównania}
  6. not
  7. also
  8. alternatively
- Operatory na tym samym poziomie są wykonywane od lewej do prawej.
- Przed operatorami `also` i `alternatively` może wystąpić nowa linijka.
  Dodatkowo w mnożeniu `breeding like {wyrażenie} times` przed oraz po wyrażeniu może wystąpić nowa linijka.

7 Wyrażenia

- Wyrażenie to:
  - literał
  - zmienna
- Literał to:
  - liczba całkowita (bez znaku)
  - liczba zmiennoprzecinkowa z kropką jako separator części całkowitej od ułamkowej
  - ciąg znaków w podwójnych cudzysłowach
  - :c - oznacza wartość false, c: - oznacza wartość true
- Zmienna to:
  - nazwa zmiennej
  - dostęp do zmiennej struktury
  - dostęp do zmiennej w tablicy
- Ważne! W literałach występują tylko cyfry i kropki, a nazwa jest dowolna. Dlatego `-1000` jest to zmienna o nazwie -1000. Aby zrobić -1000 należy użyć `the literal opposite of 1000`.

8 Warunki *if*

GTL ma bardzo intuicyjnie zrobione warunki:

- `> implying {warunek}` rozpoczyna warunek *if*.
- Następnie podajemy kolejne instrukcje do wykonania w *if*.
- `>or` {warunek} rozpoczyna część *else if*.
- Następnie podajemy kolejne instrukcje do wykonania w *else if*.
- `> or not` rozpoczyna część *else*.
- Następnie podajemy kolejne instrukcje do wykonania w *else*.
- `> or sth` kończy *if*'a.

Składnia if'a:

```
> implying {wyrażenie}
.
.
>or {wyrażenie}
.
.
>or not
.
.
>or sth
```

Przykład:

```
> implying anon beaten by max
> spit max
> or anon beaten by jessica
> spit jessica
> or not
> spit anon
> or sth
```

9 Pętle i iteratory

Aby stworzyć pętle *while* należy użyć poniższej składni:

- `> think that` rozpoczyna pętlę *while*, po której znajduje się warunek.
- Następnie podajemy kolejne linijki zawartości pętli.
- `> reconsider` oznacza konice pętli.

Składnia pętli while:

```
> think that {warunek}
.
.
> reconsider
```

Przykład:

```
> see anon is 0
> see jessica is 6
> think that anon doesn't vibe with jessica
> anon is joined by 2
> spit anon
> reconsider
```

10 Funkcje

- Funkcje są definiowane za pomocą słowa kluczowego `be`.
- Nazwa funkcji musi być nazwą (patrz zmienne).
- Typ zwracany musi być określony w następnej linii. Zmienna przechowująca wartość zwracaną jest wtedy tworzona. Typ jest określany przez podanie typu i dodanie końcówki `int`. Jeśli nie jest określony typ zwracany funkcja niczego nie zwraca.
- Argumenty funkcji są podawane po słowie kluczowym `likes`, a wiele argumentów oddziela się przecinkiem lub słowem `and`. Typ jest określany przez podanie typu i dodanie końcówki `int`. W przypadku pominięcia funkcja nie przyjmuje żadnych argumentów.
- Jeśli chcemy przenieść argumenty przenieść do następnej linijki trzeba zastosować słowo kluczowe `and`. `and` piszemy na początku przeniesionej linijki.
- Przekazywanie przez referencję: przed typem argumentu należy użyć słowa `someone elses`.
- Funkcja musi kończyć się słowem `profit`.
- Funkcja może mieć każdą nazwę (podobnie jak zmienna), z wyjątkiem zarezerwowanej nazwy `me`. Funkcja o nazwie `me` oznacza `main()`, która jest wykonywana domyślnie przy uruchomieniu programu.
- Aby wywołać funkcje trzeba użyć słowa `call nazwa_funkcji`. Aby wywołać funkcję z przypisaniem do zmiennej należy użyć słowa `calling`. Aby wywołać funkcję z argumentami należy po jej nazwie podać słowo `regarding`, po tym rozdzielone przecinkiem lub słowem `and` argumenty.
- Funkcje nie zagnieżdżone istnieją w całym programie. Funkcje zagnieżdżone są dostępne tylko w danym poziomie zagnieżdżenia.
- Możliwe jest przeciążenie funkcji. Deklaracja funkcji o tej samej nazwie ale innych argumetach (inny typ lub różna ilość argumentów).
- Możliwe jest podanie wartości domyślnych parametrów.

Składnia definicji funkcji:

```
> be {nazwa_funkcji}
> {typ_zwracany}ing nazwa_zmiennej
> likes {typ_zmiennej}ing {nazwa_zmiennej}, {typ_zmiennej}ing {nazwa_zmiennej}
> and {typ_zmiennej}ing {nazwa_zmiennej}
> {kod_funkcji}
> profit
```

Składnia wywołania funkcji:

```
> call {nazwa funkcji} regarding {argument}, {argument}
> and {argument}

> {zmienna} is calling {nazwa funkcji} regarding {argument}, {argument}
> and {argument}
```

Przykład:

```
> be Anon
> seeing money
> likes seeing someone elses baddie and spotting math equations
> money is math joined by the literal opposite of baddie
> baddie is joined by 1
> profit

> see baddie is 7
> see Ivone is calling Anon regarding baddie and the literal opposite of 1000
```

Funkcja `Anon` oblicza wartość na podstawie zmiennych `math` i `baddie`, a następnie zwraca wartość przez zmienną `money`.

11 Importowanie z innych plików

- Importowanie odbywa się słowem kluczowym `invite`. Po którym występuje nazwa, zagnieżdżona nazwa lub lista nazwa.
- GTL kolejność szukania użytej nazwy zmiennej:
  1. lokalna historia
  2. globalne historie
  3. zmiennej w invite
  4. pliki o podanej nazwie w `invite` i w tych plikach zaproszone nazwy struktur
  5. katalogi o podanej nazwie w `invite` i powtórzenie kroku poprzedniego
- `invite` może przyjmować listę, wtedy z tego pliku będą pobrane wskazane elementy
- Zaproszone pliki są traktowane jak struktury.
- Interpreter zapamiętuje zaproszone już pliki, unika to błędu cyrkulacji `invite`.
- Zapraszane mogą być pliki, struktury, funkcje lub inne zmienne.

Składnia importu:

```
> invite {zmienna}, {zmienna}
> and {zmienna}
> invite {plik}'s {zmienna}, {zmienna}
```

Przykład:

```
> invite basia's cake, fake
> and bake
> invite baker's bread's flour
```

12 Wypisywwanie i wpisywanie

GTL bardzo prosto robi wczytywanie i wypisywanie danych. Stosując analogię do C++:

$$\begin{array}{ccc} \text{cout} & \rightarrow & \text{spit} \\ \text{cin} & \rightarrow & \text{swallow} \end{array}$$

Składnia:

```
> see anon
> swallow anon
> spit anon
```

Przykład:

```
> see anon
> swallow anon
> spit anon
```

13 Przykładowe programy

Przykład Hello World:

```
Hello World
 > be me
 > spit "Hello, world!"
 > profit
```

Przykład fibbonaci:

```
This program spits out given ammount of numbers of the fibonacci sequence
 >be fibonacci
 > like anon
 > see apple is 1
 > see pear is 1
 > see brother is 0
 > think that anon beats brother
 > parity is brother whatever left from 2
 > implying parity
 > spit apple
 > apple is joined by pear
 > or not
 > spit pear
 > pear is joined by apple
 > or sth
 > brother evolves
 > reconsider
 > profit

 > be me
 > see something
 > swallow something
 > call fibonacci regarding something
 > profit
```