Lipski Kamil Krzysztof
nr indeksu: 283689

Dokumentacja projektu

Język umożliwia podstawowe przetwarzanie zmiennych zawierających wartości liczbowe z jednostkami SI z zakresu dwóch działów fizyki: kinematyki i dynamiki. Jednostki oraz ich relacje są predefiniowane. Oprócz podstawowych zastosowań (tj. obliczania wartości wyrażeń fizycznych), język ułatwia określanie poprawności wykonanych operacji (co zostało uzyskane właśnie przez wdrożenie do języka jednostek i ich relacji – obliczenia są wykonywane nie tylko na wartościach zmiennych, ale też na jednostkach sprowadzanych do kombinacji jednostek podstawowych, tj. metrów, kilogramów i sekund).

Elementy wspierane przez język:

- instrukcja warunkowa typu `if - else`
- pętla typu `while`
- operatory: `+, -, *, /, (), !, &, |, ==, !=, >, >=, <, <=`
- definiowanie funkcji (bez możliwości rozdzielenia deklaracji i definicji)
- zmienne (w tym zmienne lokalne)
- typy: zmienna całkowitoliczbowa z jednostką lub bez, typ napisowy (nie w pełni wspierany, tylko do `print()`)
- stałe

Jednostki

Nazwa jednostki	Jednostka (dostępna dla użytkownika)	Co reprezentuje jednostka?
metr	m	długość
kilogram	kg	masa
sekunda	s	czas
herc	Hz	częstotliwość / prędkość kątowa
niuton	N	siła
dżul	J	energia / praca / moment siły
wat	W	moc
-	m/s	prędkość
-	m/s^2	przyspieszenie
niutonosekunda	N*s	pęd

niuton metr sekunda / dżul sekunda	J*s	moment pędu
-	kg*m^2	moment bezwładności

Program składa się z trzech modułów: leksera, parsera i drzewa AST. Najpierw jest wykonywana analiza leksykalna, następnie składniowa wraz z tworzeniem drzewa AST, a na koniec drzewo (czyli przetworzony dostarczony kod, w postaci gotowej do realizacji) zostaje wykonane i jest zwracany wynik.

W czasie wykonywania drzewa AST pomiędzy poszczególnymi jego węzłami jest przekazywany kontekst (specjalny obiekt), czyli wszędzie potrzebne listy: funkcji, zmiennych lokalnych, globalnych, ale również zmienne poboczne, określające np. czy znajdujemy się w pętli (zapobieżenie ponownej deklaracji zmiennych w instrukcji `while`).

Wspomniane listy funkcji i zmiennych są tworzone na bieżąco w trakcie wykonania drzewa, co pozwala na poprawną obsługę odwołań / wywołań. Z tego właśnie powodu listy te zawierają specjalnie tworzone obiekty zmiennych i funkcji nie znajdujące się w drzewie – w nim umieszczone są węzły odpowiadające faktowi deklaracji zmiennej / funkcji, a ich wykonanie kończy się dodaniem odpowiedniego tworu do którejś z list.

Opis funkcjonalny

- **JEDNOSTKI**
  - użytkownik może używać jedynie wymienionych wyżej jednostek – podanie innej będzie skutkować błędem parsowania,
  - jednostki są rozkładane na jednostki podstawowe (metry, kilogramy i sekundy). W tym celu została utworzona odpowiednia struktura (`UnitDistribution`), która przechowuje w polach krotność poszczególnych jednostek podstawowych (np. „m/s”, to struktura postaci: `{kg = 0, m = 1, s = -1}`),
  - w czasie działań na jednostkach jest wykonywana odpowiednia arytmetyka: dodawanie i odejmowanie uda się jedynie wtedy, gdy jednostki obu składników są takie same, podobnie dla operatorów relacyjnych. Mnożenie i dzielenie to odpowiednio sumowanie i odejmowanie poszczególnych składowych rozkładu jednostki. Działania z operatorami relacyjnymi powodują powstanie stałej odpowiadającej typowi `bool`, bez jednostki, z wartością liczbową 0 lub 1,
  - jednostka raz przypisana do zmiennej nie może być zmieniona (wyjątek: deklaracja bez definicji, wtedy jest ustawiana odpowiednia flaga),
  - zwrócenie / wypisanie jednostki uda się jedynie wtedy, gdy jest ona w zbiorze jednostek zdefiniowanych, czyli na przestrzeni programu mogą powstawać nowe jednostki (odpowiednia flaga), ale nie mogą znaleźć się one na wyjściu.
- **ZMIENNE/STAŁE**
  - zmienne mogą mieć przypisaną jednostkę lub nie,
  - w języku można używać zarówno zmiennych globalnych, jak i lokalnych,
  - zmienne lokalne przesłaniają globalne,
  - zmienne lokalne mogą być deklarowane w funkcjach bądź instrukcjach warunkowych,
  - nie można ponownie deklarować zmiennych globalnych (w głównym toku wykonania programu) lub zmiennych lokalnych (w tym samym podbloku),

- argumenty funkcji są traktowane jako zmienne lokalne, indywidualne dla każdego wywołania funkcji,
- jeśli zmienna zostaje zadeklarowana wewnątrz pętli `while`, kolejne iteracje nie skutkują próbą ponownej deklaracji tej zmiennej – jest jej jedynie przypisywana wartość podana przy deklaracji tej zmiennej (jeśli taka została podana),
- w zagnieżdżonych instrukcjach warunkowych zmienna zadeklarowana w instrukcji nadrzędnej nie może być ponownie zadeklarowana i przesłoniona w instrukcji podrzędnej,
- zmienne niezainicjalizowane mają przypisany tak zwany „null object", który pozwala rozwiązywać konflikty wynikające z braku przypisania wartości do zmiennej; taki sam mechanizm został zastosowany przy funkcjach, które nie zwracają żadnej wartości,
- została zdefiniowana specjalna struktura dla stałych – jest ona polem w klasie zmiennej, ona też może być wspomnianym „null object"; wprowadzenie tej struktury jest spowodowane koniecznością obsługi stałych w czasie działania programu (wyniki obliczeń, zwracanie wartości).

- **FUNKCJE**
  - mogą zwracać wartość lub nie,
  - parametry nie mają przypisanych jednostek – dopiero przy wywołaniu funkcji są im przypisywane wyliczone wartości argumentów wraz z jednostkami,
  - funkcje nie zmieniają wartości zmiennych podanych jako argumenty,
  - można zdefiniować wiele funkcji o tym samym identyfikatorze, ale muszą różnić się ilością parametrów,
  - parametry nie mają wartości domyślnych.

Błędy w kodzie wykrywane na poszczególnych etapach działania interpretera powodują zakończenie jego działania i wypisanie odpowiedniego komunikatu użytkownikowi (w przypadku leksera, parsera i drzewa AST (na poziomie jego budowy) wraz ze wskazaniem miejsca w kodzie, w którym pojawił się błąd, natomiast zawsze z podaniem kontekstu, to jest fragmentu kodu z błędem).

Stworzony język jest językiem wsadowym. Jego interpreter został napisany w języku C++. Jest on uruchamiany za pośrednictwem terminala przez podanie nazwy i pliku wejściowego zawierającego kod do interpretacji, tj. poleceniem:

```
./interpreter PLIK_WEJSCIOWY [>PLIK_WYJSCIOWY]
```

Na wyjście (terminal bądź dookreślony przez użytkownika plik) są przekazywane ciągi znaków wypisywane przez funkcję `print()` + to, co zostanie zwrócone za pomocą polecenia `return` w głównym toku wykonania programu.

Dla każdego z modułów zostały przygotowane testy jednostkowe, które sprawdzają ich podstawowe funkcjonalności: w lekserze – poprawność generowanych tokenów oraz rozpoznawania i reakcji na błędy, w parserze – czy właściwie zostały rozpoznane poszczególne konstrukcje (możliwe dzięki zaimplementowanej opcji konwersji drzewa AST na stringa), w drzewie AST – czy jest ono poprawnie konstruowane (testowanie każdego typu węzła).

Działanie całości programu zostało przetestowane na kilku reprezentatywnych przypadkach (przykładowych programach) – w tym przypadku trudno byłoby skonstruować testy jednostkowe, ze względu na powstające w czasie wykonywania drzewa powiązania między jego elementami oraz złożoność samej procedury.