Zadanie 13B (8 pkt)

Zadanie polega na implementacji interpretera prostego języka programowania niskiego poziomu operującego na stosie. Dostępne są tylko liczby typu `float`. Możliwe jest wykonywanie na nich 4 operacji arytmetycznych (+ - * /) oraz tworzenie i usuwanie zmiennych.

Język inspirowany jest językiem CIL (Common Intermediate Language) – wszystkie operacje wykonywane są na stosie. Przed wykonaniem operacji (instrukcji programu) wymagane jest umieszczenie wszystkich niezbędnych danych na szczycie stosu. Następnie wartości te są zdejmowane ze stosu, wykonywana jest operacja, a jej wynik zostaje z powrotem umieszczony na szczycie stosu. W pliku `Operation.hpp` znajduje się enum `OperationTypes` zawierające wszystkie operacje możliwe do wykonania przez interpreter.

Funkcja `main` zawarta w pliku `Main.cpp` jest już w pełni zaimplementowania i nie można jej modyfikować z wyjątkiem odkomentowania kolejnych etapów zadania. Cała funkcjonalność zadania powinna zostać zaimplementowana w klasie `Interpreter`.

Etap 1 (2 pkt)

Należy dodać prywatne pole typu `std::stack` parametryzowane typem `std::string`. Pole to będzie reprezentować stos na którym będą przeprowadzane wszystkie operacje.

Zaimplementować metody:

- `float PopValue()` – ściąga i zwraca wartość ze szczytu stosu przekonwertowaną na `float`. W celu konwersji można użyć metody `StringToFloat` zaimplementowanej w klasie `Interpreter`.
- `void PushValue(float value)` – wrzuca wartość przekonwertowaną na `std::string` na szczyt stosu. Do konwersji można użyć metody `FloatToString` zaimplementowanej w klasie `Interpreter`.
- `void Push(const std::string& param)` – wrzuca wartość na stos.
- `void Add()` – ściąga 2 wartości ze stosu, dodaje je i zapisuje wynik na stosie (do interakcji ze stosem należy używać `PopValue` i `PushValue`).
- `void Subtract()` – ściąga 2 wartości ze stosu, odejmuje od pierwszej drugą i zapisuje wynik na stosie.
- `void Multiply()` – ściąga 2 wartości ze stosu, mnoży je i zapisuje wynik na stosie.
- `void Divide()` – ściąga 2 wartości ze stosu, dzieli pierwszą przez drugą i zapisuje wynik na stosie.
- `void PrintDataStack(std::ostream& out) const` – wypisuje całą zawartość stosu do podanego strumienia w kolejności od szczytu do dna (niezbędne będzie wykonanie kopii stosu).

Etap 2 (2 pkt)

Należy dodać prywatne pole typu `std::multimap` trzymające wszystkie utworzone zmienne. Klucze są typem `std::string` (nazwa zmiennej), natomiast wartości typem `float` (wartość zmiennej).

Zaimplementować metody:

- `std::string PopName()` – ściąga i zwraca string ze szczytu stosu.

- `float PopValue()` – należy zmodyfikować metodę w taki sposób, że jeśli na szczycie stosu jest nazwa zmiennej to należy ściągnąć ją ze stosu i zwrócić wartość ostatnio zadeklarowanej zmiennej o tej nazwie (ostatni element z multimapy zmiennych o podanym kluczu). W przeciwnym wypadku metoda działa jak w etapie 1.
- `void CreateVariable()` – ściąga ze stosu nazwę zmiennej, następnie jej wartość i dodaje nową zmienną do multimapy (do interakcji ze stosem należy używać `PopName` i `PopValue`).
- `void RemoveVariable()` – ściąga ze stosu nazwę zmiennej i usuwa z multimapy ostatnią zmienną o pasującym kluczu.
- `void Store()` – ściąga ze stosu nazwę zmiennej, wartość i przypisuje zmiennej pobraną wartość.
- `void Print()` – ściąga wartość ze stosu i drukuje ją na ekran.
- `void PrintVariables(std::ostream& out) const` – wypisuje wszystkie zmienne (nazwa i wartość) do strumienia. Należy użyć funkcji `std::for_each` oraz wyrażenia lambda.

Etap 3 (2 pkt)

Klasa `Operation` z pliku `Operation.hpp` reprezentuje pojedynczą operację programu do wykonania. Zawiera typ instrukcji oraz parametr który jest ustawiany tylko dla operacji Push zawierający wartość lub nazwę do umieszczenia na stosie.

Należy dodać prywatne pole typu `std::vector<Operation>` zawierające listę wszystkich dodanych do wykonania operacji.

Należy dodać prywatne pole `std::queue` będące adapterem kontenera `std::list` zawierającego obiekty typu `Operation`. Kolejka ta będzie zawierać wszystkie pozostałe do wykonania operacje (w odróżnieniu od listy operacji nie ma tutaj już wykonanych instrukcji).

Zaimplementować metody:

- `void SetOperations(const std::vector<Operation>& operations)` – kopiuje podane w argumencie operacje do listy operacji w dodanym polu klasy oraz do dodanej kolejki (kontenery są wcześniej czyszczone).
- `void Execute()` – z kolejki wyjmowana jest pierwsza operacja, która zostaje przekazana do zaimplementowanej metody `ExecuteOperation`. Czynność ta powtarzana jest do chwili wyczyszczenia kolejki. W przypadku wystąpienia standardowego wyjątku `std::exception` należy wyświetlić `"ERROR:"` wraz z wiadomością z wyjątku, oraz przerwać wykonywanie operacji z kolejki.
- `void PrintOperations(std::ostream& out) const` – wypisuje wszystkie operacje z listy do przekazanego strumienia. Należy użyć metody `std::copy` do przekopiowania wartości do strumienia (`std::ostream_iterator`).
- `void PrintOperationsQueue(std::ostream& out) const` – wypisuje do podanego strumienia wszystkie operacje z kolejki.

Etap 4 (2 pkt)

Należy zaimplementować metodę `void ParseOperations(const std::vector<std::pair<std::string, std::string>>& operations)` która dla wektora par stringów będzie parsowała zawarte w nim operacje a następnie ustawi je przy pomocy metody `SetOperations`.

Pierwszym elementem pary jest typ operacji (`"CreateVariable"` dla `OperationTypes::CreateVariable` itd.). Drugi element to parametr i tak samo jak w klasie `Operation` zawiera on nazwę lub wartość do umieszczenia na stosie przez operację Push.

W implementacji nie można używać żadnych napisanych przez siebie pętli ani instrukcji warunkowych.

Do konwersji ze stringa z typem operacji do typu `OperationTypes` należy posłużyć się kontenerem `std::map` z odpowiednimi typami, natomiast do konwersji całego wektora par należy użyć funkcji `std::transform`, iteratora wstawiającego do tymczasowego kontenera operacji oraz wyrażenia lambda.

Wyjście programu:

```
==================================== Etap 1 ====================================
============= Pusty interpreter =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

============= 2 elementy na stosie (7.7 i 12.3) =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:
  0: 7.7
  1: 12.3

============= Suma 7.7 + 12.3 = 20.0 =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:
  0: 20

============= Odjecie wyniku od 2.5 (2.5 - 20 = -17.5) =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:
  0: -17.5

============= Dzielenie i dodawanie (201.5 / 10 + 7.25 = 27.4). Na stosie 27.4 i -
17.5 =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:
  0: 27.4
  1: -17.5

==================================== Etap 2 ====================================
============= Na stosie Num1 i 12.3 =============
VARIABLES:
OPERATIONS:
OPERATIONS QUEUE:
```

```
DATA STACK:
  0: Num1
  1: 12.3

============= Stworzenie zmiennej Num1 = 12.3 =============
VARIABLES:
    Num1 = 12.3
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

============= 2.3 - Num1 = -10.0 =============
VARIABLES:
    Num1 = 12.3
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:
  0: -10

============= Num1 = -10 =============
VARIABLES:
    Num1 = -10
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

============= Num2 = Num1 * 9.87 (Num2 = -98.7) =============
VARIABLES:
    Num1 = -10
    Num2 = -98.7
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

============= Num1 = Num2 (2 zmienne Num1 i 1 Num2) =============
VARIABLES:
    Num1 = -10
    Num1 = -98.7
    Num2 = -98.7
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

============= Wyswietl Num1 (druga) (-90) =============
-90
============= Wyswietl Num1 (pierwsza) (-10) =============
-10
============= 2 zmienne. Num1 = -10, Num2 = -98.7 =============
VARIABLES:
    Num1 = -10
    Num2 = -98.7
OPERATIONS:
OPERATIONS QUEUE:
DATA STACK:

==================================== Etap 3 ====================================
============= Przed wykonaniem =============
VARIABLES:
OPERATIONS:
```

```
    Push: 123.3
    Push: Num1
    CreateVariable
    Push: Num1
    Print
    Push: 7
    Push: Num1
    Add
    Push: Num1
    Store
    Push: Num1
    Print
OPERATIONS QUEUE:
    Push: 123.3
    Push: Num1
    CreateVariable
    Push: Num1
    Print
    Push: 7
    Push: Num1
    Add
    Push: Num1
    Store
    Push: Num1
    Print
DATA STACK:

123.3
130.3
============= Po wykonaniu =============
VARIABLES:
    Num1 = 130.3
OPERATIONS:
    Push: 123.3
    Push: Num1
    CreateVariable
    Push: Num1
    Print
    Push: 7
    Push: Num1
    Add
    Push: Num1
    Store
    Push: Num1
    Print
OPERATIONS QUEUE:
DATA STACK:

====================================== Etap 4 ======================================
============= Blad parsowania =============
ERROR: invalid map<K, T> key
============= Przed wykonaniem =============
VARIABLES:
OPERATIONS:
    Push: 2
    Push: Param3
    CreateVariable
    Push: 3
    Push: Param2
```

```
    CreateVariable
    Push: 4
    Push: Param1
    CreateVariable
    Push: Param1
    Push: Param1
    Multiply
    Push: Param2
    Push: Param2
    Multiply
    Push: Param3
    Push: Param3
    Multiply
    Add
    Add
    Push: Result
    CreateVariable
    Push: Param1
    RemoveVariable
    Push: Param2
    RemoveVariable
    Push: Param3
    RemoveVariable
    Push: Result
    Print
    Push: Param1
    Push: Result
    Multiply
OPERATIONS QUEUE:
    Push: 2
    Push: Param3
    CreateVariable
    Push: 3
    Push: Param2
    CreateVariable
    Push: 4
    Push: Param1
    CreateVariable
    Push: Param1
    Push: Param1
    Multiply
    Push: Param2
    Push: Param2
    Multiply
    Push: Param3
    Push: Param3
    Multiply
    Add
    Add
    Push: Result
    CreateVariable
    Push: Param1
    RemoveVariable
    Push: Param2
    RemoveVariable
    Push: Param3
    RemoveVariable
    Push: Result
    Print
```

```
    Push: Param1
    Push: Result
    Multiply
DATA STACK:

============= Blad "Invalid value" na ostatniej lini (Multiply). Result = 29
=============
29
ERROR:
    Invalid value
VARIABLES:
    Result = 29
OPERATIONS:
    Push: 2
    Push: Param3
    CreateVariable
    Push: 3
    Push: Param2
    CreateVariable
    Push: 4
    Push: Param1
    CreateVariable
    Push: Param1
    Push: Param1
    Multiply
    Push: Param2
    Push: Param2
    Multiply
    Push: Param3
    Push: Param3
    Multiply
    Add
    Add
    Push: Result
    CreateVariable
    Push: Param1
    RemoveVariable
    Push: Param2
    RemoveVariable
    Push: Param3
    RemoveVariable
    Push: Result
    Print
    Push: Param1
    Push: Result
    Multiply
OPERATIONS QUEUE:
    Multiply
DATA STACK:
  0: Param1

============= Po wykonaniu (Result = 29) =============
VARIABLES:
    Result = 29
OPERATIONS:
    Push: 2
    Push: Param3
    CreateVariable
    Push: 3
```

```
    Push: Param2
    CreateVariable
    Push: 4
    Push: Param1
    CreateVariable
    Push: Param1
    Push: Param1
    Multiply
    Push: Param2
    Push: Param2
    Multiply
    Push: Param3
    Push: Param3
    Multiply
    Add
    Add
    Push: Result
    CreateVariable
    Push: Param1
    RemoveVariable
    Push: Param2
    RemoveVariable
    Push: Param3
    RemoveVariable
    Push: Result
    Print
    Push: Param1
    Push: Result
    Multiply
OPERATIONS QUEUE:
    Multiply
DATA STACK:
  0: Param1

Press any key to continue . . .
```