Programowanie i metody numeryczne

Projekt ***Odwrotna notacja polska***

Bartłomiej Zglinicki
Bartlomiej.Zglinicki@fuw.edu.pl

1. Wprowadzenie

Tradycyjny sposób zapisu wyrażeń arytmetycznych nakazuje umieszczać znaki działań dwuargumentowych (np. dodawania, mnożenia) pomiędzy ich argumentami oraz stosować nawiasy w celu zaznaczenia kolejności wykonywania działań. I tak, wartość wyrażenia zapisanego jako

$$5 * (2 + 3)$$

otrzymamy, dodając do siebie liczby 2 i 3, a następnie mnożąc wynik tego dodawania przez 5. Notacja taka jest niezwykle czytelna i wygodna dla człowieka, jednak nie dla komputera. W systemach informatycznych łatwiej jest stosować inną notację, nazywaną *odwrotną notacją polską* (ONP; ang. *Reverse Polish Notation*, RPN), w której symbol operacji jest umieszczany po jej argumentach, a stosowanie nawiasów nie jest konieczne. Na przykład przywołane wyżej wyrażenie, które w notacji tradycyjnej ma postać $5 * (2 + 3)$, w odwrotnej notacji polskiej przyjmuje postać

$$2\ 3 + 5\ *$$

Analiza takiego zapisu jest bardzo prosta. Czytamy wyrażenie od lewej do prawej. Za każdym razem, gdy napotkamy w nim liczbę, zapisujemy ją „na marginesie", gdy zaś spotkamy operator (np. symbol działania arytmetycznego albo funkcji), bierzemy liczby zapisane na marginesie jako ostatnie, w takiej ilości, jakiej wymaga ten operator (np. dla operatorów $+$, $-$, $*$ i $/$, reprezentujących działania dwuargumentowe, będą to dwie liczby, zaś dla operatora sin, reprezentującego funkcję jednej zmiennej – jedna liczba), usuwamy je z marginesu, wykonujemy odpowiednią operację z tymi liczbami jako argumentami, a następnie zapisujemy na marginesie wynik tej operacji. Po przeanalizowaniu w ten sposób całego wyrażenia powinniśmy dostać na marginesie jedną liczbę, będącą wartością tego wyrażenia. Szczegółowy opis przedstawionego tu algorytmu wraz z przykładami można znaleźć m.in. na stronie
`https://pl.wikipedia.org/wiki/Odwrotna_notacja_polska`

Algorytm przekształcenia wyrażenia arytmetycznego zapisanego w notacji tradycyjnej na wyrażenie zapisane w odwrotnej notacji polskiej również jest bardzo prosty. Jego dokładny opis i przykłady jego zastosowania można znaleźć na przykład na stronach
`https://pl.wikipedia.org/wiki/Odwrotna_notacja_polska`
`https://en.wikipedia.org/wiki/Shunting-yard_algorithm` (ang.)

W implementacji obu tych algorytmów przydatna jest struktura danych określana jako *stos* (ang. *stack*). Cechuje ją niezwykła prostota, na stosie można bowiem wykonywać zasadniczo tylko dwie operacje: odłożenie nowego elementu na stos oraz zdjęcie ze stosu ostatnio odłożonego elementu. Biblioteka standardowa języka C++ posiada implementację stosu – typ `std::stack` z pliku nagłówkowego `stack`.

Implementując algorytm konwersji notacji tradycyjnej na odwrotną notację polską warto również wykorzystać strukturę danych nazywaną *kolejką* (ang. *queue*). Jest ona niemal identyczna jak stos, różni się od niego tylko tym, że z kolejki zdejmowany jest element odłożony na nią jako pierwszy, a nie jako ostatni. W bibliotece standardowej języka C++, w pliku nagłówkowym `queue` zdefiniowany jest typ `std::queue` stanowiący implementację kolejki.

2. Opis projektu

Celem projektu jest napisanie programu `calc` będącego prostym kalkulatorem działającym w oparciu o odwrotną notację polską. Program powinien umożliwiać przekształcenie wyrażenia zapisanego w notacji tradycyjnej na wyrażenie przedstawione w odwrotnej notacji polskiej oraz obliczenie jego wartości. Obsługiwane przez kalkulator wyrażenia arytmetyczne w notacji tradycyjnej mogą zawierać:

- liczby zmiennoprzecinkowe z kropką (`.`) jako separatorem dziesiętnym; w przypadku liczb całkowitych separator może zostać pominięty (np. zamiast `9.0` można napisać `9`); załóż, że wszystkie liczby występujące w obsługiwanych wyrażeniach arytmetycznych mogą być reprezentowane przez typ `double`,
- pięć podstawowych działań arytmetycznych: dodawanie (`+`), odejmowanie (`-`), mnożenie (`*`), dzielenie (`/`) i potęgowanie (`^`); mnożenie zawsze musi być reprezentowane znakiem `*` (np. poprawnym wyrażeniem jest `3 * exp 5`, nie jest nim zaś `3 exp 5`),
- funkcję wykładniczą $f(x) = e^x$, reprezentowaną jako jednoargumentowy operator `exp`,
- wyłącznie nawiasy okrągłe `(` i `)` (np. poprawnym wyrażeniem jest `2.5 * ((1 + 2) * exp(2 + 5))`, nie jest nim zaś `2.5 * [(1 + 2) * exp(2 + 5)]`).

Liczby, operatory i nawiasy występujące w wyrażeniu mogą, ale nie muszą być rozdzielone znakiem spacji, dopuszczalne są zatem na przykład następujące wyrażenia:

`5 * (2 + 3) * exp(1 + 2)`, `5 * ( 2 + 3 ) * exp ( 1 + 2 )`, `5*(2+3)*exp(1+2)`.

Załóż, że podane na wejściu wyrażenia arytmetyczne są poprawne, a więc mają sens matematyczny oraz zawierają wyłącznie dozwolone elementy.

Program `calc` powinien zawierać pomocniczą klasę `Expression`.

2.1. Klasa `Expression`

Klasa `Expression` ma reprezentować wyrażenie arytmetyczne.

W klasie tej zaimplementuj:

- konstruktor jednoargumentowy, przyjmujący jako argument łańcuch tekstowy typu `std::string` zawierający wyrażenie arytmetyczne, które ma być reprezentowane przez instancję klasy, zapisane w tradycyjnej notacji (np. `(1.3 + 4) / 2`),
- operator wywołania `()`, nie przyjmujący żadnych argumentów i zwracający liczbę typu `double` stanowiącą wartość wyrażenia arytmetycznego reprezentowanego przez instancję klasy; obliczając wartość wyrażenia należy je najpierw zapisać w odwrotnej notacji polskiej; na przykład kod

  ```
  Expression e("(1.3 + 4) / 2");
  auto value = e();
  ```

  powinien tworzyć zmienną `value` typu `double` o wartości `2.65`,
- publiczną metodę `RPN`, nie przyjmującą żadnych argumentów i zwracającą łańcuch tekstowy typu `std::string` przedstawiający reprezentowane przez instancję klasy wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej; na przykład kod

  ```
  Expression e("(1.3 + 4) / 2");
  auto rpn = e.RPN();
  ```

  powinien tworzyć zmienną `rpn` typu `std::string` o wartości `"1.3 4 + 2 /"`,

- operator `<<`, wypisujący do strumienia wyjściowego typu `std::ostream` wartość wyrażenia arytmetycznego reprezentowanego przez instancję klasy; na przykład kod

  ```
  Expression e("(1.3 + 4) / 2");
  std::cout << e << std::endl;
  ```

  powinien wypisywać na standardowe wyjście `2.65`.

Kod źródłowy klasy `Expression` powinien zostać umieszczony w osobnych plikach: deklaracja klasy – w pliku nagłówkowym `Expression.hpp`, zaś definicja jej metod – w pliku `Expression.cpp`.

2.2. Program `calc`

Program `calc` po uruchomieniu powinien wypisać na standardowe wyjście znak zachęty (np. `>`) i oczekiwać, aż na standardowym wejściu pojawi się wyrażenie arytmetyczne zapisane w tradycyjnej notacji. Gdy użytkownik wpisze wyrażenie i zatwierdzi je klawiszem `[Enter]`, program ma wypisać jego wartość na standardowe wyjście, a następnie ponownie wyświetlić znak zachęty, oczekując na kolejne wyrażenie do obliczenia. W przypadku, gdy wyrażenie zostanie poprzedzone znakami `RPN` i spacją, zamiast jego wartości program powinien wypisać na standardowe wyjście to samo wyrażenie zapisane w odwrotnej notacji polskiej. Jeśli użytkownik zamiast wyrażenia wpisze `exit` lub `quit`, program powinien zakończyć działanie.

Obliczanie wartości wyrażeń arytmetycznych oraz zapisywanie ich w odwrotnej notacji polskiej powinno być realizowane z wykorzystaniem klasy `Expression`.

Przykładowe wykonanie
Wywołanie:
Windows: `calc.exe`
macOS / Linux: `./calc`
Wyjście
`>`
Wejście
`5 * (2 + 3) [Enter]`
Wyjście
`25`
`>`
Wejście
`(1.3 + 4) / 2 [Enter]`
Wyjście
`2.65`
`>`
Wejście
`RPN (1.3 + 4) / 2 [Enter]`
Wyjście
`1.3 4 + 2 /`
`>`
Wejście
`exit [Enter]`

3. Co dalej?

Jeśli tematyka projektu Cię zainteresowała, możesz w przyszłości rozbudować napisany przez siebie program, dodając obsługę większej ilości operacji matematycznych (funkcje trygonometryczne, hiperboliczne

itd.), a także dodatkowe funkcjonalności, np. rysowanie wykresów funkcji i zapisywanie ich w plikach graficznych (można tu wykorzystać jedną z wielu zewnętrznych bibliotek), numeryczne różniczkowanie i całkowanie, historię wyrażeń, pamięć itp. Możesz również pokusić się o napisanie programu w wersji z graficznym interfejsem użytkownika, posługując się jedną z przeznaczonych do tego bibliotek (np. Qt, Dear ImGui, WinUI).