Zadanie 7+8 23-30 kwietnia 2025 r.

kurs języka C++

wyrażenia i instrukcje

Instytut Informatyki
Uniwersytetu Wrocławskiego Paweł Rzechonek

Prolog

Wyrażenia arytmetyczne mają fundamentalne znaczenie w każdym języku programowania – są to dowolne wyrażenia typu liczbowego złożone z liczb, zmiennych, funkcji, operatorów i nawiasów. Wyrażenia arytmetyczne nie stanowią samoistnych instrukcji ale są ich ważną częścią.

Każde wyrażenie można przedstawić w postaci drzewa (nie zawsze binarnego), w którym liście reprezentują argumenty a węzły wewnętrzne operacje arytmetyczne (operacje unarne i binarne) albo funkcje matematyczne o dowolnej arności (dlatego drzewa wyrażeń nie zawsze są drzewami binarnymi). Argumentami operatorów albo funkcji w takim drzewie są inne wyrażenia (poddrzewa wyrażeń). Sposób łączenia ze sobą poddrzew w jedno wyrażenie determinują priorytety i łączność operatorów oraz rozstawienie nawiasów (operacja wykonująca się na samym końcu w trakcie obliczania wyrażenia zostanie umieszczona w korzeniu drzewa).

Zadanie 1

Zdefiniuj abstrakcyjną klasę bazową `wyrazenie`, reprezentującą całkowitoliczbowe wyrażenie arytmetyczne. W klasie tej umieść deklaracje abstrakcyjnych metod `oblicz()` oraz `zapis()`. Metoda `oblicz()` doprecyzowana w klasach potomnych będzie obliczać wartość wyrażenia i zwracać wynik typu `int`; metoda `zapis()` ma zwracać napis typu `string` reprezentujący całe wyrażenie wraz z dopisanymi niezbędnymi nawiasami – należy przy tym uwzględnić priorytety operatorów (na przykład priorytet mnożenia jest wyższy niż priorytet dodawania) oraz ich łączność (na przykład mnożenie jest lewostronnie łączne a potęgowanie jest łączne prawostronnie).

W następnej kolejności zdefiniuj cała hierarchię klas dziedziczących po klasie `wyrazenie`, które będą reprezentowały elementy drzewa wyrażenia – operatory jako węzły wewnętrzne i operandy jako liście. Dobrze przemyśl projekt tej hierarchii, aby w klasach potomnych dokładać jak najmniej składowych i maksymalnie ograniczyć modyfikację funkcjonalności.

Operandy to liczby, zmienne i stałe. Klasa `liczba` ma reprezentować liczbę całkowitą użytą w wyrażeniu (opakowanie na wartość typu `int`). Klasa `stala` ma reprezentować określoną wartość poprzez nazwę mnemoniczną typu `string`; zdefiniuj co najmniej dwie stałe (dziedziczące po abstrakcyjnej klasie `stala`): klasę `zero` jako odpowiednik wartości 0 oraz klasę `jeden` jako odpowiednik wartości 1. Klasa zmienna to nazwany schowek na liczbę (zmienna

ma mieć określoną nazwę typu `string`, przez którą będzie można odwołać się do zbioru zmiennych (nazw skojarzonych z obiektami typu `int`); wartość zmiennej można wykorzystać do obliczenia jakiegoś wyrażenia, ale zmienną można też ustawić, wpisując do niej nową wartość. Zmienne pamiętaj w zbiorze asocjacyjnym typu `vector<pair<string,int>>`. Zbiór ten umieść jako prywatne pole statyczne w klasie `zmienna` i dopisz kilka publicznych statycznych metod pozwalających zarządzać tym zbiorem (dodawanie, usuwanie i modyfikacja zmiennych).

Operatory natomiast reprezentują podstawowe operacje arytmetyczne unarne (klasa `operator1`) i binarne (klasa `operator2` dziedzicząca po `operator1`). Operator unarny to przede wszystkim klasa `minus` do zmiany znaku na przeciwny (można też zdefiniować inne operatory unarne, jak wartość bezwzględna `abs` czy informacja o znaku `sgn`).

Operatory binarne dotyczą operacji arytmetycznych (dodawanie, odejmowanie, mnożenie, dzielenie całkowite, reszta z dzielenia, potęgowanie i logarytm dyskretny) oraz operacji relacyjnych (mniejsze, większe, mniejsze-równe, większe-równe, równe i różne), których wynikiem jest zero/fałsz albo jeden/prawda (można też zdefiniować inne operatory binarne, jak minimum `min` czy maksimum `max`).

Pamiętaj o arności operatorów, ich priorytetach i łączności w kontekście prezentacji wyrażeń za pomocą funkcji `zapis()`. Operatory unarne mają najwyższy priorytet i przeważnie są prefiksowe. Operatory binarne są lewostronnie łączne za wyjątkiem potęgowania (na przykład 2^{2^k}, co zapiszemy tekstowo jako 2^2^k, będzie obliczone od prawej strony 2^(2^k), przy czym symbol ^ oznacza potęgowanie) oraz logarytmu dyskretnego (na przykład $\log_2 \log_2 k$, co zapiszemy tekstowo jako 2_2_k, będzie obliczone od prawej strony 2_(2_k), przy czym symbol _ oznacza logarytmowanie). Co się tyczy priorytetów operacji binarnych, to najwyższy priorytet mają potęgowanie i logarytmowanie, potem jest mnożenie, dzielenie i reszta z dzielenia, dalej będzie dodawanie i odejmowanie a najniższy priorytet mają operacje relacyjne.

Definicje wszystkich tych klas umieść w przestrzeni nazw `obliczenia`. Uzupełnij to zadanie o program testowy napisany poza przestrzenią nazw `obliczenia` – program ma rzetelnie sprawdzić działanie obiektów reprezentujących wyrażenie arytmetyczne. W programie testowym skonstruuj różne drzewa obliczeń, wypisz każde z nich posługując się metodą `zapis()` a potem oblicz i wypisz ich wartości używając metody `oblicz()`. Na przykład wyrażenie `2 ^ (x / 3 – 1)` należy zdefiniować następująco:

```
Wyrazenie w = new potega(
    new liczba(2),
    new odejmowanie(
        new dzielenie(
            new zmienna("x"),
            new liczba(3)
        )
    )
);
```

Zadanie 2

Zdefiniuj abstrakcyjną klasę bazową `instrukcja`, reprezentującą wykonanie jakiejś instrukcji w programie. W klasie tej umieść deklaracje abstrakcyjnych metod `wykonaj()` oraz `zapis()`. Metoda `wykonaj()` doprecyzowana w klasach potomnych będzie realizować określoną czynność obliczeniową; metoda `zapis()` ma zwracać napis typu `string` reprezentujący program zapisany w postaci ciągu instrukcji wraz z dopisanymi wcięciami, aby zwiększyć czytelność programu.

W następnej kolejności zdefiniuj zbiór klas dziedziczących po klasie `instrukcja`, które będą reprezentowały instrukcje proste i strukturalne w programie: klasa `deklaracja` dla deklaracji zmiennej (wszystkie zmienne inicjalizuj domyślnie wartością 0), klasa `przypisanie` dla instrukcji przypisania wartości obliczonego wyrażenia typu `wyrazenie` (z poprzedniego zadania) do wskazanej zmiennej typu `zmienna` (z poprzedniego zadania), klasa `blok` dla instrukcji blokowej, klasy dla instrukcji warunkowych (takich jak instrukcje *if* oraz *if-else*), klasy dla instrukcji pętli (takie jak instrukcje *while* oraz *do-while*), klasę `czytanie` dla instrukcji odczytującej liczbę całkowitą ze standardowego wejścia i umieszczającej ją w określonej zmiennej oraz klasę `pisanie` dla instrukcji wypisującej na standardowe wyjście obliczoną wartość zadanego wyrażenia.

Warunek w instrukcjach warunkowych lub w pętlach jest wyrażeniem – warunek jest prawdziwy wtedy i tylko wtedy, gdy wartością wyrażenia jest liczba różna od 0. Instrukcja blokowa powinna być inicjalizowana dowolną liczbą instrukcji wewnętrznych (konstruktor z listą wartości `initializer_list<instrukcja>`); poza tym instrukcja ta ma spamiętywać wszystkie zmienne, które zostały utworzone w tym bloku i na końcu ma je usunąć (nie wolno tworzyć zmiennych o takich samych nazwach). Konstruktory klas reprezentujących różne instrukcje powinny sprawdzać, czy ich argumenty są równe `nullptr`, a jeśli tak to należy zgłosić odpowiedni wyjątek.

Definicje wszystkich tych klas umieść w przestrzeni nazw `obliczenia`. Uzupełnij to zadanie o program testowy napisany poza przestrzenią nazw `obliczenia` – program ma rzetelnie sprawdzić działanie obiektów reprezentujących instrukcje proste i strukturalne. W programie testowym stwórz programu (ciąg instrukcji) wykonujący test pierwszości. Program ten może działać według następującego schematu:

```
var n;
read n;
if (n < 2) write 0;
else
{
    var p;
    p = 2;
    var wyn;
    while (p * p <= n)
    {
        if (n % p == 0)
        {
```

```
            wyn = p;
            p = n;
        }
        p = p + 1;
    }
    if (wyn > 0) write 0;
    else write 1;
}
```

Uwaga

Podziel program na pliki nagłówkowe (definicje klas) i źródłowe (definicje funkcji składowych zadeklarowanych w klasach). Funkcję `main()` z testami umieszczaj w osobnym pliku.

Ważne elementy programu

- Optymalna hierarchia klas pozwalająca definiować różne elementy wyrażenia; na szczycie tej hierarchii ma się znaleźć abstrakcyjna klasa `wyrazenie` z czysto wirtualnymi metodami abstrakcyjnymi `oblicz()` i `zapis()`.
- Nadpisanie metod `oblicz()` i `zapis()` w klasach potomnych.
- Wykorzystanie priorytetów operatorów do zminimalizowania liczby wypisywanych nawiasów przez metodę `zapis()`.
- Hierarchia klas pozwalająca definiować różne instrukcje proste i strukturalne; na szczycie tej hierarchii ma się znaleźć abstrakcyjna klasa `instrukcja` z czysto wirtualnymi metodami abstrakcyjnymi `wykonaj()` i `zapis()`.
- Nadpisanie metod `wykonaj()` i `zapis()` w klasach potomnych.
- Wykorzystanie informacji o zagnieżdżeniu instrukcji do czytelnego wypisanie programu z wcięciami przez metodę `zapis()`.
- Zablokowanie kopiowania i przenoszenia dla wyrażeń i instrukcji.
- Zgłaszanie wyjątków w konstruktorach i funkcjach składowych.
- W funkcji `main()` należy przetestować obiekty wszystkich klas nieabstrakcyjnych.