

ZAAWANSOWANE PROGRAMOWANIE KOMPUTEROWE WNE (2023)

KRZYSZTOF ZIEMIAŃSKI

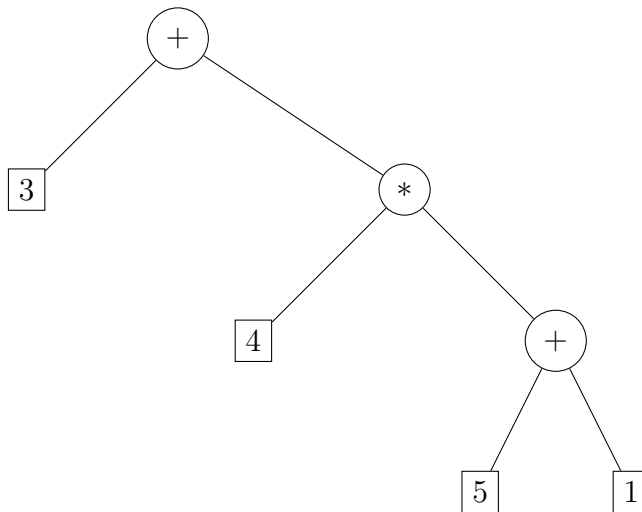
9. WYRAŻENIA ARYTMETYCZNE

Celem tych zajęć będzie stworzenie prostego kalkulatora, który po wczytaniu tekstu zawierającego wyrażenie arytmetyczne obliczy jego wartość. Np. po wpisaniu

$$3+4*(5+1)$$

otrzymamy wartość 27. Kalkulator będzie operował na liczbach całkowitych nieujemnych i obsługiwał działania $+$, $*$ oraz nawiasy. Wyliczanie odbywa się w dwóch krokach. Najpierw tworzymy reprezentację wyrażenia, a potem je wyliczamy korzystając z tej reprezentacji.

9.1. **Reprezentacja wyrażień.** Wyrażenia będą reprezentowane za pomocą drzew. Przykładowe wyrażenie podane powyżej będzie wyglądało tak:



Jak widzimy będziemy potrzebować trzech rodzajów węzłów:

- stałych liczbowych,
- sum,
- iloczynów.

Poszczególnym rodzajom węzłów będą odpowiadać różne klasy; z drugiej strony jednak wszystkie te rodzaje reprezentują wyrażenia. Zdefiniujemy je więc jako klasy pochodne klasy reprezentującej wszystkie wyrażenia.

9.2. **Dziedziczenie — klasy pochodne.** Jeśli A jest pewną klasą, to możemy zdefiniować jej klasę pochodną B:

```
class B: public A {
    // deklaracje pól i metod
};
```

Obiekt klasy B będzie zawierał wszystkie pola zadeklarowane w klasie A i ponadto pola zadeklarowane w klasie B. Podobnie jest z metodami: możemy używać zarówno metod zadeklarowanych z klasy A jak i tych z klasy B. Najważniejszym mechanizmem związanym z dziedziczeniem klas jest możliwość ponownego zdefiniowania funkcji w klasie pochodnej. Wtedy działanie tej funkcji będzie zależało od rzeczywistego typu obiektu; dla obiektów typu A funkcja będzie działała tak jak zdefiniowano to w klasie A, a dla obiektów typu B, tak jak w klasie B. Takie funkcje nazywamy wirtualnymi; jeśli chcemy, żeby funkcja była wirtualna, jej deklarację poprzedzamy słowem `virtual`.

W tym przykładzie definiujemy klasę `Punkt`.

```
class Punkt {
public:
    Punkt(double _x, double _y): x(_x), y(_y) {}
    double x;
    double y;
    virtual void drukuj() {
        cout << "(" << x << ", " << y << ")";
    }
};
```

Następnie definiujemy jej klasę pochodną

```
class KolorowyPunkt: public Punkt {
public:
    KolorowyPunkt(double _x, double _y, string kolor)
        : Punkt(_x, _y), kolor(_kolor) {}
    string kolor;
    virtual void drukuj() {
        cout << "(" << x << ", " << y << ") Kolor: " << kolor;
    }
};
```

Proszę zauważyć, że w konstruktorze klasy `KolorowyPunkt` musimy podać parametry do pewnego konstruktora klasy `Punkt`. O oto przykład użycia:

```
Punkt* p=new Punkt(2,5);
p->drukuj(); // drukuje bez koloru
delete p;
p=new KolorowyPunkt(3,7,"czarny");
p->drukuj(); // a teraz z kolorem
delete p;
```

Musimy korzystać ze wskaźników — wskaźnik do typu `Punkt` może wskazywać na `KolorowyPunkt`, ale do statycznej zmiennej typu `Punkt` nie możemy przypisać kolorowego punktu (tj. możemy, ale informacja o kolorze zostanie utracona).

9.3. **Wyrażenia.** Wyrażenia arytmetyczne będą reprezentowane przez klasę

```
class Wyrazenie {
```

```
public:
    virtual int wartosc()=0;
    virtual void drukuj()=0;
};
```

Wyrażenia można drukować i obliczać ich wartość, ale obie te czynności są niezdefiniowane. Nie możemy wydrukować "ogólnego" wyrażenia, bo nie wiem czym ono jest; nie możemy nawet stworzyć obiektu tej klasy. Takie klasy nazywają się klasami abstrakcyjnymi. Można ich użyć tylko do zdefiniowania klas pochodnych. Klasa reprezentująca stałe liczbowe to

```
class Stala: public Wyrazenie {
public:
    Stala(int _a): a(_a) {}
    virtual int wartosc() { return a; }
    virtual void drukuj() { cout << a; }
private:
    int a;
};
```

Obiekty tej klasy możemy już tworzyć, drukować i wyliczać ich wartości (choć nie jest to szczególnie interesujące). Pożytek z klas pochodnych można zauważyć na przykładzie kolejnej klasy reprezentującej sumy wyrażeń:

```
class Suma: public Wyrazenie {
public:
    Suma(Wyrazenie* _w1, Wyrazenie* _w2): w1(_w1), w2(_w2) { }
    virtual ~Suma() { delete w1; delete w2; }
    virtual int wartosc() { return w1->wartosc() + w2->wartosc(); }
    virtual void drukuj() {
        cout << "(";
        w1->drukuj();
        cout << "+";
        w2->drukuj();
        cout << ")";
    }
private:
    Wyrazenie* w1;
    Wyrazenie* w2;
};
```

Obiekt klasy Suma reprezentuje sumę dwóch dowolnych wyrażeń: stałych, sum, iloczynów, a nawet innego rodzaju wyrażeń (wystarczy stworzyć nowe klasy pochodne klasy Wyrazenie). Wartość sumy obliczy się jako suma wartości wyrażeń wskazywanych przez w1 i w2, które to wyrażenia oblicza się z kolei we właściwy dla siebie sposób.

9.4. Strumienie. Wyrażenia arytmetyczne będziemy wczytywać ze strumienia (typu `istream`), przykładem takiego strumienia jest `cin`. Można też wczytywać wyrażenia z napisów typu `string` np. tak

```
string s("2+2*(7+2)");
stringstream ss(s); // tworzymy strumień podłączony do napisu
w = czytajWyrazenie(ss); // używamy ss jak cin
```

Wymaga to załączenia biblioteki `sstream`. Znaki wczytujemy za pomocą metod `get`:

```
f.get(c);
```

wczytuje znak ze strumienia `f` i umieszcza go w zmiennej `c`, oraz z funkcji `peek`:

```
f.peek();
```

zwraca kolejny znak ze strumienia, ale go nie pobiera; funkcja `peek` może zwrócić wartość `EOF`, co oznacza, że w strumieniu nie ma już więcej znaków.

9.5. Gramatyki, czyli jak wczytywać wyrażenia. Poprawne wyrażenia arytmetyczne można opisać za pomocą następującego przepisu (nazywa się to gramatyką):

$$\begin{aligned} \text{Wyrażenie} &\rightarrow \text{Składnik} \mid \text{Składnik} + \text{Wyrażenie} \\ \text{Składnik} &\rightarrow \text{Czynnik} \mid \text{Czynnik} * \text{Składnik} \\ \text{Czynnik} &\rightarrow \text{Stała} \mid (\text{Wyrażenie}) \end{aligned}$$

Inaczej mówiąc, każde wyrażenie jest sumą składników, każdy składnik jest iloczynem czynników, a każdy czynnik jest "atomowym" wyrażeniem: liczbą lub wyrażeniem zamkniętym w nawiasy. Funkcje wczytujące wyrażenia również są zbudowane wg tego schematu:

- Funkcja `czytajWyrażenie` czyta składnik, następnie sprawdza, czy kolejnym znakiem jest `+`; jeśli tak, czyta wyrażenie i zwraca sumę składnika i wyrażenia; jeśli nie, zwraca tylko wyrażenie.
- Funkcja `czytajSkładnik` czyta czynnik, następnie sprawdza, czy kolejnym znakiem jest `*`; jeśli tak, czyta składnik i zwraca iloczyn czynnika i składnika; jeśli nie, zwraca tylko czynnik.
- Funkcja `czytajCzynnik` sprawdza, co jest pierwszym znakiem. Jeśli to nawias, czyta nawias, wyrażenie i nawias zamykający. Jeśli to cyfra, wczytuje liczbę. Jeśli żadne z powyższych, wypisuje się komunikat o błędzie.

Wynikiem działania tych funkcji jest wskaźnik do wczytanego wyrażenia.

9.6. Zadania.

- (1) Zmienić klasy `Suma` i `Iloczyn` tak, aby przechowywały sumy złożone z więcej niż dwóch składników (iloczynu wielu czynników).
- (2) Dodać obsługę liczb ujemnych i odejmowania.