Objektum elvű alkalmazások fejlesztése

Kifejezés lengyel formára hozása és kiértékelése

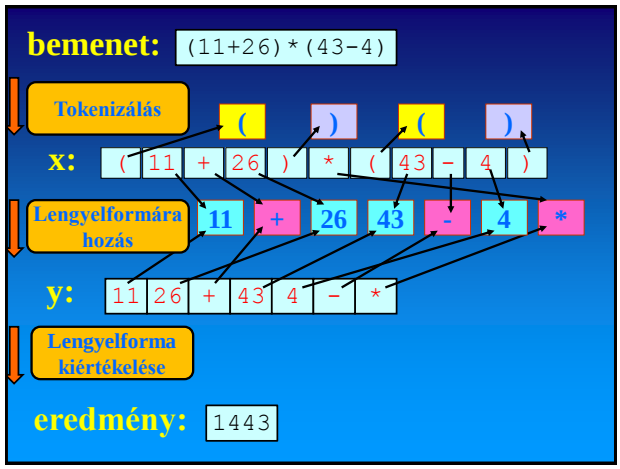
Készítette: Gregorics Tibor
Szabóné Nacsa Rozália

Feladat

Alakítsunk át egy infix formájú aritmetikai kifejezést postfix (lengyel) formájúra, és számoljuk ki az értékét.

(11+26)*(43-4) → **11 26 + 43 4 - *** → **1443**

Mindkét lépéshez egy-egy vermet használunk, az elsőben a műveleti jeleket és a nyitózárójeleket helyezzük el, a másodikban pedig az operandusokat illetve a részeredményeket.
Szükségünk lesz két sorozatra is, amelyekben a kezdeti kifejezés tokenizált formáját, illetve a közbülső (lengyel) formát tároljuk.

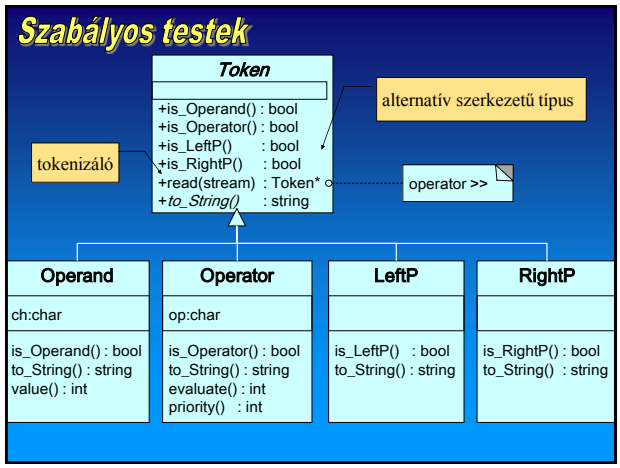


Megoldási terv

1. Tokenizáljuk a karaktersorozatként megadott infix formájú kifejezést és a tokenek címeit elhelyezzük egy *x* sorozatban.
2. Lengyel formára alakítjuk az *x* sorozatbeli kifejezést: azaz a tokenek címét postfix formában soroljuk fel egy *y* sorozatban.
3. Kiértékeljük az *y* sorozatbeli lengyel formát.

Kell:
Különféle tokenek típusai (alternatív szerkezetű típus)
Token és abból származtatott Operandus,
Operator, Balzárójel, Jobbzárójel osztályok
sorozatok típusa: BiQueue<Token*>
veremek típusai: Stack<Token*>, Stack<int>

Szabályos testek



A Token osztály: deklaráció

```
class Token
{
  public:
    class IllegalCharacterException{ ... };

    virtual ~Token();
    virtual bool is_LeftP()    const {return false;}
    virtual bool is_RightP()   const {return false;}
    virtual bool is_Operand()  const {return false;}
    virtual bool is_Operator() const {return false;}
    virtual bool is_End()      const {return false;}

    virtual string to_String()  const = 0;

  friend
    std::istream& operator>>(std::istream&, Token*&);
};
```

felüldefiniálható

specifikációban még nem szerepelt

token.h

Token osztály: deklaráció

```
class Token
{
   ...
   public:
       class IllegalCharacterException{
           private:
               char ch;
           public:
               IllegalCharacterException(char c) : ch(c){}
               char message() const { return ch;}
       };
       ...
};
```

token.h

Token osztály: implementáció

```
istream& operator>> (istream &s, Token* &t){
   char ch;
   s >> ch;
   switch(ch){
      case '+': case '-': case '*': case '/':
                  t = new Operator(ch);     break;
      case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
                  s.putback(ch);
                  int intval;
                  s >> intval;
                  t = new Operand(intval); break;
      case '(':   t = new LeftP();          break;
      case ')':   t = new RightP();         break;
      case ';':   t = new End();            break;
      default:    if(!s.fail())
                    throw new Token::IllegalCharacterException(ch);
   }
   return s;
}
```

Csak előjel nélküli egész számok

vissza a pufferbe

a kifejezést pontosvessző zárja le

token.cpp

Operand osztály: deklaráció

```
class Operand: public Token
{
   public:
      Operand(int v) {val=v;}

      bool is_Operand()   const {return true; }

      string to_String()  const {
         std::ostringstream ss;
         ss << val;
         return ss.str();
      }

      int value() const {return val;}

   protected:
      int val;
};
```

konverzió

token.h

Operator osztály: deklaráció

```
class Operator: public Token
{
   public:
      Operator(char o) {op = o;};

      bool is_Operator()  const {return true; }

      string To_String()  const {
         string ret;
         ret = op;
         return ret;
      }

      int priority()              const;
      int evaluate(int a, int b) const;

   protected:
       char op;
};
```

token.h

Operator osztály: implementáció

```
int Operator::priority() const {
   switch(op){
      case '+': case '-': return 1;
      case '*': case '/': return 2;
      default: return 3;
   }
}

int Operator::evaluate(int a, int b) const {
   switch(op){
      case '+': return a+b;
      case '-': return a-b;
      case '*': return a*b;
      case '/': return a/b;
   }
}
```

token.cpp

RightP, LeftP, End osztály: deklaráció + implementáció

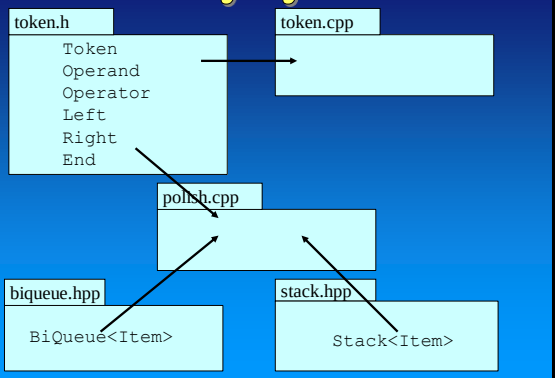
```
class RightP: public Token{
   public:
      bool is_RightP()   const{return true; }
      string to_String() const{return ")"; }
};

class LeftP: public Token{
   public:
      bool is_LeftP()    const {return true; }
      string to_String() const {return "("; }
};

class End: public Token{
   public:
      bool is_End()      const {return true; }
      string to_String() const {return ";"; }
};
```

token.h

Csomag diagram



Kifejezés tokenizálása

```
int main()
{
   cout << "Add meg az aritmetikai kifejezest!\n";

   BiQueue<Token*> x;
   try{
      Token *t;
      cin >> t;
      while(!t->is_End())
         x.hiext(t);
         cin >> t;
      }
   }catch(Token::IllegalCharacterException *ex){
      cout << "Illegális karakter: "
           << ex->message() << endl;
      delete ex;
      deallocateToken(x);
      exit(1);
   }
```

tokent olvas

beolvasás dobja

felszabadítja a beolvasás során végzett helyfoglalásokat

polish.cpp

Tokenek felszabadítása

```
void deallocateToken(BiQueue<Token*> &x)
{
   BiQueue<Token*>::Enumerator itx = x.createEnumerator();

   for(itx.first(); !itx.end(); itx.next()){
      delete itx.current();
   }
}
```

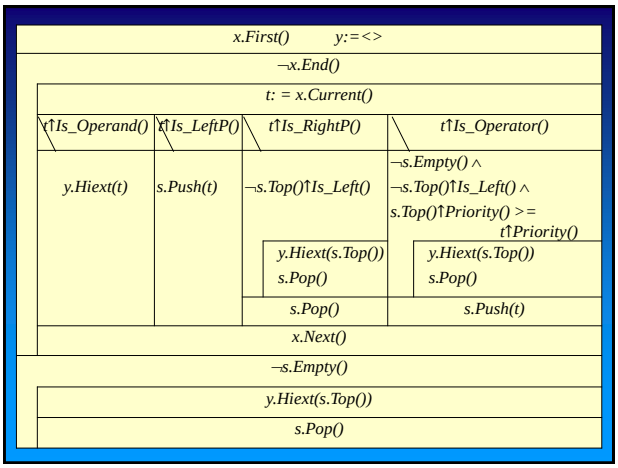
felsoroló

polish.cpp

Lengyel formára hozás

A bemenő sorozat nyitó zárójeleit és műveleti jeleit (ez utóbbiak esetén a precedenciájuk szem előtt tartásával) egy verembe tesszük. Csukó zárójel esetén kiürítjük a verem tartalmát a leg(f)első nyitózárójeléig a kimenő sorozatba. A bemenő sorozat Minden más jelét közvetlenül a kimenő sorozatba másoljuk.

(	11	+	26	)	*	(	43	-	4	)



Lengyel formára hozás

```
BiQueue<Token*> y;
Stack<Token*>   s;
BiQueue<Token*>::Enumerator itx = x.createEnumerator();

for(itx.first(); !itx.end(); itx.next()){
   Token *t = itx.current();

   ... // négy-ágú elágazás ld. következő dián

}

while(!s.empty()){
   if(s.top()->is_LeftP()){
      cout << "Szintaktikai hiba!/n";
      deallocateToken(x);
      exit(1);
   }
   else y.hiext(s.pop());
}
```

felsoroló

hiba lehetőség:
több nyitó zárójel,mint csukó

polish.cpp

```
if(t->is_Operand())        y.hiext(t);
else if (t->is_LeftP())    s.push(t);
else if (t->is_RightP()){
   try{
      while(!s.Top()->is_LeftP()) y.hiext(s.pop());
      s.pop();
   }catch(Stack<Token*>::Exceptions ex){
      if(ex==Stack<Token*>::EMPTYSTACK){
                  cout << "Szintaktikai hiba!/n";
                    deallocateToken(x); exit(1);
         }
   }
}else if (t->is_Operator()) {
   while(!s.empty() && s.top()->is_Operator() &&
           ((Operator*)s.top())->priority()
                        >=((Operator*)t)->priority() )
      y.hiext(s.Pop());
   s.push(t);
}else{
   cout << "Szintaktikai hiba!/n";
   deallocateToken(x); exit(1);
}
```

hiba lehetőség: több a csukó zárójel, mint nyitó

„static casting”: Az `s.top()->priority()` nem jó, mert `Token`-ben nincs `priority()`

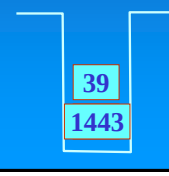
hiba lehetőség: nem tudom mi lehetne

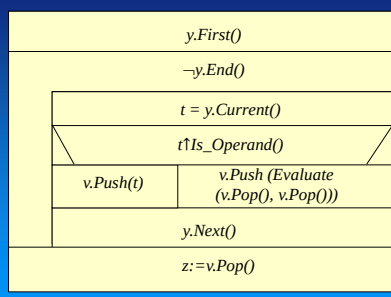
polish.cpp

Lengyel forma kiértékelése

A kiértékeléshez is használjunk egy vermet, ahová az operandusokat illetve az eredményeket tesszük.
A bemenő sorozat operandusait a verembe tesszük, műveleti jel olvasása esetén a verem tetején levő értékeket a kijelölt művelettel feldolgozzuk és az eredményt a verembe tesszük.

11	26	+	43	4	-	*





Lengyel forma kiértékelése

```
try{
   Stack<int> v;
   BiQueue<Token*>::Enumerator ity=y.createEnumerator();
   for(ity.first(); !ity.end(); ity.next()){
      Token *t = ity.current();
      if (t->is_Operand())
         v.push( ((Operand*)t)->value() );
      else
         v.push( ((Operator*)t)->evaluate
                                 (v.pop(), v.pop()) );
   }
   int r = v.pop();
   if(!v.empty()){
      cout << "Szintaktikai hiba!/n";
      deallocateToken(x);
      exit(1);
   }
   cout << "A kifejezés értéke: " << r << endl;
   deallocateToken(x);
}
```

felsoroló

statikus konverzió

statikus konverzió

hiba lehetőség: több operandus

polish.cpp

Lengyel forma kiértékelése

```
try{

   ... // ld. előző dia

}catch(Stack<int>::Exceptions ex){
   if(ex==Stack<Token*>::EMPTYSTACK){
      cout << "Szintaktikai hiba!/n";
      deallocateToken(x);
      exit(1);
   }
}
```

hiba lehetőség: kevés operandus

polish.cpp