laboratorium: zadanie 8 *termin: 18–19 kwietnia 2012 r.*

KURS JĘZYKA C++

DŁUGIE LICZBY

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

Zadanie.

Zdefiniuj klasę `LiczbaCalk` reprezentującą liczbę całkowitą o nieograniczonym zakresie. Liczby te powinny być kodowane binarnie w systemie *uzupełnienia dwójkowego U2*. Do pamiętania długich liczb całkowitych wykorzystaj tablicę bitów z wcześniejszego zadania.

```
class LiczbaCalk
{
protected:
    TabBit *bity;
public:
    LiczbaCalk () throw ();
    LiczbaCalk (int n) throw ();
    LiczbaCalk (const LiczbaCalk &x) throw ();
    virtual ~LiczbaCalk () throw ();
    LiczbaCalk & operator= (const LiczbaCalk &x) throw ();
// ...
};
```

Klasa `LiczbaCalk` powinna być wyposażona w konstruktor domyślny (wartość domyślna nowoutworzonej liczby ma wynosić 0), konstruktor inicjalizowany wartością typu `int`, konstruktor kopiujący, operator przypisania kopiującego i destruktor. Nie zapomnij przy każdej metodzie zadeklarować jakie wyjątki one zgłaszają.

W klasie `LiczbaCalk` zdefiniuj operatory umożliwiające wykonywanie obliczeń arytmetycznych (dodawanie, odejmowanie, mnożenie, dzielenie, reszta z dzielenie i zmiana znaku). Pamiętaj, że klasa reprezentująca długą liczbę może być dość duża i nie powinno się (o ile to nie jest konieczne) przekazywać jej przez wartość za pomocą stosu. Można więc zdefiniować po dwie wersje każdego operatora arytmetycznego: jeden jako funkcja zaprzyjaźniona a drugi jako składowy operator przypisania. Do obliczeń arytmetycznych mogą ci się przydać operatory bitowe oraz przesunięcia.

```
class LiczbaCalk
{
// ...
public:
    LiczbaCalk operator- () throw ();
    friend LiczbaCalk operator+ (const LiczbaCalk &x, const LiczbaCalk &y) throw ();
    LiczbaCalk & operator+= (const LiczbaCalk &y) throw ();
    friend LiczbaCalk operator/ (const LiczbaCalk &x, const LiczbaCalk &y) throw (div0);
    LiczbaCalk & operator/= (const LiczbaCalk &y) throw (div0);
// ...
};
```

W trakcie obliczeń obliczeń arytmetycznych (w szczególności przy mnożeniu) wartość liczby może szybko rosnąć — powinieneś o tym pamiętać, aby używać odpowiednio dużej tablicy bitów do przechowywania liczby, a w razie konieczności tablice taką należy powiększyć (albo pomniejszyć). Przy dzieleniu zwróć uwagę na wartość dzielnika i w przypadku próby dzielenia przez 0 zgłoś wyjątek `div0`. Klasę wyjątku `div0` zdefiniuj samodzielnie dziedzicząc po `exception` z biblioteki standardowej.

Zaprogramuj także zaprzyjaźnione operatory do czytania ze stumienia wejściowego `operator>>` i pisania do strumienia wyjściowego `operator<<` długich liczb całkowitych w postaci dziesiętnej.

```
class LiczbaCalk
{
// ...
public:
    friend istream operator>> (istream &we, LiczbaCalk &x);
    friend ostream operator<< (ostream &wy, const LiczbaCalk &x);
};
```

Definicję klasy `LiczbaCalk` umieść w przestrzeni nazw `obliczenia`. Potem stwórz bibliotekę statyczną albo współdzieloną z obydwiema klasami i nazwij ją `liczby.lib` pod Windowsem albo `libliczby.a` pod Linuxem.

Uzupełnienie.

Na koniec napisz program, który rzetelnie przetestuje wszystkie metody w klasie `LiczbaCalk`. Uzupełnieniem programu testowego niech będzie funkcja rekurencyjna, która wylicza wartość zadanej liczby Catalana. W swoim programie testowym wylicz i wypisz 100-tną liczbę Catalana.

Uwaga.

Nie używaj w swoim kodzie globalnej dyrektywy `using namespace`.

Wskazówka.

Każdy n–ty wyraz ciągu Catalana określony jest wzorem jawnym $C_n = \frac{1}{n+1}\binom{2n}{n}$ dla $n \geq 0$. Rekurencyjnie ciąg jest określony w następujący sposób:

$$\begin{aligned} C_0 &= 1 \\ C_n &= \tfrac{4n-2}{n+1} C_{n-1} \quad \text{dla } n > 0 \end{aligned}$$