Projekt zaliczeniowy

Liczenie całki ze stringa

Igor Skiba
Bartosz Kirker
Daniel Szura

ONP

- **Odwrotna notacja polska** – sposób zapisu wyrażeń arytmetycznych, w którym znak wykonywanej operacji umieszczony jest po operandach (zapis postfiksowy), a nie pomiędzy nimi jak w konwencjonalnym zapisie algebraicznym (zapis infiksowy) lub przed operandami jak w zwykłej notacji polskiej (zapis prefiksowy). Zapis ten pozwala na całkowitą rezygnację z użycia nawiasów w wyrażeniach, jako że jednoznacznie określa kolejność wykonywanych działań.

ONP

```
(2+3)*5
```

w ONP wygląda tak:

```
2 3 + 5 *
```

natomiast:

```
((2+7)/3+(14-3)*4)/2
```

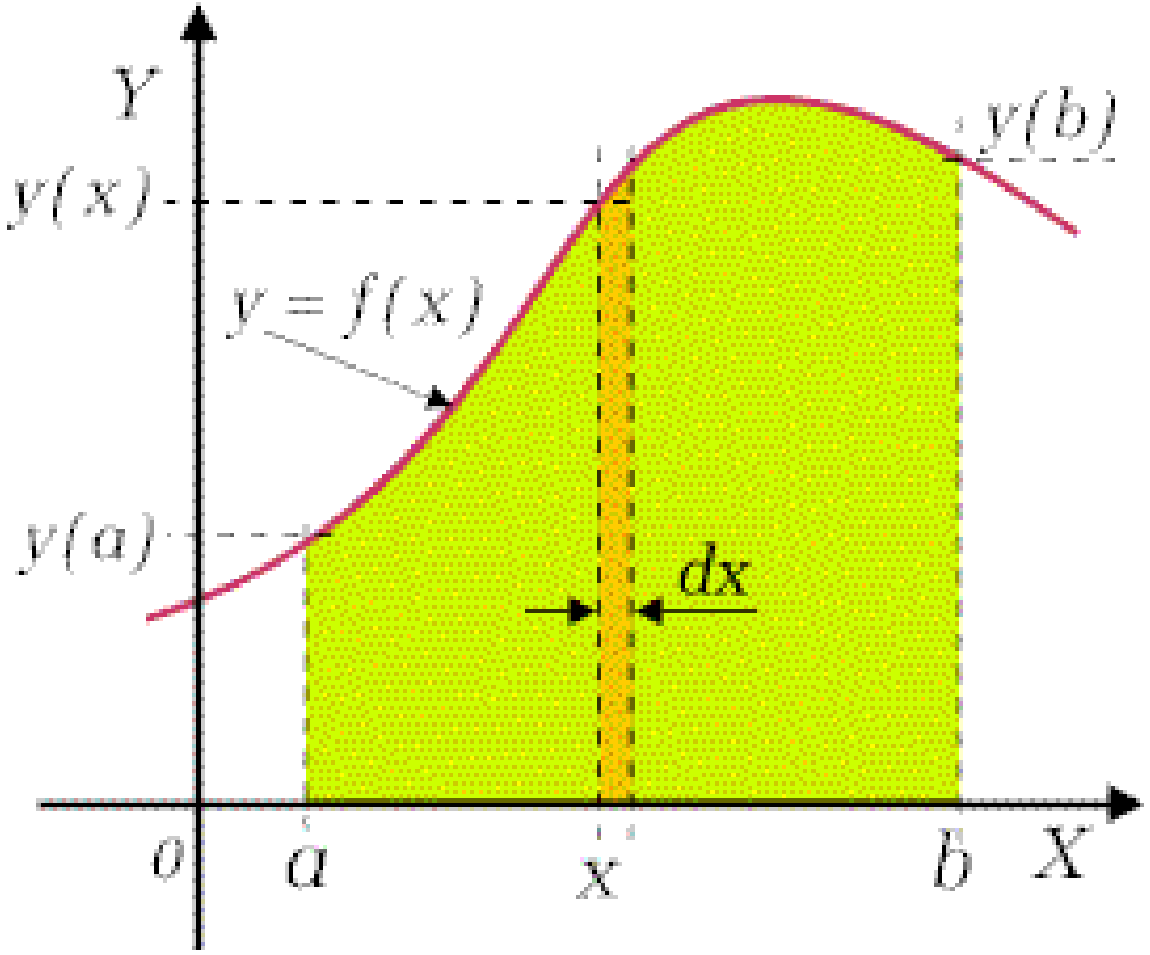
zapisuje się następująco:

```
2 7 + 3 / 14 3 - 4 * + 2 /
```

Metody Monte-Carlo

- **Metoda Monte Carlo** jest stosowana do modelowania matematycznego procesów zbyt złożonych aby można było przewidzieć ich wyniki za pomocą podejścia analitycznego. Istotną rolę w metodzie MC odgrywa losowanie wielkości charakteryzujących proces, przy czym losowanie dokonywane jest zgodnie z rozkładem, który musi być znany.

Całka



Całki policzone metoda analityczną Wolfram|Alpha

- X =10,Y= 10,
- Cos(x) = - 0,529919 + c,
- X+Y = 1000 + c,
- X*Y = 2500 + c,
- Cos(x)+Cos(y) = -10,5983 + c,
- X*Y+2*X = 5836,1298 + c,

Całki policzone metodą numeryczną

- X=10, Y=10,
- X cos = - 0,53416
- X Y + = 999,602
- X Y * = 2500,8167
- X cos Y cos + = -10,9778
- X Y * 2 X * + = 5836,7314

Stos

- Do stworzenia metody która skonwertuje nam zapis infiksowy(algebraiczny) na postfiksowy(onp), użyliśmy stosu. Najpierw za pomocą „pop” zrzucamy pierwszy elementy ze stosu żeby je dodać za pomocą „push” dodajemy poprzedni element i aktualny na gorę stosu.

```
Stack ^stack = gcnew Stack();

if( Double::TryParse(w, wynik )== false ){
    if( w == "+" ){
        Op ^tmp = (Op^)stack->Peek();
        stack->Pop();
        Op ^tmp1 = (Op^)stack->Peek();
        stack->Pop();
        stack->Push(( gcnew Dodaj( tmp, tmp1 )));
```

Opis

Ilustracja

Fragment kodu
Metoda Monte – Carlo

```
double pole( Op ^f, int n, double xs, double xk, double ys, double yk ){
    int trafien = 0;
    double xl = xk - xs;
    double yl = yk - ys;
    double fs = 0;
    Random r;
    double dx = xk - xs;
    double dy = yk - ys;
    for( int i = 0; i < n; i++ ){
        double x = xs + r.NextDouble()*dx;
        double y = ys + r.NextDouble()*dy;
        fs += f->licz(x,y) / n;
    }
    return dx*dy * fs;
}
```

Fragment kodu
Klasa głowna + klasa pochodna

```
ref class Op : IDisposable{
    public:
        double virtual licz(double x, double y){
            return 0;
        }
        !Op(){
        }
        ~Op(){
        }
};
ref class Dodaj: public Op{
private:
    Op ^ x, ^ y;
public:
    Dodaj(Op ^ x, Op ^ y){
        this->x = x;
        this->y = y;
    }
    double virtual licz(double x, double y) override{
        double wynik = this->x->licz(x,y) + this->y->licz(x,y);
        //Console::WriteLine("dodaj"+ wynik);
        return wynik;
    }
};
```

Ciekawe elementy kodu

- Uchwyt„^” (hat) – modyfikuje typ specyfikator, aby wskazać, że deklarowany obiekt powinien być automatycznie usuwany, gdy system określi, że nie jest on już dostępny.
- gcnew – gcnew jest dla obiektów referencyjnych .NET ; obiekty tworzone są automatycznie z gcnew (garbage-collected)
- Ref class – pozwala przekazywanć dane między komponentami.

Koniec