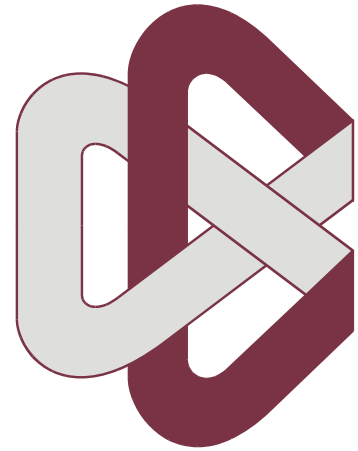


CIMAT

Optimización de código 3

Miguel Vargas-Félix

miguelvargas@cimat.mx
http://www.cimat.mx/~miguelvargas

OOP operator overloading

Estos dos códigos... ¿hacen lo mismo?

```
#include "Vector.h"
#include <stdlib.h>

int main(int argc, char** argv)
{
    if (argc != 2) return 1;
    int size = atoi(argv[1]);

    Vector A(size);
    Vector B(size);
    Vector C(size);
    double r = 1;

    C = -A + r*B;

    return 0;
}
```

```
make test1
time ./test1 50000000
```

```
#include "Vector.h"
#include <stdlib.h>

int main(int argc, char** argv)
{
    if (argc != 2) return 1;
    int size = atoi(argv[1]);

    Vector A(size);
    Vector B(size);
    Vector C(size);
    double r = 1;

    for (int i = 0; i < size; ++i)
    {
        C.data[i] = -A.data[i] + r*B.data[i];
    }
    return 0;
}
```

```
make test2
time ./test2 50000000
```

Vamos a correrlos y ver...

Sobrecarga de operadores

La sobreacarga de operadores permite escribir operaciones con objectos de forma más legible, por ejemplo, la operación

$$C = -A + r*B;$$

requiere tener sobrecargados los operadores: escalar por vector, negativo de un vector, suma de vectores y el operador igual.

Multiplicación escalar por vector

Se implementaría como:

```
Vector operator * (double a, const Vector& B)
{
    Vector C(B.size);
    for (int i = 0; i < C.size; ++i)
    {
        C.data[i] = a*B.data[i];
    }
    return C;
}
```

El resultado es un nuevo vector que contiene la multiplicación.

Negativo de un vector

```
Vector operator - (const Vector& A)
{
    Vector C(A.size);
    for (int i = 0; i < C.size; ++i)
    {
        C.data[i] = -A.data[i];
    }
    return C;
}
```

Se crea un nuevo vector que contine el negativo del vector A.

Suma de vectores

```
Vector operator + (const Vector& A, const Vector& B)
{
    Vector C(A.size);
    for (int i = 0; i < C.size; ++i)
    {
        C.data[i] = A.data[i] + B.data[i];
    }
    return C;
}
```

El resultado es un nuevo vector que contiene la suma de los vectores.

Operador igual

Éste es un miembro de la clase:

```
Vector& Vector::operator = (const Vector& A)
{
    if (size != A.size)
    {
        Resize(A.size);
    }
    for (int i = 0; i < size; ++i)
    {
        data[i] = A.data[i];
    }
    return *this;
}
```

Como se puede ver todos estos operadores tienen un ciclo **for** cada uno.

Las llamadas a los operadores serán generadas por el compilador al escribir la expresión

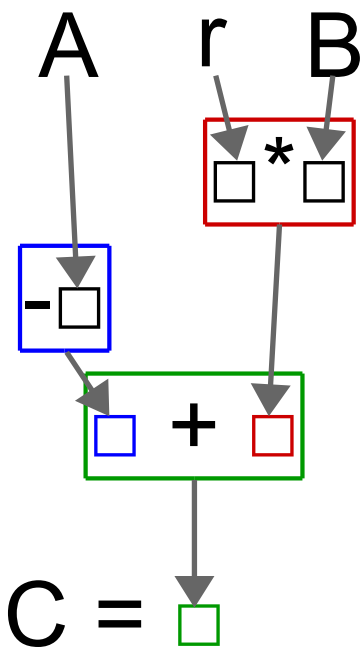
C = -A + r*B;

El costo de la sobrecarga de operadores

Veamos como se realiza la operación

$$C = -A + r*B;$$

La operación se hace en forma de árbol, primero los monomios y luego el polinomio



En este proceso se crean tres Vectores temporales, que guardan los resultados parciales:

□*□ se crea para guardar resultado de la multiplicación escalar vector.

-□ se crea para guardar resultado del operador menos al vector.

□ + □ se crea para guardar resultado de la suma de los vectores temporales anteriores.

Una vez que el resultado de la suma se asigna a C, estos tres vectores temporales son destruidos.

En total se hacen 3 alocaciones de memoria para 3 vectores temporales, es decir se utiliza el doble de memoria.

Para realizar la operación C = -A + r*B; se hacen 4 ciclos **for**: □*□, -□, □ + □, C = □.

En cambio, sin utilizar sobrecarga de operadores:

```
for (int i = 0; i < size; ++i)
{
    C.data[i] = -A.data[i] + r*B.data[i];
}
```

Sólo se utiliza un ciclo **for** y no se requiere reservar/liberar memoria extra.

¿Preguntas?

miguelvargas@cimat.mx

Referencias

The C++ FAQ: http://www.parashift.com/c++-faq/

C++ Frequently Questioned Answers: http://yosefk.com/c++fqa/